UML im Unterricht: Systematische objektorientierte Problemlösung mit Hilfe von Szenarien am Beispiel der Türme von Hanoi

Ira Diethelm[1], Leif Geiger[2], Albert Zündorf[2]

[1]Gaußschule-Gymnasium am Löwenwall
Löwenwall 18a, 38100 Braunschweig
i.diethelm@tu-bs.de

[2]IPS, TU-Braunschweig
Postfach 3329, 38106 Braunschweig
l3_g5@gmx.de, zuendorf@upb.de

Abstract: Dieses Papier berichtet über unsere Unterrichtserfahrungen mit Story Driven Modeling, einer neuen, systematischen Vorgehensweise zur Entwicklung von objektorientierten Programmen. Das Problem wird mit Hilfe eines Objektspiels erarbeitet, mit Hilfe von UML-Szenario-Diagrammen analysiert und dann systematisch in ein Programm überführt.

1 Einleitung

Im Informatikunterricht werden „Prinzipien, Konzepte und Strategien zur Planung, Konstruktion, Beschreibung und Bewertung abstrakter Informatiksysteme thematisiert“ [Hu00]. Wir wollen mit unserem Papier einen Vorschlag für eine Strategie zur systematischen Modellierung eines abstrakten Problems machen. Dieser Vorschlag beruht auf einem neuen Vorgehensmodell aus der Softwaretechnik, dem sogenannten *Story Driven Modeling* [KNNZ00, Zü01, DGMZ02]. Unser Beispielprojekt, das zur Zeit in einem Grundkurs der Klasse 12 am Beispiel der Türme von Hanoi läuft, besteht im wesentlichen aus den Teilschritten handlungsorientiertes *Objektspiel*, *Story-Boarding* und *Ableitung des Programms*. Diese Projektphasen und die darin enthaltenen systematischen Vorgehensweisen werden in den folgenden Abschnitten ausführlicher vorgestellt. Wir schließen dann mit einer Bewertung unserer bisherigen Unterrichtserfahrungen und einem Überblick über weitere Pläne.

2 Objektspiel

Beim Objektspiel erarbeiten die Schüler ein erstes Gefühl dafür, mit welchen Objektstrukturen sie ihre Aufgabe lösen können. Dabei wird, ausgehend von der realen Welt, mit Hilfe von Anschauungsmaterial das Problem *durchgespielt*. In unserem Beispielprojekt erhalten die Schüler eine kurze Einführung in die Geschichte der Türme von Hanoi und werden dann in Gruppen von 3-4 Schülern aufgeteilt. Jede Gruppe erhält 4 große Styroporscheiben und 3 Ablagescheiben aus Pappe und die Aufgabe, das Problem der Türme von Hanoi mit 1 bis 4 Scheiben zu lösen und die Lösung zu dokumentieren. Die Schüler probieren in dieser stark handlungsorientierten Phase verschiedene Strategien aus und erarbeiten sich so ein Grundverständnis für das zu lösende Problem und erste Ideen für mögliche Lösungsansätze.

Der erste Schritt zur Programmierung ist dann die Modellierung des konkreten Problems mit Hilfe von UML-Objektdiagrammen, vgl. Abbildung 1. Dabei gehen wir systematisch vor, indem zunächst die konkreten (Spielzeug-) Objekte, ausgehend von der Dokumentation der Schüler, benannt und an der Tafel in einem Objektdiagramm notiert werden. Im Fall der Türme von Hanoi sind dies die verschiedenen Scheiben *größteScheibe*, *nächstKleinere*, *überNächste* etc.

Abbildung 1: erstes Objektdiagramm für das Türme von Hanoi Szenario

Auf der Basis der identifizierten Objekte werden dann die Beziehungen zwischen den Objekten herausgearbeitet. Im Beispiel der Türme von Hanoi ergibt sich die *liegt auf* Beziehung zwischen den Scheiben sehr leicht aus der Problemstellung. Bei komplexeren Beziehungen zwischen den Objekten oder bei nicht so naheliegenden abstrakten Hilfsobjekten kann ein intensives Objektspiel mit CRC-ähnlichen Objektkarten zur Visualisierung herangezogen werden, vgl. das Beispiel Flaschendrehen in [life02, SN02]. Es kann dabei die Schwierigkeit auftreten, dass von den Schülern nicht realisiert wird, dass das Objekt, das sie im Spiel repräsentieren, eine auf die Assoziationen, Attribute und Methoden eingeschränkte Kenntnis der Modellwelt besitzt. Dies kann aber durch reale einschränkende Mittel wie das Verbinden der Augen der Schüler erfahrbar gemacht werden.

Um die Lösungsmethode aufrufen zu können, und auch für ihre Herleitung, ist es in unserem Beispiel sinnvoll, ein zentrales *hanoi* Objekt einzuführen. Es ist denkbar, das Beispiel ohne dieses zusätzliche Wurzelobjekt erfolgreich zu modellieren. Unsere Erfahrung zeigt jedoch, dass viele Probleme leichter zu modellieren sind, wenn die verwendete Objektstruktur eine Zusammenhangskomponente bildet, in der jedes Objekt von jedem anderen aus über eine Folge von Links erreicht werden kann. Hierzu hat sich die Einführung von zusätzlichen Wurzelobjekten bewährt. In unserem Beispielprojekt haben die Schüler sogar selbst ein *hanoi* Objekt vorgeschlagen, das mindestens Links zu allen Ablageflächen besitzt, vgl. Aktivität *ESA* in Abbildung 2.

3 Story-Boarding

Für das weitere Vorgehen müssen nun die Arbeitsschritte des Objektspiels und die dabei entstehenden Objektstrukturen detailliert protokolliert werden. Hierfür

verwenden wir in unserem Ansatz sogenannte Story-Boards, [Zü01]. Story-Boards sind eine Variante von UML Aktivitätsdiagrammen, bei denen die einzelnen Aktivitäten durch spezielle Objektdiagramme, sogenannte Schnappschüsse, beschrieben werden. Abbildung 2 zeigt ein einfaches Story-Board für das Türme von Hanoi Problem bei nur einer Scheibe. Die erste Aktivität *ESA* in Abbildung 2 enthält ein normales Objektdiagramm, dass die Ausgangssituation modelliert. Wir benutzen ein *hanoi* Objekt, dem die Objekte *start*, *hilfs* und *ziel* gehören. Auf dem *start* Objekt liegt eine Scheibe mit Namen *größteScheibe*.

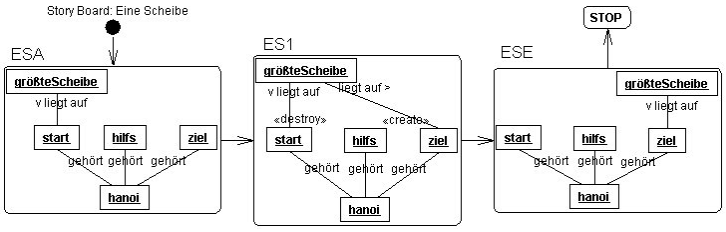

Abbildung 2: Türme von Hanoi Szenario für eine Scheibe

In der Aktivität *ES1* von Abbildung 2 wird modelliert, wie die Scheibe vom *start* zum *ziel* Objekt befördert wird. In unserer Modellierung geschieht dies durch das Löschen des *liegt auf* Links zwischen *größteScheibe* und *start* und durch das Erzeugen eines neuen *liegt auf* Links zwischen *größteScheibe* und *ziel*. Damit solche Veränderungen der Objektstruktur einfach modelliert werden können, erlauben Story-Boards, dass in einem Objektdiagramm zu löschende Objekte oder Links mit dem üblichen UML Marker $\ll destroy \gg$ gekennzeichnet werden. Zu erzeugende Objekte oder Links werden mir dem UML Marker $\ll create \gg$ gekennzeichnet. Zusätzlich können innerhalb der Objekte aktuelle Attributbelegungen in der Form *attrName == Wert* modelliert werden und Attributveränderungen in der Form *attrName := NeuerWert*. Dies wird in unserem Beispiel aber nicht benötigt. Die letzte Aktivität *ESE* von Abbildung 2 zeigt die Ergebnissituation unseres Beispielszenarios an: die große Scheibe wurde auf das Ziel verschoben.

Abbildung 3 zeigt nun das Story-Board für den nächst komplizierteren Fall, die Verschiebung von zwei Scheiben. Die Ausgangssituation unterscheidet sich von dem ersten Szenario durch eine weitere Scheibe mit Namen *nächstKleinere*, die auf der Scheibe *größteScheibe* liegt, vgl. erste Aktivität *ZSA* in Abbildung 3. Um die größte Scheibe bewegen zu können, muss in der zweiten Aktivität *ZS1* von Abbildung 3 zunächst die nächst kleinere Scheibe auf das *hilfs* Objekt verschoben werden. In der Aktivität *ZS2* kann dann die größte Scheibe wie gewohnt auf das Ziel verschoben werden und danach kann die nächst kleinere Scheibe in Aktivität *ZS3* wieder auf die größte Scheibe zurück gelegt werden.

Die Erstellung von Story-Boards kann sehr gut in der häuslichen Nacharbeit zur Vertiefung der Erfahrungen aus dem Objektspiel eingesetzt werden, nachdem sie

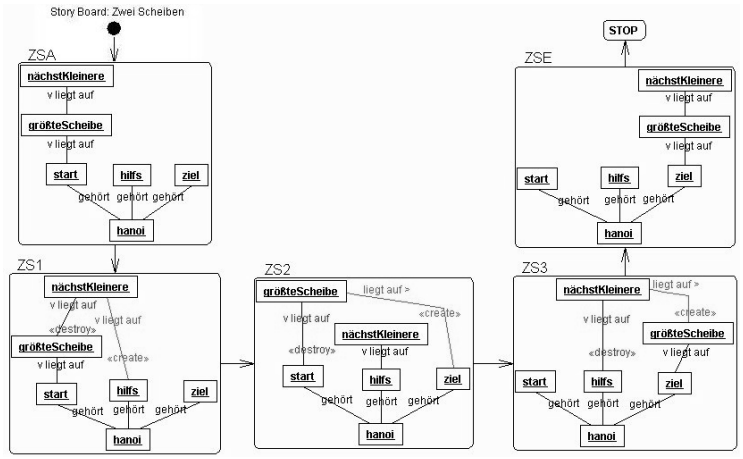

Abbildung 3: Türme von Hanoi Szenario für zwei Scheiben

im Unterricht gemeinsam für ein Beispiel durchgeführt wurde. Während des Objektspiels machen sich die Schüler geeignete Notizen und die ersten Entwürfe für Objektdiagramme werden gemeinsam an der Tafel erarbeitet.

Man beachte, dass Story-Boards in unserem Ansatz nur dazu dienen, beispielhafte Objektspiele zu protokollieren. Das heißt, Story-Boards modellieren bisher nur mögliche Ablaufszenarien. Sie modellieren nicht den allgemeinen Fall. Prinzipiell erlauben die zu Grunde liegenden Aktivitätsdiagramme auch die Modellierung von Fallunterscheidungen und Schleifen und damit die Behandlung von Sonderfällen und Ausnahmen. Die Betrachtung solcher Spezialfälle in dieser frühen Problemlösungsphase überfordert die Schüler jedoch erfahrungsgemäß. Daher werden diese eher algorithmischen Aspekte erst in der nächsten Phase, also bei der Ableitung von Methoden-Implementierungen aus den Beispielszenarien, behandelt.

4 Ableitung des Programms

Nachdem das Problem ausreichend mit Hilfe von Story-Boards und Szenarien verstanden und protokolliert worden ist, kann in der nächsten Projektphase die systematische Ableitung eines Programmes beginnen. Dies umfasst die Ableitung eines Klassendiagramms und die Ableitung der verwendeten Methoden.

Ableitung des Klassendiagramms: Das Klassendiagramm entsteht in einem fortlaufenden Prozess, in dem es immer mehr verfeinert wird. Die erste Fassung des Klassendiagramms entsteht schon während des Story-Boardings. Wird ein Objekt einer noch nicht bekannten Klasse in einem Story-Board benutzt, wird diese Klasse in das Klassendiagramm eingetragen. Eine Attributwertzuweisung bzw. -überprüfung in einem Story-Board führt zu einer Attributdeklaration in der da-

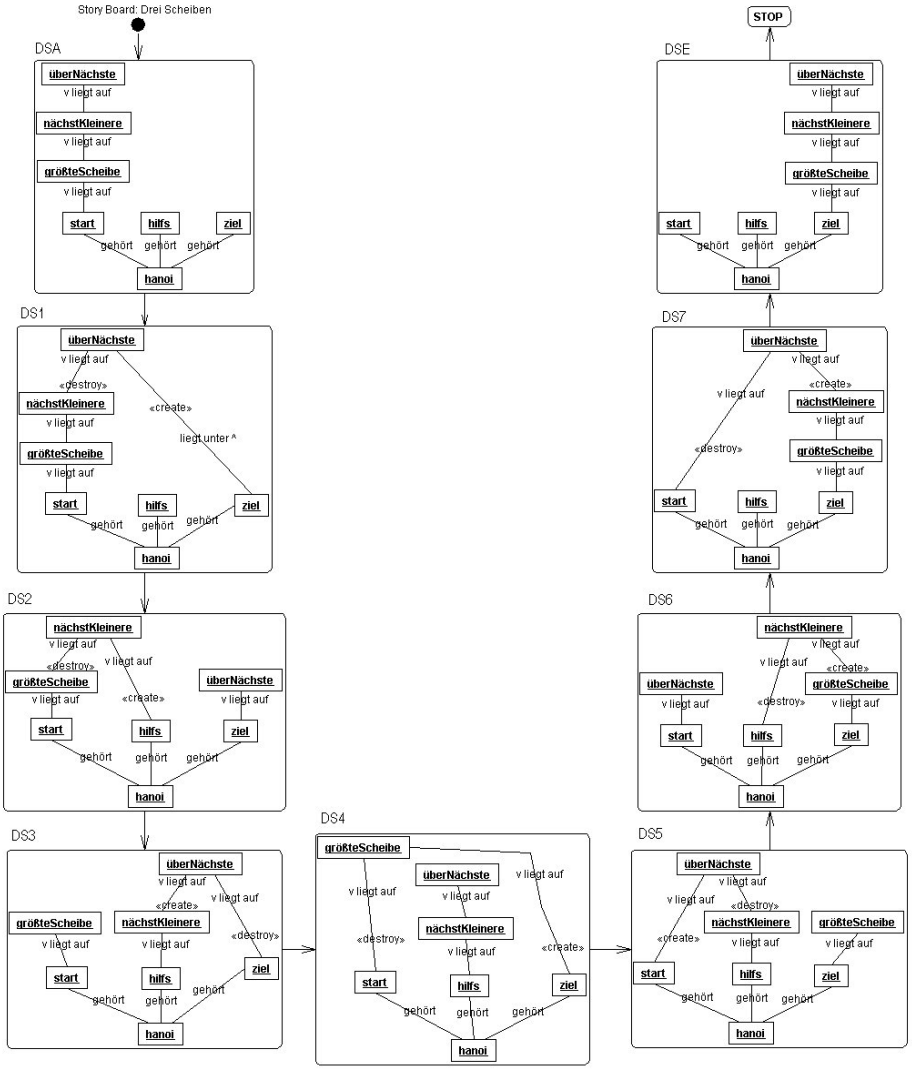

Abbildung 4: Türme von Hanoi Szenario für drei Scheiben

zugehörigen Klasse. Bei einem Methodenaufruf wird die Methode im Klassendiagramm eingetragen und ein Link im Story-Board resultiert in einer Assoziation zwischen den zugehörigen Klassen.

Betrachten wir das Objektdiagramm *ZSA* in Abbildung 3: Hier ist die Entscheidung, zu welcher Klasse die einzelnen Objekte gehören, noch nicht getroffen (bzw. nicht erkennbar). Man muss sich also Gedanken über die Zuordnung zu Klassen ma-

chen: Es ist sicherlich sinnvoll, eine Klasse *Scheibe* einzuführen, zu der die Objekte *nächstKleinere* und *größteScheibe* gehören. Desweiteren benötigt das Zusammenhangsobjekt *hanoi* eine eigene Klasse, nennen wir sie *Hanoi*. Die Objekte *start*, *hilfs* und *ziel* repräsentieren jeweils eine Ablagefläche, sollten also Instanz einer gemeinsamen Klasse *Ablage* sein. Die Einführung einer solchen Klasse *Ablage* führt aber später zu vielen umständlichen Fallunterscheidungen, ob eine Scheibe auf einer anderen Scheibe oder auf einer Ablage liegt. Um dies zu vermeiden, wurde in unserem Beispiel die Designentscheidung getroffen für die Ablageflächen auch die Klasse *Scheibe* zu verwenden. Diese Entscheidung ist keineswegs trivial und muss den Schülern durch Hilfen (die Ablageflächen sind im Objektspiel auch große Scheiben) nahe gelegt bzw. vorgegeben werden. Die Links zwischen *hanoi* und den Objekten *start*, *hilfs* und *ziel* resultieren in einer Assoziation zwischen den Klassen *Hanoi* und *Scheibe*. Der Link zwischen *start* und *größteScheibe* führt zu einer Selbstassoziation der Klasse *Scheibe*. Dies ist ebenfalls nicht trivial, jedoch ist es bei unserer Vorgehensweise aus den Story-Boards ersichtlich.

In diesem Beispiel ändert die Analyse der restlichen Objektdiagramme nach der selben Vorgehensweise nichts mehr am resultierenden Klassendiagramm (vgl. Abbildung 5). Nach dem Story-Boarding müssen noch die Kardinalitäten der Assoziationen und die Typen der Attribute überprüft werden. Es gilt folgende Faustregel: Jede Assoziation ist zunächst vom Typ 1-zu-1. Existiert ein Objektdiagramm, in dem ein Objekt mehrere Links einer 1-zu-1 Assoziation hat, wird diese zu einer 1-zu-n Assoziation (vgl. Aktivität *ESA*: das Objekt *hanoi* besitzt drei *gehört* Links). Auf dieselbe Art können auch m-zu-n Assoziationen entstehen. Unklarheiten müssen durch Vergleiche mit dem Objektspiel beseitigt werden (Kommt es vor, dass mehrere Scheiben auf einer Scheibe liegen?).

Im nächsten Schritt, der im folgenden Kapitel vorgestellt wird, werden alle benutzten Methoden mitsamt der Rückgabetypen und Parameter ins Klassendiagramm eingetragen.

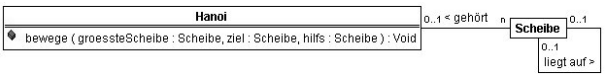

Abbildung 5: Abgeleitetes Klassendiagramm für die Türme von Hanoi

Ableitung der Methoden: Nun folgt der zentrale Schritt unseres Vorgehens. In diesem Abschnitt soll die Ausformulierung des Methodenrumpfes aus den Beispielen und Story-Boards abstrahiert werden. Zum besseren Verständnis möchten wir mit der Spezifikation der Methode beginnen, bevor wir die Herleitung betrachten.

Am Ende unserer Vorgehensweise entsteht die folgende Methode *bewege()*. Abbildung 6 zeigt das *Storydiagramm*, in dem die Lösungsmethode durch ein Aktivitätsdiagramm mit eingebetteten Kollaborationsdiagrammen implementiert ist. Fujaba generiert aus diesen Diagrammen Java Code, der genau die dargestellten Schritte vollautomatisch ausführt, vgl. [Fu02, FNT98].

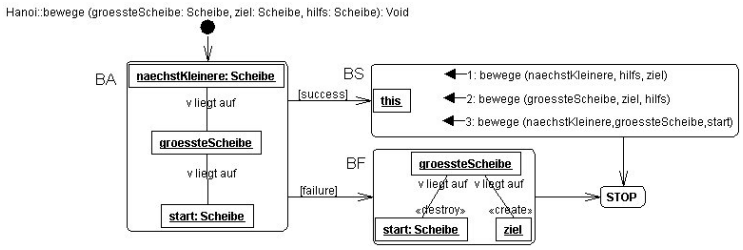

Abbildung 6: Abgeleitete Implementierung der Methode bewege

In dem Kollaborationsdiagramm der ersten Aktivität BA ist *groessteScheibe* bereits bekannt, da dieses Objekt als erster Parameter beim Methodenaufruf übergeben wird. Zur Unterscheidung von bereits bekannten Objekten, die direkt verwendet werden können, und noch unbekannten Objekten, die erst noch in der aktuellen Laufzeitobjektstuktur gefunden werden müssen, werden bekannte Objekte ohne Klassenangabe dargestellt. Ausgehend von dem bekannten Objekt *groessteScheibe* wird entlang der angegebenen Links die vorhandene Objektstruktur überprüft. So wird einerseits überprüft, ob es einen *liegt auf* Link zu einem Objekt der Klasse Scheibe gibt, das *start* genannt wird, und ob es ein weiteres Objekt dieser Klasse gibt, das einen *liegt auf* Link zu *groessteScheibe* besitzt und daher *naechstKleinere* genannt wird. In unserem Beispiel gibt es immer eine Scheibe, die unter einer zu bewegenden Scheibe liegt. Der Link zum *start* Objekt und dieses Objekt selbst sind also in jedem Fall vorhanden. Für den Link auf die nächst kleinere Scheibe gibt es zwei Fälle, die auftreten können:

Fall 1: Die größte zu bewegende Scheibe liegt wie beschrieben unter einer nächst kleineren Scheibe. Dann verläuft die Überprüfung der Objektstruktur erfolgreich. Die Aktivität BA wird daher über die *success* Kante verlassen und die Ausführung wird mit der Aktivität BS fortgesetzt.

Fall 2: Die größte zu bewegende Scheibe ist die oberste Scheibe. Dann gibt es keine, die darauf liegt, und die Überprüfung schlägt fehl. Die Aktivität BA wird über die *failure* Kante verlassen und die Ausführung wird mit der Aktivität BF fortgesetzt.

Im Fall 2 kann die größte zu bewegende Scheibe direkt auf das *ziel* bewegt werden. Dies geschieht, indem der bestehende *liegt auf* Link zu der Scheibe darunter zerstört und ein neuer *liegt auf* Link zum *ziel* erzeugt wird, vgl. Aktivität *BF*. Im Fall 1 müssen zunächst die nächst kleinere und alle darüberliegenden Scheiben auf einen Hilfsstapel bewegt werden (1:), bevor die größte zu bewegende Scheibe auf das Ziel bewegt werden kann (2:). Anschließend müssen die nächst kleinere und alle darüber liegenden vom Hilfsstapel auf die größte Scheibe gelegt werden (3:). Dies leisten die rekursiven Aufrufe auf dem *this* Objekt in der Aktivität BS. Bei dem ersten Aufruf dient das ursprüngliche *ziel* ggf. als Hilfsstapel und beim dritten die ursprüngliche

start Scheibe.

Für die Ableitung der Methode müssen die Änderungen der Objektstruktur analysiert werden, die in den Story-Boards protokolliert wurden. Hierzu wurde jede Aktivität in den Abbildungen 2 bis 4 wie folgt bezeichnet: Die ersten beiden Buchstaben geben das Szenario an, z.B. *ES* für *E*ine *S*cheibe. A bzw. E stehen für *A*nfangs- bzw. *E*nd-Situation, die Zahlen geben die Nummer des Änderungsschrittes an.

Als Erstes wird der einzige Änderungsschritt im ersten Szenario ES1 mit denen des zweiten verglichen. Die Schüler sollen die Änderungen aus ES1 wiedererkennen. Dabei ergibt sich einerseits, dass die Änderungen aus ES1 mit denen aus ZS2 identisch sind. Andererseits treten dieselben Änderungen auch in ZS1 und ZS3 auf, jedoch mit veränderten Start- und Ziel-Angaben: In ZS1 ist *ziel* mit *hilfs* und in ZS3 *start* mit *hilfs* und *ziel* mit *größteScheibe* vertauscht worden.

Im nächsten Schritt wird das dritte Szenario analysiert. Hier kann ES1 mit DS4 identifiziert werden. Ferner entsprechen alle anderen Änderungsschritte im dritten Szenario ebenfalls ES1 mit anderen Start- und Ziel-Angaben. In einem weiteren Analyseschritt lässt sich feststellen, dass DS1 bis DS3 außerdem dem zweiten Szenario entspricht, wobei *ziel* mit *hilfs* vertauscht wurde und hier *nächstKleinere* als größte zu bewegende Scheibe fungiert. DS5 bis DS7 entsprechen ebenfalls dem zweiten Szenario, wobei *start* mit *hilfs* und *ziel* mit *größteScheibe* vertauscht wurde.

Als zusätzliche Hilfe kann für ein oder mehrere Beispiele der vollständige Rekursionsbaum erarbeitet werden, der hier exemplarisch in Abbildung 7 für drei Scheiben angegeben ist. Die Zahl nach *bewege* dient als Hilfe und gibt die Anzahl der dabei zu bewegenden Scheiben an. Der Methodenaufruf *bewege3(größteScheibe, ziel)* (abgekürzt als *b3(gS,z)* zerfällt in folgende Schritte:

$$
b3(gS,z)\begin{cases} b2(nK,h)\begin{cases} b1(uN,z) \\ b1(nK,h) \\ b1(uN,nK) \end{cases} \\ b1(gS,z) \rightarrow b1(gS,z) \\ b2(nK,gS)\begin{cases} b1(uN,s) \\ b1(nK,gS) \\ b1(uN,nK) \end{cases} \end{cases}
$$

Abbildung 7: Rekursionsbaum für das Szenario mit drei Scheiben

Dies lässt sich nun auf das (nicht abgebildete) Szenario mit 4 Scheiben übertragen und für beliebig viele Scheiben verallgemeinern. Die Lösungsmethode *bewege()* für n Scheiben, in der die größte Scheibe mit allen darüber liegenden Scheiben auf *ziel* bewegt wird, besteht demzufolge aus drei Teilen:

1: In den ersten Schritten werden die nächst kleinere Scheibe und alle darüber liegenden, d.h. $n-1$ Scheiben, auf *hilfs* bewegt, wobei ggf. *ziel* als Hilfsablage

dient: *bewege(naechstKleinere, hilfs)*.

2: Die größte zu bewegende Scheibe wird im mittleren Schritt auf *ziel* bewegt, wenn keine Scheiben mehr auf ihr liegen. Eine Hilfsablage ist hier nicht nötig: *bewege(groessteScheibe, ziel)*.

3: In den letzten Schritten werden alle $n-1$ Scheiben, die nun auf *hilfs* liegen, nämlich die nächst kleinere und alle darüber liegenden, auf die größte Scheibe bewegt, wobei ggf. das ursprüngliche *start* als Hilfsablage dient:
bewege(naechstKleinere, groessteScheibe).

Die Methode *bewege()* benötigt offensichtlich mindestens zwei Parameter: die größte zu bewegende Scheibe und das Ziel. Allerdings wird auch die Hilfsablage benötigt. Sie könnte zwar während der Ausführung bei jedem Schritt bestimmt werden, aber dies würde für die Schüler eine sehr große Schwierigkeit darstellen, so dass es einfacher ist, die Hilfsablage als zusätzlichen Parameter zu übergeben. Daraus ergeben sich die drei folgenden rekursiven Aufrufe:

1. *bewege(nächstKleinere, hilfs, ziel)*
2. *bewege(größteScheibe, ziel, hilfs)*
3. *bewege(nächstKleinere, groessteScheibe, start)*

So entsteht also durch die systematische Analyse der Story-Boards und durch die Identifikation von gleichartigen Änderungsschritten und -schrittfolgen in für die Schüler nachvollziehbarer Weise die Implementierung des gesuchten Programms.

5 Zusammenfassung und Ausblick

Dieses Papier stellt Story Driven Modeling als Vorgehensmodell für den Informatik Unterricht vor. Das Problem wird in einem Objektspiel analysiert. Die Schritte des Objektspiels werden in sogenannten Story-Boards protokolliert. Aus diesen Story-Boards wird in einem einfachen Schritt das Klassendiagramm abgeleitet. Schließlich werden im wichtigsten Schritt durch die Analyse der Story-Boards systematisch die Implementierungen der verwendeten Methoden erarbeitet.

Nach unseren Erfahrungen hat sich Story Driven Modeling in der Schule sowohl am Beispiel der Türme von Hanoi als auch am Beispiel des Flaschendrehen Programms bewährt. Darüberhinaus wurde Story Driven Modeling seit dem Wintersemester 2000/2001 mit großem Erfolg in einer Reihe von Grundstudiumsveranstaltungen an der TU-Braunschweig und an der Universität Paderborn eingesetzt. Aufgrund dieser durchweg positiven Erfahrungen empfehlen wir diese Vorgehensweise für alle Arten von objektorientierten Problemstellungen sowohl im Unterricht als auch in der allgemeinen Programmierpraxis.

In der Zukunft werden wir Story Driven Modeling in der Schule für weitere Beispiel-Probleme einsetzen. Hier ist sowohl an klassische Probleme aus dem Bereich Algorithmen und Datenstrukturen gedacht, also Sortierverfahren und verschiedene Datenstrukturen, als auch an weitere praktische Probleme wie z. B. Aufzugsteuerungen. In einem speziell geförderten Formel-X Projekt sollen Schüler im September 2002 das Türme von Hanoi Problem mit Hilfe von Lego Mindstorms Robotern in einer realen Umgebung lösen. Auch für die praktischen Probleme sind solche Lego Mindstorms Umsetzungen geplant.

Die besonderen Stärken von Story Driven Modeling liegen in den anschaulichen Problemlösungsschritten der ersten Phasen, wo stark handlungsorientiert auf der Objekt-Ebene gearbeitet wird. Dies ist nach unseren Erfahrungen gerade für Anfänger und Schüler deutlich einfacher als die sonst übliche Vorgehensweise, bei der sehr früh Klassendiagramme und Methoden erstellt werden. Story Driven Modeling ist damit der erste Ansatz, der eine konkrete inhaltliche Problemlösungsstrategie anbietet, die die schwierige Lücke zwischen einer Problemstellung in der realen Welt und der Problemlösung in einem Programm systematisch überbrückt.

Literatur

[DGMZ02] I. Diethelm, L. Geiger, T. Maier, A. Zündorf: Turning Collaboration Diagram Strips into Storycharts; Workshop on Scenarios and state machines: models, algorithms, and tools; ICSE 2002, Orlando, Florida, USA, 2002.

[FNT98] T. Fischer, J. Niere, L. Torunski: Konzeption und Realisierung einer integrierten Entwicklungsumgebung für UML, Java und Stroy-Driven-Modeling, Diplomarbeit bei A. Zündorf, Universität-Gesamthochschule Paderborn, 1998.

[Fu02] Fujaba Homepage, Universität Paderborn, http://www.fujaba.de/.

[Hu00] P. Hubwieser: Didaktik der Informatik - Grundlagen, Konzepte, Beispiele, Springer Verlag, Berlin, 2000.

[KNNZ00] H. Köhler, U. Nickel, J. Niere, A. Zündorf: Integrating UML Diagrams for Production Control Systems; in Proc. of ICSE 2000 - The 22nd International Conference on Software Engineering, June 4-11th, Limerick, Ireland, acm press, pp. 241-251 (2000)

[life02] $life^3$-Homepage, Universität Paderborn, http://life.uni-paderborn.de/.

[SN02] C. Schulte, J. Niere: Thinking in Object Structures: Teaching Modelling in Secondary Schools; in Sixth Workshop on Pedagogies and Tools for Learning Object Oriented Concepts, ECOOP, Malaga, Spanien, 2002.

[Zü01] A. Zündorf: Rigorous Object Oriented Software Development, Habilitation Thesis, University of Paderborn, 2001.