Rettet Prinzessin Ada: Am leichtesten objektorientiert

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

[1]Gaußschule
Löwenwall 18a
D-38100 Braunschweig

[2]SE, Universität Kassel
Wilhelmshöher Allee 73
D-34121 Kassel

(ira.diethelm | leif.geiger | albert.zuendorf)@uni-kassel.de

http://www.se.eecs.uni-kassel.de/se/

Abstract: Dieser Beitrag diskutiert einen Artikel aus der GI-Zeitschrift Log In Nr. 128/29 (2004) [He04], der in einem Abschnitt den Nutzen objektorientierter Konzepte für die schulische Ausbildung stark hinterfragt. Wir stellen hier eine alternative, unserer Ansicht nach für Schüler geeignetere Herangehensweise im Sinne des „Objects first"-Ansatzes vor. Der LogIN-Artikel zeigt eine objektorientierte Lösung einer anspruchsvollen Wege-Such-Aufgabe. Sie verwendet die objektorientierten Konzepte dabei vor allem zur Strukturierung des Suchalgorithmus. Es wird dort kritisiert, dass die Vielzahl der syntaktischen Konstrukte, die für eine solche Strukturierung der Algorithmusimplementierung benötigt wird, bei den eher kleinen Aufgabenstellung im schulischen Bereich nicht gerechtfertig ist. Dem stimmen wir zu. Wir halten objektorientierte Konzepte für den Anfangsunterricht aber trotzdem für sehr wichtig. Die Stärke von objektorientierten Konzepten für den Anfangsunterricht liegt unserer Ansicht nach dabei aber nicht auf der Strukturierung von Quelltexten, sondern auf der Modellierung komplexer Daten. Wie wir dies in den Vordergrund stellen und damit eine, wie wir finden, einfachere objektorientierte Lösung Schritt für Schritt mit Schülern herleiten können, stellen wir in diesem Beitrag vor.

1 Einleitung

Der Artikel „Möglichkeiten und Grenzen maschineller Intelligenz", erschienen in der GI-Zeitschrift Log In Nr. 128/29 (2004), Seite 66 ff. [He04], stellt anhand eines Beispieles objektorientierte und prozedurale Programmierung aus Sicht der Modellierung gegenüber. In diesem Artikel zitiert der Autor Röhner [Rö03]:

> Modellierungstechniken sind dann wertvoll, wenn sie ein breites Anwendungsspektrum aufweisen, also zur Lösung einer großen Klasse von Problemen eingesetzt werden können.

[He04] erkennt den Nutzen der Objektorientierung zum Lösen großer komplexer Probleme in der Softwareentwicklung an, bezweifelt aber den Nutzen von Objektorientierung

zum Lösen der im Informatikunterricht auftretenden eher kleinen Problemklassen. Hierfür macht er das „aufwendige Begriffssystem“ der Objektorientierung verantwortlich, das er zum Lösen kleinerer Probleme als unnötigen Overhead darstellt. Auch der Einsatz von Werkzeugen wie BlueJ soll hierbei keine Vereinfachung bringen. Seine Argumentation führt der Autor unter anderem anhand einer prozeduralen und einer objektorientierten Lösung einer Aufgabe der polnischen Informatik-Olympiade 2001/02. Die vorgestellte objektorientierte Lösung ist jedoch in unseren Augen die prozedurale Lösung erweitert um eine Einteilung in Klassen. Dies ist in der Tat bei kleinen Programmen meistens nicht sinnvoll. Dabei verliert sich [He04] in der Diskussion von Details, wie z. B. Sichtbarkeiten. Hieraus zieht der Artikel dann die Schlussfolgerung, dass eine objektorientierte Herangehensweise für das vorgestellte Problem nicht sinnvoll ist.

Wir werden in diesem Artikel eine alternative Lösung dieser Aufgabe vorstellen und daran zeigen, dass viele der „aufwendigen Begriffe“ der Objektorientierung im Schulunterricht nicht gebraucht werden. Durch didaktische Reduktion und Konzentration auf die wesentlichen Konzepte der Objektorientierung wird die vorgestellte Lösung aus unserer Sicht sogar einfacher als die prozedurale Lösung und somit für Schüler leichter nachvollziehbar. Auch werden wir Vereinfachungen aufzeigen, die durch den Einsatz von passenden Werkzeugen zu erreichen sind.

2 Die Aufgabe

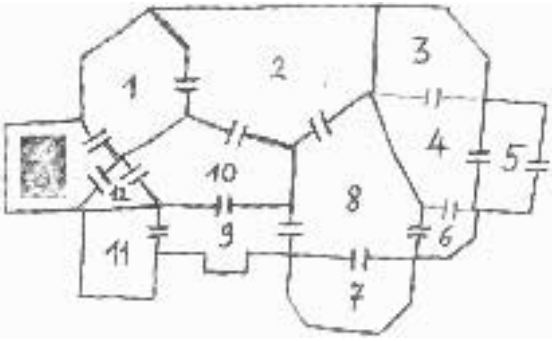

Abbildung 1: Grundriss des Schlosses (aus [He04])

Das Beispiel, das [He04] anführt, stammt aus der polnischen Informatik-Olympiade 2001/2002. In dieser Aufgabe geht es um ein Schloss, das aus mehreren Räumen besteht, die durch Flure miteinander verbunden sind. Jeder Raum kostet einen bestimmten Betrag Eintritt. In einem Zimmer befindet sich Prinzessin Ada. Aufgabe ist es nun, dem armen, vor dem Schloss wartenden Prinz Minichip einen Tipp zu geben, wie er zu Prinzessin Ada kommt und dabei sein Anfangsvermögen exakt ausgibt. Abbildung 1 zeigt einen möglichen Grundriss des Schlosses. Die Raumnummern sind hier gleichzeitig die Eintrittspreise.

3 Datenmodellierung

3.1 Lösung nach Heubaum

Betrachten wir nun zunächst Heubaums Lösungsansatz: Er modelliert den Grundriss als Adjazenzmatrix.

```
int[][] rp =
 {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  0,  0,  0,  0},
  {0, 0, 2, 0, 0, 0, 0, 0, 0, 0,  0,  0,  0, 13},
  {0, 1, 0, 0, 0, 0, 0, 0, 8, 0, 10,  0,  0,  0},
  {0, 0, 0, 0, 4, 0, 0, 0, 0, 0,  0,  0,  0,  0},
  {0, 0, 0, 3, 0, 5, 6, 0, 0, 0,  0,  0,  0,  0},
  {0, 0, 0, 0, 4, 0, 0, 0, 0, 0,  0,  0,  0,  0},
  {0, 0, 0, 0, 4, 0, 0, 0, 8, 0,  0,  0,  0,  0},
  {0, 0, 0, 0, 0, 0, 0, 0, 8, 0,  0,  0,  0,  0},
  {0, 0, 2, 0, 0, 0, 6, 7, 0, 9,  0,  0,  0,  0},
  {0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 10, 11,  0,  0},
  {0, 0, 2, 0, 0, 0, 0, 0, 0, 9,  0,  0, 12,  0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 9,  0,  0,  0,  0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10,  0,  0, 13},
  {0, 1, 0, 0, 0, 0, 0, 0, 0, 0,  0,  0, 12,  0}};
```

Zur Lösung der Aufgabe modelliert Heubaum den Zustand als Folge bisher besuchter Räume inklusive dem aktuellen Inhalt der Geldbörse des Prinzens. Ein Zug ist dann als Wahl des als nächstes zu betretenen Raums modelliert. Man beachte, dass Heubaum die Folge besuchter Räume als Zeichenkette darstellt. Dies kann leicht zu weiteren Modellierungsproblemen führen: Was verwende ich als Trennzeichen bei mehrstelligen Raumnummern? Wie füge ich Räume in die Folge ein und vor allem, wie bekomme ich sie wieder heraus? Letzteres ist nur durch mehr oder weniger komplizierte String-Manipulationen möglich. Heubaum kann dies in seiner Lösung allerdings durch eine rekursive Lösung des Problems umgehen.

Als objektorientierte Lösung stellt Heubaum im Wesentlichen die oben dargestellte imperative Lösung plus eine gewisse Einteilung in Klassen vor. Er führt eine Klasse *Spielposition* für den Zustand, eine Klasse *Spielzug* für den nächsten Suchschritt, eine Klasse *Daten* zur Speicherung der Adjazenzmatritzen und eine Klasse *Baumdemo* zur Implementierung des Suchalgorithmus ein. Heubaum benutzt also von den objektorientierten Konzepten im Wesentlichen nur Klassen zur Strukturierung seines Programms. Die Datenmodellierung hingegen ist nicht objektorientiert. Heubaum benutzt hierfür weiterhin Adjazenzmatrizen. Um mehrere Grundrisse zu modellieren, verwendet Heubaum nicht etwa mehrere Instanzen der Klasse *Daten*, sondern führt für jeden Grundriss eine neue Subklasse *DatenX* ein. Abbildung 2 zeigt das zugehörige Klassendiagramm.

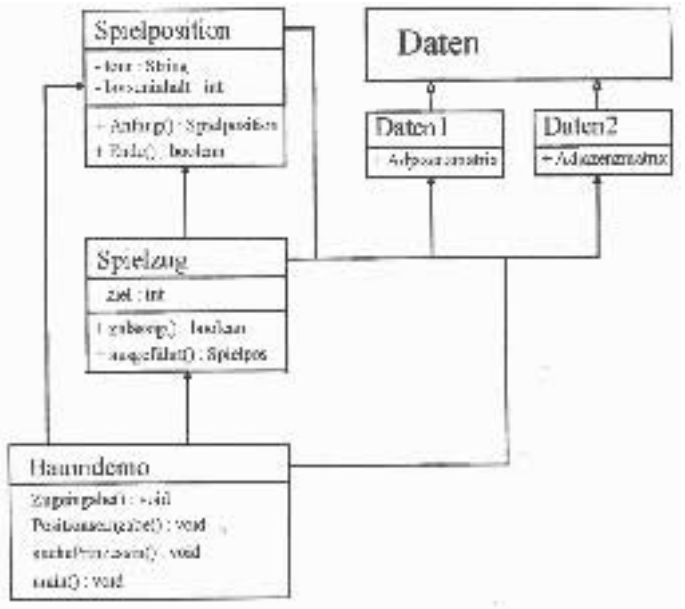

Abbildung 2: Klassendiagramm von Heubaum

3.2 Eine „objektorientiertere“ Lösung

Unser Verständnis von Objektorientierung setzt nicht die Programmstruktur, sondern die Datenstruktur in den Mittelpunkt, siehe auch [DGZ05]. Am Besten beginnt man bei unserem Ansatz nicht mit einem Klassendiagramm, sondern mit einem Objektdiagramm. Bei unserem Vorgehen lernen die Schüler also zuerst die Begriffe Objekt, Attribut und Beziehung/Link. Mit diesen drei Begriffen lässt sich bereits ein Modell für den Grundriss des Schlosses aus Abbildung 1 erstellen. Dieses lässt sich in Form eines UML Objektdiagramms formalisieren, siehe Abbildung 3. In Abbildung 3 sind die Räume als Objekte modelliert. Die Türen sind durch Beziehung *tuer* zwischen zwei Räumen abgebildet. Die Eintrittspreise werden durch das Attribut *kosten* angegeben. Ein solches Objektdiagramm lässt sich z. B. einfach gemeinsam mit den Schülern an der Tafel entwickeln.

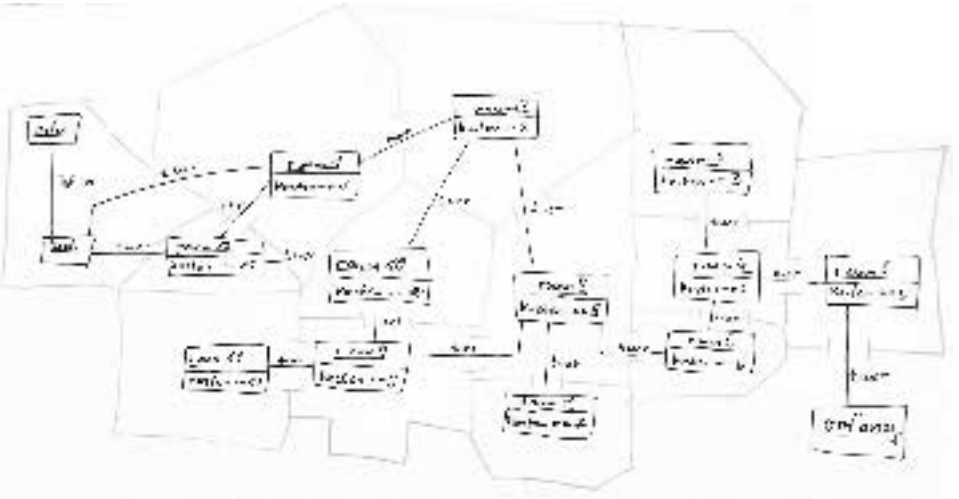

Abbildung 3: Objektdiagramm

Im Unterricht kann nun das Klassendiagramm als „Bauplan“ für Objektdiagramme eingeführt werden. Hierbei muss jedem Objekt eine passende Klasse zugeordnet werden. Objekte, die Attribute und Links mit gleichen Namen haben, werden der gleichen Klasse zugeordnet. Attribute im Objektdiagramm werden dann zu Attributen der zugehörigen Klasse, Links zwischen Objekten werden zu Assoziationen zwischen den zugeordneten Klassen. Lediglich über die Kardinalität der Assoziationen muss noch entschieden werden. Aus dem obigen Objektdiagramm lässt sich mit diesen Regeln ganz systematisch das Klassendiagramm in Abbildung 4 ableiten. Die Klasse *Besuch* dient zur Darstellung eines Suchschrittes und wird erst in der Verhaltensmodellierung (siehe nächstes Kapitel) hinzugefügt.

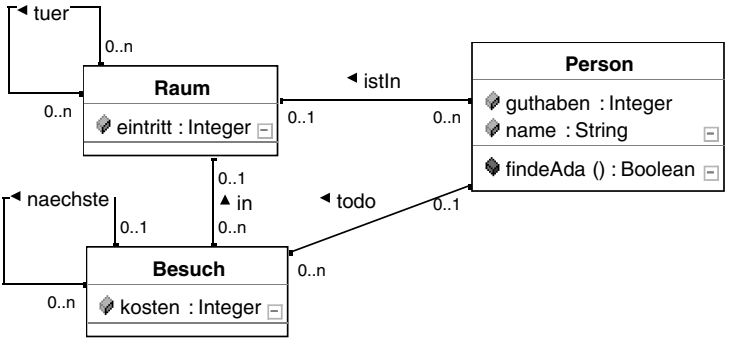

Abbildung 4: resultierendes Klassendiagramm

Man beachte, dass hier auf die Verwendung der Vererbung, der verschiedenen Beziehungs-

arten (Assoziation, Aggregation, Komposition), Diskussion von Sichtbarkeit und auf Design Pattern bewusst verzichtet wurde. Diese „Zusatz-Features“ des Klassendiagramms sind für die Modellierung im Anfangsunterricht nicht notwendig und können daher im Sinne didaktischer Reduktion zunächst weggelassen werden. Diese Konzepte würden am Anfang tatsächlich ein unnötig „aufwendiges Begriffssystem“ darstellen.

Der nächste Schritt ist nun, das Klassendiagramm in ein ausführbares Modell (beispielsweise Java Klassendateien) zu überführen. Hier können durch die Verwendung von CASE Tools erhebliche Erleichterungen des weiteren Unterrichts erreicht werden, da die meisten CASE Tools automatisch Quellcode aus Klassendiagrammen generieren können. Die Werkzeuge Rational Rose, Rhapsody und Fujaba können hierbei auch korrekt implementierte bidirektionale Assoziationen generieren, wie sie aus unserer Sicht sinnvoll sind.

Der nächste Schritt im Unterricht ist jetzt das Testen des generierten Modells. Hierzu bietet sich die Verwendung eines Objekt Browsers an. Ein Objekt Browser bietet die Möglichkeit, zur Laufzeit eines Programms die aktuellen Daten als Objektstruktur zu visualisieren. Dies geschieht idealerweise wieder als UML Objektdiagramm. Mögliche Objekt Browser bietet z. B. BlueJ, DOBS (Teil der Fujaba Tool Suite) und die Eclipse Plugins JavaSpider und eDOBS. Bislang unterstützen nur DOBS und eDOBS eine korrekte Darstellung von Links bei zu-n-Assoziationen. Des Weiteren kann man mit diesen Werkzeugen neue Objektstrukturen anlegen und bestehende verändern. Abbildung 7 zeigt einen Screenshot des DOBS Tools.

4 Verhaltensmodellierung

Um die Prinzessin Ada Aufgabe zu lösen, kann man zwei verschiedene Lösungsansätze wählen: den rekursiven oder den iterativen Ansatz. Bei rekursiven Ansätzen werden wichtige Teile des Verhaltens durch die impliziten Datenstrukturen auf dem Prozedurkeller gesteuert. Bei unserem Ansatz bietet es sich an, diese für Anfänger meist schwer nachvollziehbaren impliziten Informationen durch eine geeignete Objektstruktur explizit zu modellieren. Wir haben uns daher in diesem Beitrag für einen iterativen Ansatz entschieden, bei dem wir den Suchvorgang explizit modellieren.

Um unseren Lösungsansatz zu erarbeiten, spielen wir mit den Schülern ein Beispielszenario durch. Als Ausgangssituation verwenden wir das Objektdiagramm aus Abbildung 3. Wir erweitern dieses Objektdiagramm um den Prinzen, der auf der Anfangsposition steht. Wir führen dann Besuchsobjekte ein, die das Betreten und Verlassen eines Raums beschreiben. Dabei wird in diesen Besuchsobjekten das jeweilige Restguthaben gespeichert. Haben wir einen Raum besucht, dann erzeugen wir Folgebesuche für alle Nachbarräume. Diese Folgebesuche organisieren wir in einer Todo-Liste. Diese Todo-Liste arbeiten wir dann iterativ ab, bis ein vollständiger Baum von möglichen Besuchs-Folgen entsteht oder bis wir eine Lösung des Problems gefunden haben.

Als mögliche Änderungsoperationen werden unseren Schülern dabei das Erzeugen und Löschen von Objekten, das Erzeugen und Löschen von Links, das Ändern von Attributwerten und das Senden von Nachrichten vermittelt. Diese Änderungsoperationen lassen

sich leicht durch Wegwischen oder Neuzeichnen von Elementen an der Tafel verdeutlichen. Die einzelnen Szenario-Schritte können dabei entweder einfach textuell oder durch ein sogenanntes Storyboard (vgl. Abbildung 5) dokumentiert werden.

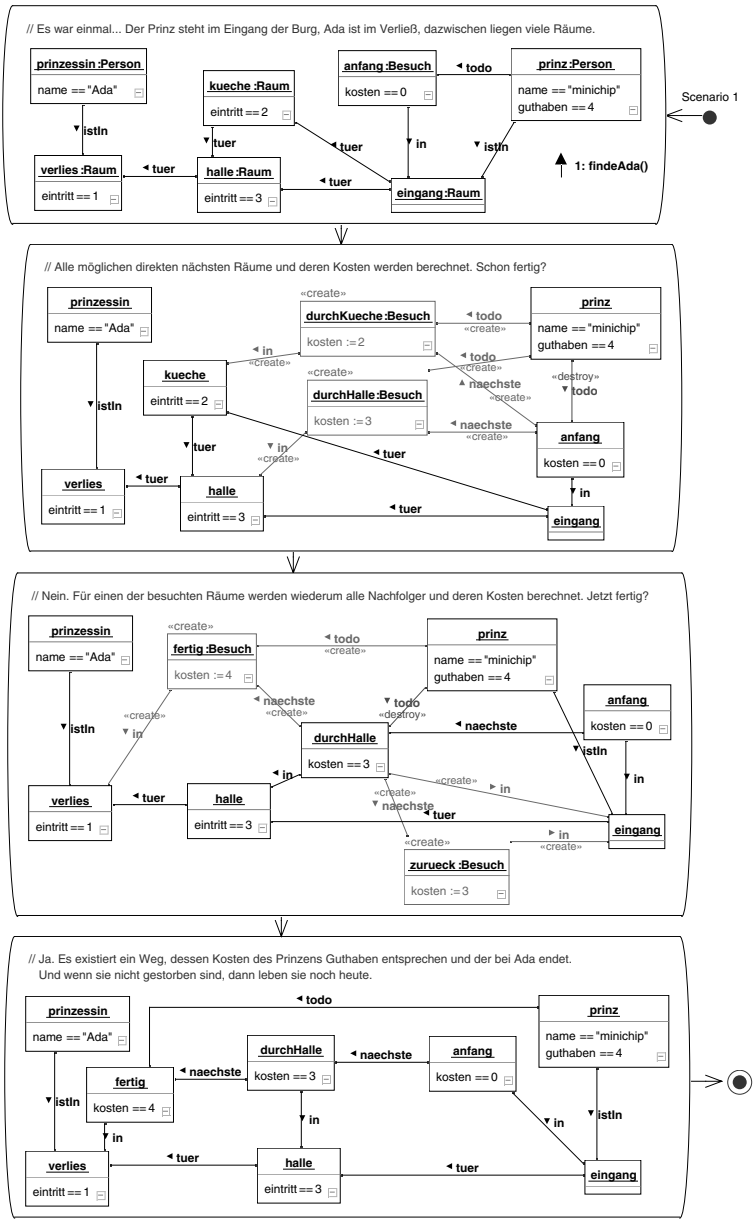

Abbildung 5: Storyboard

Abbildung 5 zeigt ein Szenario für die skizzierte Herangehensweise. Die erste Aktivität modelliert die Startsituation. Diese ist ähnlich zum Objektdiagramm in Abbildung 3, nur dass hier zur Vereinfachung die Anzahl der Räume auf vier (Verlies, Küche, Halle und Eingang) reduziert wurde. Zusätzlich steht auch noch Prinz Minichip im Eingang, was wiederum durch ein Objekt der Klasse `Person` realisiert wurde. Das Objekt `anfang` der Klasse `Besuch` stellt dar, dass der Prinz bis jetzt nur die Eingangshalle besucht und dafür noch kein Geld ausgegeben hat.

Die erste Aktivität beschreibt zusätzlich die Aktion, die die Ausführung des Szenarios anstößt. In diesem Fall ist dies der Aufruf der Methode `findeAda()` auf dem Objekt `prinz`. Die nachfolgenden Aktivitäten beschreiben nun die Arbeitsweise dieser Methode in der gegebenen Beispielsituation.

In der zweiten Aktivität wird ein Besuch aus der Liste der `todo` Liste des Prinzen ausgewählt (hier das Objekt `anfang`). Für alle möglichen nachfolgenden Räume (`halle` und `kueche`) wird ein neuer Besuch angelegt und mittels eines `naechste` Links als nachfolgender Besuch markiert. Für die nachfolgenden Besuche werden die entstehenden Kosten ausgerechnet und im `kosten` Attribut abgelegt. Am Schluß wird der `todo` Link zum `anfang` Objekt gelöscht, da dieses nun abgehandelt ist.

In Aktivität drei wird nun das Objekt `durchHalle` als zu bearbeitender Besuch ausgewählt. Mit diesem Besuch wird dann wie oben verfahren. Man beachte, dass es hier nicht mehr möglich ist, von der Halle in die Küche zu wechseln, da dies das Guthaben des Prinzen übersteigen würde.

Da mit dem Objekt `fertig` ein Besuch existiert, der bei der Prinzessin ankommt und genau das Guthaben des Prinzens verbraucht, ist in Aktivität 4 ein Weg gefunden. Das Szenario terminiert hier. Diese Aktivität stellt also die Endsituation dar.

Nun soll eine allgemeine Lösung des Problems programmiert werden. Die Implementierung der verwendeten Klassen mit ihren Attributen und Beziehungen überlässt man wie diskutiert am einfachsten einem CASE Tool. Bleibt die Implementierung der Methode `findeAda()`. Will man diese direkt in einer objektorientierten Sprache wie z. B. Java programmieren, dann sind textuelle Protokolle der elementaren Arbeitsschritte des Beispielszenarios eine guter Ausgangspunkt. Die Implementierung der Klassendiagramme stellt für unsere elementaren Änderungsoperationen wie Erzeugen eines Objekts oder ändern eines Attributs entsprechende Grundoperationen zur Verfügung. Damit können Schritte wie „bis die Todo-Liste leer ist, nimm das nächste Element aus der Todo-Liste“, „für jede Tür lege einen Nachfolgebesuch an“ usw. relativ leicht abgeleitet und umgesetzt werden.

Um die Ableitung der Methodenrümpfe zu vereinfachen, verwenden wir im Fujaba Ansatz anstelle einer direkten Programmierung sogenannte Regeldiagramme. Fujabas Regeldiagramme haben eine ähnliche Syntax wie die oben vorgestellten Storyboards. Es existieren dieselben Operationen zur Änderung der Objektstruktur. Allerdings wird für Methodenimplementierungen auch noch Kontrollfluss benötigt. Diesen modellieren wir durch ein UML Aktivitätsdiagramm. Abbildung 6 zeigt ein solches Regeldiagramm, das den skizzierten Algorithmus zur Objektsuche implementiert.

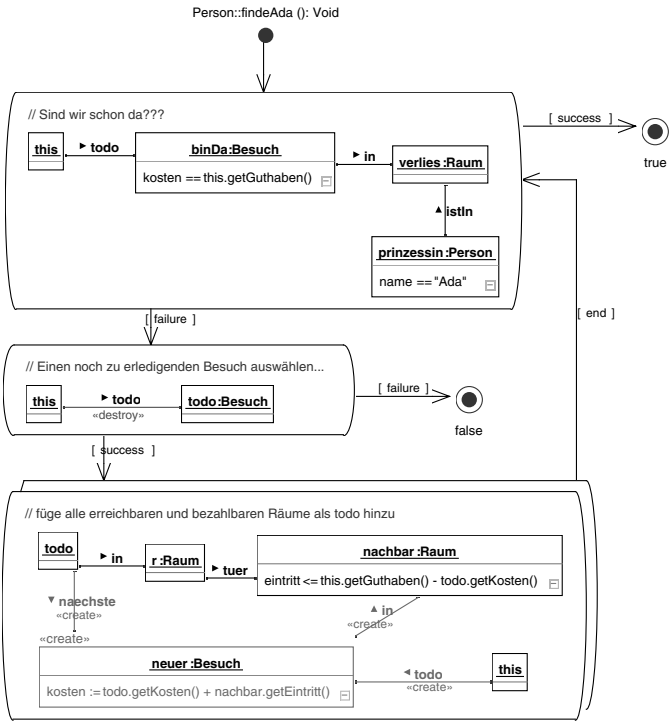

Abbildung 6: Modell der Methode `findeAda()`

Die erste Aktivität in Abbildung 6 beschreibt die Abbruchbedingung des Algorithmus. Bei der Ausführung einer solchen Aktivität wird versucht, die durch das enthaltene Objektdiagramm beschriebene Objektstruktur zu finden. Die Suche startet beim `this` Objekt, dem Objekt, auf dem die Methode aufgerufen wurde. Von hier aus wird dann versucht, über den `todo` Link ein Objekt vom Typ `Besuch` zu finden, dessen `kosten` Attribut gleich dem `guthaben` Attribut des `this` Objekts ist. Wird ein solches Objekt gefunden, wird es mit `binDa` bezeichnet. Von hier wird über den `in` Link der Raum gesucht, in dem dieser Besuch ankommt. Kann in diesem Raum nun über den `istIn` Link eine Person gefunden werden, die Ada heißt, ist die Suche erfolgreich gewesen. Die Aktivität wird in diesem Fall über die `[success]` Kante verlassen. Der Algorithmus terminiert.

Kann kein solcher Endbesuch gefunden werden, wird die Aktivität über die `[failure]` Kante verlassen. In der nächsten Aktivität wird jetzt ein beliebiger Besuch über den `todo` Link ausgewählt. Wenn kein Besuch gefunden wird, dass heißt es ist kein Besuch mehr in der `todo` Liste, existiert keine Lösung. Die Aktivität wird über die `failure` Kante verlassen und die Methode terminiert. Falls aber ein Besuch gefunden wurde, wird dieser mit `todo` benannt und aus der `todo` Liste gelöscht. Man beachte, dass man an dieser Stelle durch ein entsprechendes Auswahlkriterium für den Besuch verschiedene Suchstrategien (Depth-first, Breadth-first, greedy) im Unterricht thematisieren kann.

In der untersten Aktivität taucht das `todo` Objekt ohne nachfolgenden Klassennamen auf.

Das heißt, dass das `todo` Objekt von der vorangehenden Aktivität bekannt (bound) ist und weiter verwendet wird. Von hier aus wird der besuchte Raum und von dort ein Nachbarraum (über den `tuer` Link) gesucht. Der Nachbarraum muss aber mit dem verbleibenden Guthaben noch bezahlbar sein. Wird ein solcher Nachbarraum gefunden, wird dafür ein `neuer` Besuch erzeugt, der in die `todo` Kante eingehängt wird und als Nachfolger des `todo` Besuchs markiert wird. Die Kosten dieses Besuchs werden ausgerechnet und im `kosten` Attribut abgelegt. Da diese Aktivität einen doppelten Rand besitzt (sog. for-each activity), wird dies nicht nur für einen Nachfolgeraum, sondern für alle Vorkommen dieser Objektstruktur ausgeführt. Es werden also alle bezahlbaren Nachfolgeräume markiert. Ist dies geschehen, wird wieder zur ersten Aktivität zurückgesprungen, das heißt die Abbruchbedingung wird erneut überprüft, die Schleife beginnt von vorn.

Im nächsten Schritt muss nun das Regeldiagramm in ein ausführbares Programm übersetzt werden. Regeldiagramme können zur Konzeptfindung mit Papier und Bleistift eingesetzt und dann von Hand z. B. in Java implementiert werden. Alternativ kann das Fujaba CASE Tool aus Regeldiagrammen ausführbaren Java Code generieren.

Dieser Code kann dann getestet werden. Fujaba bietet hierfür das bereits erwähnte Werkzeug Dobs an. Abbildung 7 zeigt den Dobs nach Aufruf der `findeAda()` Methode auf dem Objekt `p6`. Die Besuche, die den Prinzen zur Prinzessin führen, sind die Objekte `b1`, `b10` und `b13`.

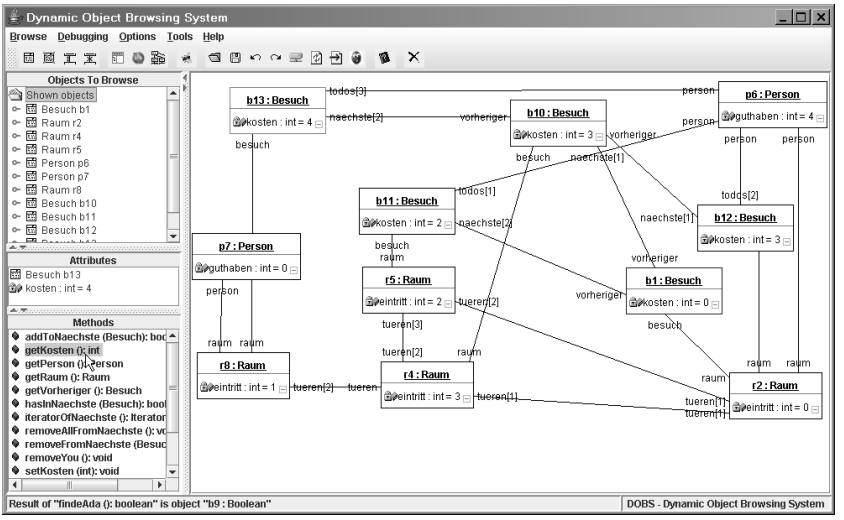

Abbildung 7: Dobs

5 Zusammenfassung und Ausblick

Um eine breitere Übersicht über die Vor- und Nachteile verschiedener Modellierungs- und Implementierungsweisen für Aufgaben wie die „findeAda“-Aufgabe zu bekommen, haben

wir eine Interviewstudie begonnen, in der wir Probanden mit unterschiedlichen Vorkenntnissen befragen. Ziel ist es, die (für Anfänger) geeignetste Herangehensweise zu finden. Bisher haben wir circa zwanzig Leitfaden gestützte Interviews durchgeführt. Dabei haben Probanden mit Informatik-Vorkenntnissen aus dem Bereich Algorithmen und Datenstrukturen sehr häufig die Analogien zur kürzesten Wegesuche erkannt und entsprechend den Schlossgrundriss als Graph modelliert und diesen dann entweder als Adjazenzmatrix oder funktional mit Hilfe von Listen gespeichert. Eine kaufmännische Auszubildende wählte eine Karteikarten-Methapher, die unserer objektorientierten Modellierung nahe kommt. Ein Proband, der schon im Fujaba Ansatz ausgebildet war, wählte exakt unsere Modellierung.

In einem zweiten Teil der Interviews haben wir die Aufgabe in Richtung Datenmodellierung anspruchsvoller gestaltet. Wir haben Türen eingeführt, die abgeschlossen sein können und für die Schlüssel in anderen Räumen versteckt sind. Zusätzlich sind in den Räumen drei Geschenke versteckt, die der Prinz finden und mitnehmen muss. Diese Erweiterungen der Aufgabenstellung wurden von den imperativen und funktionalen Probanden meist durch zusätzliche Arrays oder Listen zur Speicherung der Türzustände, der Schlüsselpositionen, der Geschenkpositionen und der Besitztümer des Prinzen beantwortet. Diese Vielzahl von zusätzlichen Arrays oder Listen wurde schon bei diesem kleinen Beispiel sehr schnell sehr unübersichtlich. Bei der Bitte um eine Zusammenfassung ihrer Lösung am Ende des Interviews hatten diese Kandidaten meist selbst schon große Schwierigkeiten, sich daran zu erinnern, welches Array oder welche Liste sie wozu verwenden wollten und wie ein Suchschritt in diesen Datenstrukturen abläuft.

Die beiden Kandidaten mit der objektorientierten Modellierung hatten mit der Erweiterung der Aufgabenstellung praktisch keine Probleme. Sie führten Objekte für Türen, Schlüssel und Geschenke ein und ein paar zusätzliche Beziehungen.

Im typischen Programmiersprachenunterricht wird zuerst die Syntax von Kontrollstrukturen behandelt. Datenmodellierung beschränkt sich meist auf Basistypen und Arrays. Wenn nur so wenige Sprachkonstrukte zur Datenmodellierung zur Verfügung stehen, dann müssen die Aufgabentypen zwangsläufig entsprechend reduziert werden. Typisch sind mathematische Aufgaben wie Sieb des Eratosthenes oder Sortierprobleme. Graphalgorithmen gelten schon als anspruchsvoll. Lebensraumnahe Aufgabenstellungen benötigen aber häufig eine explizite Datenmodellierung. Dies kann nach unserer Erfahrung mit Hilfe von einfachen objektorientierten Basiskonzepten auch im Anfangsunterricht schon sehr erfolgreich eingeführt werden. Mit Hilfe solcher Datenmodellierungskompetenzen sind die Schüler nach unserer Meinung viel eher in der Lage, Informatikwissen zur Problemlösung in lebensraumnahen Problemstellungen anzuwenden. Der oft gefürchtete Overhead an objektorientierten Konstrukten kann durch eine Konzentration auf die Datenmodellierung auch für den Anfangsunterricht überschaubar gehalten werden. Bei der Verwendung von CASE Tools zur Codegenerierung aus Klassendiagrammen wird die Programmierung auf die problemrelevanten Arbeitsschritte beschränkt. Explizite grafische Modellierung mit Hilfe von Objektdiagrammen ermöglicht darüber hinaus sehr gut Gruppendiskussionen über Designentscheidungen. Wir haben mit expliziter objektorientierter Datenmodellierung im Anfängerunterricht seit Jahren sehr gute Unterrichtserfahrungen gemacht.

Literaturverzeichnis

[Ba98] Helmut Balzert: Lehrbuch der Software-Technik 1, 2. Aufl.,Spektrum, 1998.

[DGZ02] I. Diethelm, L. Geiger, A. Zündorf: UML im Unterricht: Systematische objektorientierte Problemlösung mit Hilfe von Szenarien am Beispiel der Türme von Hanoi. in Forschungsbeiträge zur "Didaktik der Informatik" - Theorie, Praxis und Evaluation; GI-Lecture Notes, pp. 33-42 (2002)

[DGZ05] I. Diethelm, L. Geiger, A. Zündorf: Teaching Modeling with Objects First; submitted to WCCE05 (2005)

[Di02] 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.

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

[He04] A. Heubaum: Möglichkeiten und Grenzen maschineller Intelligenz; in LOG IN Heft Nr. 128/129, pp. 62-79, 2004.

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

[KöZ00] 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)

[li02] life^3-Homepage, Universität Paderborn, http://life.uni-paderborn.de/.

[Rö03] G. Röhner: Suchbaum-Modellierung; in Informatische Fachkonzepte im Unterricht, INFOS 2003, pp. 177-178, 2003.

[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.