Erfahrungen mit dem Einsatz von Objectory für vorlesungsbegleitende Übungen in der Softwaretechnik-Ausbildung

Wilhelm Hasselbring*

Zusammenfassung

Objectory ist ein CASE-Werkzeug für die objektorientierte Entwicklungsmethodologie OOSE. In diesem Beitrag soll über den Einsatz von Objectory innerhalb der universitären Ausbildung in Softwaretechnik berichtet werden. Objectory wurde u.a. für die vorlesungsbegleitenden Übungen zu einer Softwaretechnologie-Vorlesung eingesetzt. Die Übungsaufgaben werden kurz vorgestellt und die Erfahrungen bei der Bearbeitung mittels Objectory diskutiert.

Schlüsselwörter: Vorlesungsbegleitende Übungen, Softwaretechnik, Objektorientierung, CASE, Objectory.

1 Einleitung

In den Softwaretechnik-Trends wurde vor kurzer Zeit über den Einsatz von selbst entwickelten CASE-Werkzeugen für die Lehre berichtet [Do95]. Im vorliegenden Beitrag soll über die Erfahrungen mit dem Einsatz des kommerziellen CASE-Werkzeugs Objectory [Obj95] für die vorlesungsbegleitenden Übungen einer Softwaretechnologie-Vorlesung berichtet werden. Objectory ist das CASE-Werkzeug für die objektorientierte Entwicklungsmethodologie OOSE [JCJÖ92].

2 Die Vorlesung Softwaretechnologie

Ein Überblick über die Inhalte der Vorlesung Softwaretechnologie, die im WS 95/96 an der Universität Dortmund abgehalten wurde, ist in Abbildung 1 dargestellt. Diese Vorlesung ist eine sogenannte *Stammvorlesung* (4+2 Semesterwochenstunden), die die Grundlage für Spezialvorlesungen und spezielle Kurse im Bereich der Softwaretechnik bildet. Aufgrund des Stoffumfangs können die einzelnen Themen nur ansatzweise behandelt werden. Das primäre Ziel ist, einen breiten Überblick über die Prinzipien und Techniken für die systematische Entwicklung von Software zu bieten.

[GJM91] bildete die Grundlage für diese Veranstaltung. Als zusätzlicher Schwerpunkt wurde die objektorientierte Entwicklung gewählt. Für die Teile der Vorlesung, die nicht auf [GJM91] basieren, sind in Abbildung 1 Literaturverweise angegeben. Gegenüber den früheren Jahren wurde im WS 95/96 der Schwerpunkt auf die objektorientierten Methoden gelegt (bisher wurde hier nur der Entwurf mit Eiffel behandelt). Besonders ausführlich wurde OOSE behandelt. Bei der Vorstellung von OMT, der Booch-Methodologie[1] und der neuen 'Unified Method' wurde im wesentlichen auf die Unterschiede und Gemeinsamkeiten zu OOSE eingegangen. Dabei stand im Mittelpunkt, den Studierenden die Gemeinsamkeiten der verschiedenen objektorientierten Methodologien zu verdeutlichen, obwohl teilweise sehr unterschiedliche konkrete Notationen eingesetzt werden. Wichtig ist, die jeweils gewählte Methodologie konsequent anzuwenden. Den einsetzbaren Werkzeugen kommt dabei eine zentrale Rolle zu: Software sollte mit Software entwickelt werden. Für eine derartige Lehrveranstaltung ist es wichtig, ein ausgewogenes Verhältnis zwischen den zu vermittelnden Konzepten und den eingesetzten Werkzeugen zu finden [Gli96]. Mir persönlich erscheint es sinnvoller, für die Softwareentwicklung vorzugsweise eine weniger mächtige Methodologie mit ausgereifter Werkzeugunterstützung einzusetzen, als eine besonders mächtige Methodologie, für die keine Werkzeugunterstützung existiert. Das Faktenwissen über ein konkretes Werkzeug veraltet

---

0. Einleitung und Motivation
1. Der Entwicklungsprozeß
2. Spezifikationssprachen und -methoden
   - Datenflußdiagramme
   - Endliche Automaten
   - Petri-Netze
   - E/R-Modellierung
   - Modulspezifikationssprachen
   - Algebraische Spezifikationen
   - Strukturierte Methodologien (SA/SD, JSD)
   - Formale Spezifikation mit Z [Dil94] und Object-Z [SBC92]
3. Objektorientierte Entwicklung
   - Grundlagen
   - Entwurf mit Eiffel [Mey88]
   - OOSE [JCJÖ92]
   - OMT [RMW+91]
   - Die Booch-Methodologie [Boo94]
   - Die 'Unified Method' [BR95]
4. Validation und Verifikation von Software
   - Testen, Analyse, symbolische Ausführung
   - Objektorientiertes Testen [Sne95]
5. Werkzeuge und Softwareentwicklungsumgebungen
6. Management von Softwareprojekten

Abbildung 1: Inhalte der Vorlesung Softwaretechnologie.

*Universität Dortmund, Informatik 10 (Software-Technologie), 44221 Dortmund, willi@ls10.informatik.uni-dortmund.de

[1] Unter dem Begriff *Methodologie* wird hier ein System von einzelnen Methoden verstanden [GJM91]. Methoden sollten zumindest eine Spezifikationssprache einsetzen und Methodologien sollten den gesamten Entwicklungsprozeß unterstützen.

allerdings sehr schnell, so daß bei der Ausbildung der Schwerpunkt auf die Vermittlung grundlegender Konzepte gelegt werden muß. Die praktischen Erfahrungen mit dem Einsatz eines konkreten Werkzeugs können dann das Verständnis der eher abstrakten Konzepte vertiefen. Im vorliegenden Beitrag wird nun über die Erfahrungen mit dem Einsatz von Objectory für vorlesungsbegleitende Übungen zu OOSE berichtet. Die 'Unified Method' stellt für diesen Bereich im Prinzip einen Ausblick dar, weil sie von den Entwicklern von OOSE (Jacobson), OMT (Rumbaugh) und Booch gemeinsam entwickelt wird. Objectory wurde rein pragmatisch ausgewählt, weil dafür Ausbildungslizenzen zur Verfügung standen.

Wie in Abbildung 1 zu sehen ist, wird die objektorientierte Entwicklung innerhalb eines eigenen Abschnitts dieser Stammvorlesung behandelt. In [Dis95] wird vorgeschlagen, bei einer Veranstaltung, die den Zugang zum Fachgebiet Softwaretechnik bieten soll, auf die explizite Vermittlung *herkömmlicher* Vorgehensweisen zu verzichten und statt dessen für solch eine Veranstaltung das Paradigma der Objektorientierung als integrativen Gesamtrahmen zu nehmen. Dieser Weg wurde hier nicht verfolgt, weil die objektorientierten Methodologien ja nicht von Grund auf neu entwickelt wurden, sondern sich aus den "strukturierten" Methodologien entwickelt haben. Klassendiagramme sind z.B. erweiterte E/R-Diagramme und zur Dynamikmodellierung werden i.a. Erweiterungen von endlichen Automaten eingesetzt. Indem zunächst die zugrundeliegenden Spezifikationsmethoden eingeführt werden, kann (meiner Ansicht nach) die Entwicklung hin zu objektorientierten Methodologien durch "Kombination" von Spezifikationsmethoden motiviert werden. Dadurch wird automatisch auch eine Wiederholung erreicht.

3 Die objektorientierte Entwicklungsmethodologie OOSE

OOSE ist eine in der Telekommunikationsindustrie entwickelte und eingesetzte objektorientierte Methodologie, die die iterative, inkrementelle Entwicklung betont, wobei vermieden wird, den Entwicklungsprozeß in ein vorgegebenes Schema zu fassen [JCJÖ92]. OOSE ist einer der ältesten Ansätze zur objektorientierten Entwicklung, der durch das Werkzeug Objectory unterstützt wird. In diesem Beitrag kann nur ein sehr grober Überblick gegeben werden. Für eine ausführliche Einführung in OOSE sei auf [JCJÖ92] verwiesen.

Bei der Entwicklung komplexer Softwaresysteme stellt insbesondere die Erfüllung der *tatsächlichen* Anforderungen der Anwender ein großes Problem dar. Mit der objektorientierten Entwicklungsmethodologie OOSE und dem dazugehörigen Werkzeug Objectory sollen die resultierenden Probleme stufenweise in aufeinanderfolgenden Modellen bewältigt werden. Objectory arbeitet mit fünf Modellen, wobei der Schwerpunkt in der Anforderungsanalyse liegt:

- Im **Analyseprozeß** werden die folgenden Modelle konstruiert:
  - Das Anforderungsmodell deckt die funktionellen Anforderungen ab.
  - Das Analysemodell gibt dem System eine robuste und änderbare Objektstruktur.
- Im **Konstruktionsprozeß** werden die folgenden Modelle konstruiert:
  - Das Entwurfsmodell paßt die Objektstruktur an die Implementierungsumgebung an.
  - Das Implementierungsmodell dient zur Implementierung des Systems.
  - Das Testmodell dient zum Testen des Systems.

Bestandteile des **Anforderungsmodells** sind Akteure und Anwendungsfallbeschreibungen (use cases). Die **Akteure** interagieren mit dem zu entwickelnden System. Sie sind nicht Teil des zu entwickelnden Systems. Die Akteure können Menschen in bestimmten Rollen oder technische Gerätschaften sein, die nicht genau beschrieben werden müssen. Die **Anwendungsfallbeschreibungen** beschreiben, was das zu entwickelnde System können soll. Anwendungsfälle sind sequentielle Folgen von Transaktionen, die zunächst informell beschrieben werden. Zusätzlich gehören zum Anforderungsmodell **Schnittstellenbeschreibungen** (z.B. Prototypen für graphische Benutzungsschnittstellen) sowie das **Problembereichsmodell** (domain object model), das den Ist-Zustand beschreiben soll. Ein Problembereichsmodell enthält Objekte mit Gegenstück im realen Problembereich bzw. Konzepte, mit denen das System umgehen soll. Das Problembereichsmodell kann nach gängigen anderen Verfahren (z.B. Booch oder OMT) konstruiert werden und soll nicht direkt im Entwurf erhalten bleiben, sondern nur das Problemverständnis fördern. Das Problembereichsmodell dient auch als Grundlage für das Analysemodell.

Hier sei angemerkt, daß Kommunikation, Nebenläufigkeit und Konkurrenz zwischen Anwendungsfällen nicht modelliert werden können. Anwendungsfälle kommunizieren mit Akteuren, nicht mit anderen Anwendungsfällen.

Das **Analysemodell** dient zur Strukturierung des Anforderungsmodells und zur initialen Identifikation von Objekten, die das System als änderbare Objektarchitektur strukturieren sollen. Bestandteile des Analysemodells sind **Schnittstellenobjekte** für die Außenwelt, **Entitätsobjekte** für die Repräsentation von Information mit zughörigem Verhalten und **Kontrollobjekte** für die Beschreibung von Verhalten, das mehrere Objekte betreffen kann. Implementierungsaspekte sollen im Analysemodell noch vernachlässigt werden; das Analysemodell ist aber als ein erster, grober Entwurf anzusehen. Im Problembereichsmodell werden nur Entitätsobjekte modelliert, wie sie auch in Objekt- oder Klassendiagrammen der meisten anderen objektorientierten Methodologien üblich sind. In OOSE existiert jedoch keine explizite Unterscheidung zwischen Klassen und Objekten.

Das **Entwurfsmodell** dient zur Verfeinerung und Anpassung der Anforderungs- und Analysemodelle an die Sy-

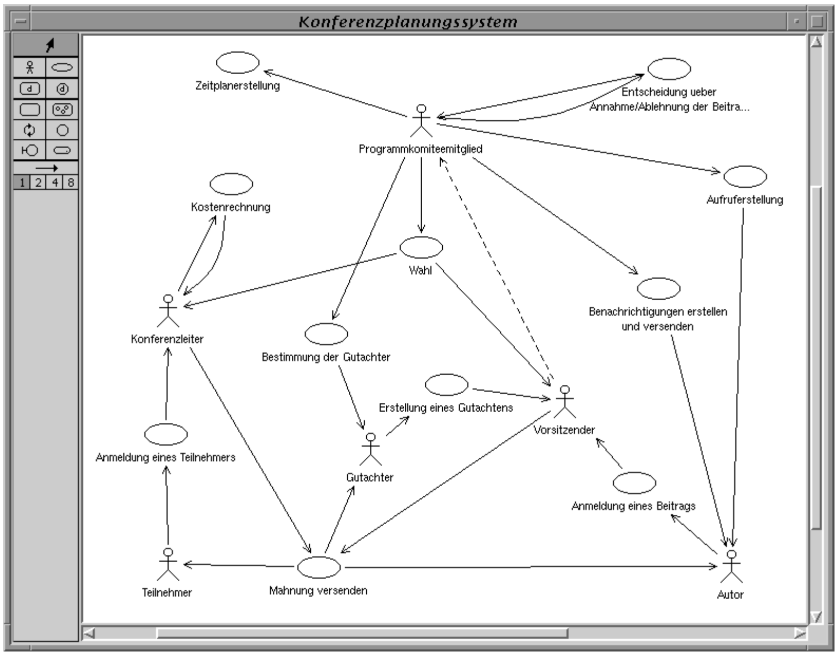

Abbildung 2: Ein abgegebenes Diagramm für Akteure und Anwendungsfälle. Ellipsen stellen Anwendungsfälle dar und Strichmännchen stellen Akteure dar. Die Pfeile sind Kommunikationsverbindungen. Vererbung wird durch gestrichelte Pfeile dargestellt.

stemumgebung (Programmiersprache, Datenbank, Netzwerk etc). Die Spezifikation der Dynamik erfolgt durch Interaktionsdiagramme und Zustandsübergangsdiagramme. Die Nachvollziehbarkeit der Transformation aus dem Analysemodell ist wichtig und wird durch Objectory unterstützt. Das Entwurfsmodell enthält nur noch Entitätsobjekte. Das **Implementierungsmodell** ist ein ausführbares Programm und das **Testmodell** besteht aus Testplänen und -ergebnissen.

Die Verwendung unterschiedlicher Notationen für Objekte in Analyse- und Entwurfsmodellen macht die Methodologie unnötig komplex. Andererseits wird durch die unterschiedlichen Notationen der Übergang von der Analyse zum Entwurf explizit gemacht. OOSE verwendet relativ einfache Notationen, die auch problemlos handschriftlich verwendet werden können. Die Modelle können allerdings leicht unübersichtlich und schlecht lesbar werden. Das Werkzeug Objectory selbst wird in [JCJÖ92] mit Hilfe von OOSE beschrieben.

4 Die Übungsaufgaben

Zur Einarbeitung in OOSE bzw. Objectory wurden zuerst die folgenden Aufgaben gestellt:

1. Zunächst sollten grundsätzliche Fragen zum Verständnis von OOSE beantwortet werden.

2. Zur Einarbeitung in das Werkzeug Objectory sollte ein vorgegebenes Beispiel untersucht werden (der Recycling-Automat aus [JCJÖ92]). Die ersten Schritte in der Handhabung von Objectory wurden detailliert beschrieben. Bestimmte Teilspezifikationen (z.B. das Analysemodell) sollten gefunden und für die Abgabe ausgedruckt werden. Eine kurze Beschreibung des Beispiels wurde gefordert.

   Das Ziel dieser Aufgabe war, die Orientierung innerhalb der verschiedenen Spezifikationen eines Anwendungsbeispiels mit den dazugehörigen Editoren zu erlernen. Das "Document Archive" von Objectory selbst bietet hierzu nur eine rudimentäre Unterstützung. Man muß schon selbst wissen, was man will bzw. macht.

3. Spezifikation eines einfachen Analysemodells für eine Universitätsverwaltung wahlweise per Hand oder mit Objectory. Die OOSE-Notation ist so einfach gehalten, daß sie problemlos mit der Hand gezeichnet werden kann.

Als Hauptaufgabe sollten die Anforderungen an ein Konferenzplanungssystem in mehreren Schritten analysiert werden:

1. In der ersten Phase sollte ein Anforderungsmodell erstellt werden. Für das Anforderungsmodell mußten Items (Begriffe im Datenlexikon), Requirements (informelle Anforderungen), Akteure und Anwendungsfälle (use cases) ermittelt werden. Ein abgegebenes Diagramm für Akteure und Anwendungsfälle ist in Abbildung 2 dargestellt. Für eine Beschreibung der Notation sei auf [JCJÖ92] verwiesen. Dieses Diagramm dient in OOSE im wesentlichen der Strukturierung der informellen Beschreibungen der einzelnen Akteure und Anwendungsfälle, die wiederum mit spezifischen Editoren erstellt werden. Die inhaltliche

Angemessenheit dieses Beispielmodells soll hier nicht diskutiert werden; das geschah mit der Vergabe von Punkten für diese Lösung. Hier soll nur ein grober Eindruck von den studentischen Lösungen vermittelt werden.

2. Darauf basierend sollte ein Problembereichsmodell und ein Analysemodell konstruiert werden. Ein abgegebenes Problembereichsmodell ist in Abbildung 3 dargestellt. Zwei sehr unterschiedliche Analysemodelle für die gleiche Aufgabe sind in den Abbildungen 4 und 5 dargestellt. Das erste Analysemodell ist im Prinzip wie ein Datenflußdiagramm aufgebaut. Im zweiten Analysemodell werden dagegen auch statische Aspekte, wie in E/R-Diagrammen, modelliert. Ob es sinnvoll ist, sowohl die Datenflußsicht als auch die Sicht auf die statischen Beziehungen zwischen Entitäten in einem Diagramm darzustellen, soll hier nicht diskutiert werden. Hier sei nur angemerkt, daß die OOSE-Analysemodellsprache sowohl statische Assoziations- und Aggregationsverbindungen als auch dynamische Kommunikationsverbindungen zwischen Objekten mit einer Spezifikationssprache zur Verfügung stellt.

   Grundsätzlich sind mit Objectory auch andere Vorgehensweisen möglich. Z.B. könnte zuerst das Problembereichsmodell modelliert werden, oder verschiedene Modelle könnten iterativ und inkrementell modelliert werden.

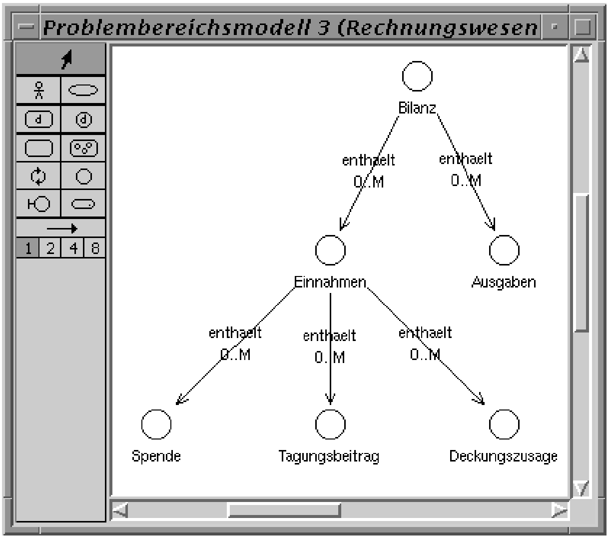

Abbildung 3: Ein Teil eines abgegebenen Problembereichsmodells zur Modellierung des Ist-Zustands. Hier wurden nur Aggregationsbeziehungen zwischen Objekten bzw. Klassen modelliert.

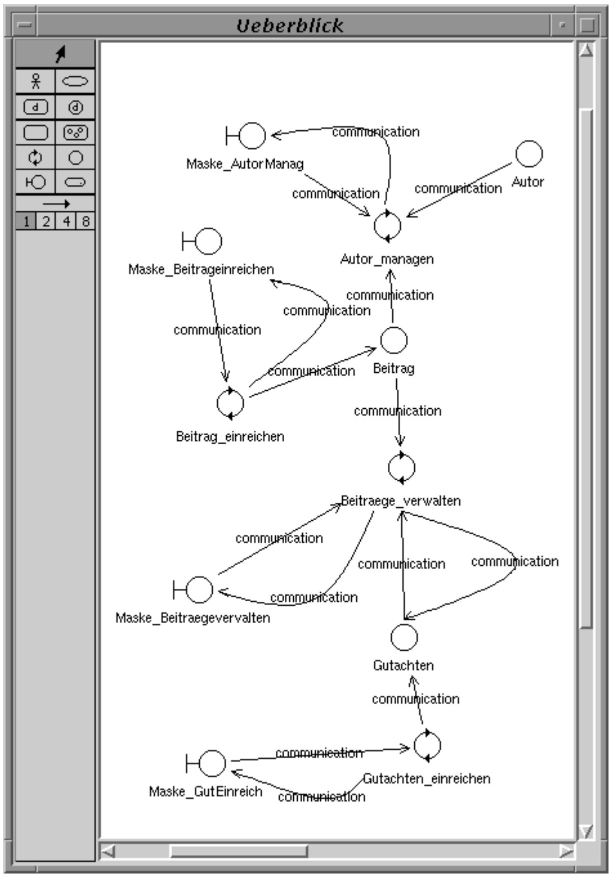

Abbildung 4: Ein Teil eines abgegebenen Analysemodells, das sehr an ein Datenflußdiagramm erinnert. Entitätsobjekte sind einfache Kreise. Schnittstellenobjekte und Kontrollobjekte sind mit ⊢ bzw. mit jeweils zwei Pfeilen dekoriert.

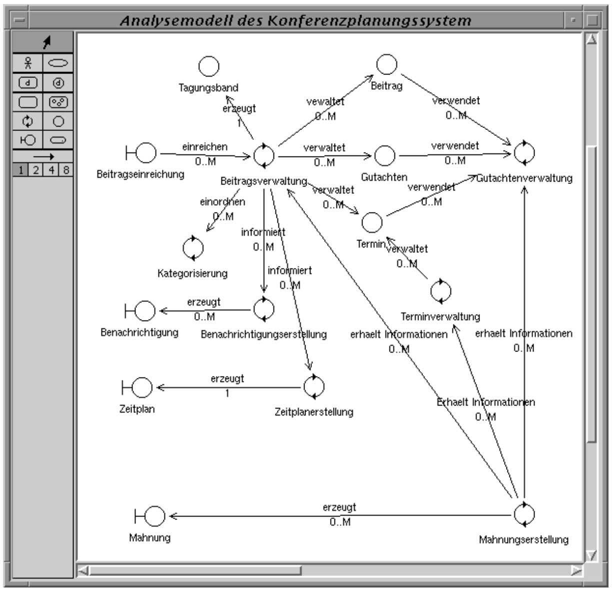

Abbildung 5: Ein Teil eines abgegebenen Analysemodells, in dem auch statische Aspekte modelliert werden.

3. Für den ersten Entwurf des Konferenzplanungssystems wurde nicht Objectory, sondern das public-domain Werkzeug xomt [SUvG95] verwendet, um auch eine andere Methodologie praktisch einzusetzen. Ein Teil eines abgegebenen OMT-Klassendiagramms ist in Abbildung 6 dargestellt. Hier können nur statische Verbindungen zwischen Klassen modelliert werden.

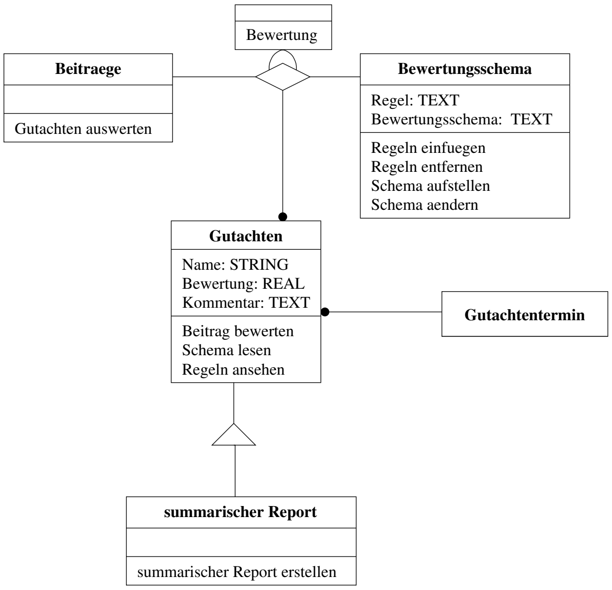

Abbildung 6: Ein Teil eines abgegebenen OMT-Klassendiagramms, das mit xomt konstruiert wurde.

5 Anmerkungen zur Bearbeitung der Übungsaufgaben

Die Lösungen der Studierenden legen die Vermutung nahe, daß sich Datenflußdiagramme besser zur Anforderungsanalyse eignen als E/R-Diagramme, die eher dem Entwurf zugeordnet werden sollten. Datenflußdiagramme haben sich ja auch in der strukturierten Analyse durchgesetzt [GJM91]. Insofern erscheint es fragwürdig, ob die von vielen objektorientierten Methodologien propagierte Analyse mit Klassendiagrammen tragfähig ist. Gelegentlich wird vorgeschlagen, aus einem natürlichsprachigen Text zuerst die Substantive herauszusuchen, um daraus die Klassen zu bestimmen [RMW+91]. Das reicht aber bestenfalls zur Beschreibung des Ist-Zustands aus, wie dies auch in OOSE im Problembereichsmodell geschieht. Zur Anforderungsanalyse sind zusätzliche Techniken sinnvoll, z.B. Prototyping [HK96]. Nach Jacobson [JCJÖ92] umfaßt das Anforderungsmodell in OOSE zusätzlich zu den Anwendungsfallbeschreibungen und dem Problembereichsmodell auch Prototypen (in erster Linie für Benutzungsschnittstellen). Objectory selbst enthält jedoch keine Werkzeuge zur Konstruktion von Benutzungsschnittstellen.

Diese Eindrücke können natürlich auch durch die Vorkenntnisse der Studierenden beeinflußt sein: funktionale und prozedurale Programmierung sowie Erfahrungen mit Datenflußdiagrammen im Software-Praktikum während des Grundstudiums.

Ein grundsätzliches Problem ist auf jeden Fall die Identifikation geeigneter Objekte bzw. Klassen. Es reicht nicht aus, einfach die Objekte der Realität im Rechner abzubilden, wie das von vielen objektorientierten Methodologien propagiert wird. Eine gute Objektstruktur muß auch berücksichtigen, wie das zu entwickelnde System benutzt werden soll. In einer Anwendung kann es sinnvoll sein, ein bestimmtes Objekt der Realität im Rechner abzubilden, in einer anderen Anwendung kann die Modellierung dieses Objekts völlig überflüssig sein. "Gute" Objekte und "gute" Objektmodelle können nur kontextabhängig ermittelt und bewertet werden. Generelle Zielsetzungen in OOSE sind die Robustheit für leichte Änderbarkeit (OOSE geht von ständigen Systemanpassungen aus) und die Unterstützung des Systemverständnisses.

Da Objectory ein integriertes CASE-Werkzeug ist [Was89], das den gesamten Entwicklungsprozeß unterstützt, ist dieses Werkzeug auch recht komplex. Dadurch ist der Aufwand für die Einarbeitung bei vorlesungsbegleitenden Übungen nicht zu vernachlässigen. Eine Präsentations-Integration ist im Prinzip gegeben, weil alle Editoren ähnlich aussehen und bedient werden. Die Bedienung selbst ist allerdings recht eigenwillig, z.B. können einzelne Editoren nur über Funktionen des Window-Managers geschlossen werden. Eine einfache dateibasierte Daten-Integration ist durch das gemeinsame Repository gegeben. Eine Kontroll-Integration existiert leider nicht. Falls ein Diagramm z.B. gleichzeitig mit einem graphischen Editor und einem Editor bearbeitet wird, der das Diagramm in Tabellenform darstellt, muß bei Änderungen in einem dieser Editoren dort ein 'Accept' eingegeben werden und im anderen Editor ein 'Update' gemacht werden, um eine konsistente Darstellung zu erhalten. Trotz allem ist die Stabilität bemerkenswert, was ja gerade für den Einsatz in der Lehre ein kritischer Punkt ist. Soweit mir bekannt wurde, ist es nie zum Absturz von Objectory gekommen.

Auch konzeptionell eignet sich Objectory bzw. OOSE gut für vorlesungsbegleitende Übungen, weil der gesamte Entwicklungsprozeß abgedeckt wird. Wenn jedoch ausdrücklich objektorientierte Modellierung gelehrt werden soll, ist der Einsatz von Objectory problematisch, da die Notationen die Studierenden eher zu strukturierten Modellierungen animiert haben. Zumindest ist das die mit den in Abschnitt 4 vorgestellten Übungsaufgaben gemachte Erfahrung. Hier stand im Vordergrund, durch praktische Erfahrungen mit dem Einsatz eines konkreten Werkzeugs das Verständnis der eher abstrakten Konzepte zu vertiefen, so daß diese Eigenschaft kein Problem darstellte. Generell ist anzumerken, daß der Einsatz von Objectory trotz der aufwendigen Einarbeitung positiv durch die Studierenden aufgenommen wurde (dieser Eindruck bestätigte sich auch bei Befragungen zur Vorlesung mittels Fragebögen).

Zusammengefaßt lassen sich die Erfahrungen mit Objectory nocheinmal wie folgt auflisten:

- Die Komplexität ist für vorlesungsbegleitende Übungen ein gewisses Problem, das berücksichtigt werden muß.

- Objectory ist ausreichend stabil. Instabilität wäre aus meiner Sicht ein KO-Kriterium.
- Auch konzeptionell eignet sich Objectory bzw. OOSE gut für Übungen zur Softwaretechnik-Vorlesung, weil der gesamte Entwicklungsprozeß abgedeckt wird.
- Für die Softwaretechnik-Ausbildung in objektorientierter Modellierung ist der Einsatz von Objectory problematisch, da die Notationen die Studierenden eher zu strukturierten Modellierungen animiert haben.

Zu xomt sei angemerkt, daß dieses Werkzeug in der uns zur Verfügung stehenden Version noch recht instabil ist. Außerdem ist xomt im wesentlichen nur ein Zeichenprogramm für OMT-Klassendiagramme (für das OMT-Objektmodell [RMW+91]), so daß ich den Einsatz nicht empfehlen kann.

6 Weitere Aktivitäten

Objectory wird in Dortmund auch von einer Projektgruppe (einer einjährigen Veranstaltung für eine Gruppe von zwölf Informatikstudierenden im Hauptstudium), und für Diplomarbeiten eingesetzt. Basierend auf der Softwaretechnologie-Vorlesung findet im SS 96 ein Kurs "Objektorientierte Software-Entwicklung mit Objectory" statt, im dem die Teilnehmer alle Entwicklungsphasen bis zum Test und der Anpassung an neue Anforderungen durchlaufen sollen.

In weiteren Diplomarbeiten wurden bzw. werden auch andere CASE-Werkzeuge eingesetzt. Ein Beispiel ist StP-OMT [Int93], das sich zumindest in der uns zur Verfügung stehenden Version 1 als nicht handhabbar und eher hinderlich für den praktischen Einsatz erwiesen hat. Ein weiteres Beispiel ist Rational Rose [Rat95] für die Booch-Methodologie, das sich als sehr ausgereift gezeigt hat. Leider sind die Hochschulpreise für Rational Rose so hoch, daß wir dieses Werkzeug aus finanziellen Gründen nicht in größeren Veranstaltungen einsetzen können.

Danksagung

Die Firma Objectory AB aus Kista (Schweden) stellte freundlicherweise eine gebührenfreie Ausbildungslizenz von Objectory zur Verfügung.

Dank gilt Renate Beckmann, Uli Bieker und Claus Pahl für die Erstellung der Aufgaben und die Betreuung des Übungsbetriebes sowie der Informatik-Rechner-Betriebsgruppe und dem Software-Technologie-Labor der Universität Dortmund für die Unterstützung beim Einsatz der Systemsoftware bzw. mit Objectory.

Literatur

[Boo94] G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, 2. Auflage, 1994.

[BR95] G. Booch und J. Rumbaugh. Unified Method for object-oriented development, documentation set, Version 0.8. Technischer Bericht, Rational Software Corporation, Santa Clara, CA, 1995.

[Dil94] A. Diller. *Z: An introduction to formal methods*. Wiley, 2. Auflage, 1994.

[Dis95] S. Dissmann. Objektorientierung als Schwerpunkt der Software-Technik-Ausbildung. In A. Spillner und U. Breymann, Hrsg., *Software Engineering im Unterricht der Hochschulen SEUH '95*, Seiten 119–128. Teubner-Verlag, 1995.

[Do95] V.-T. Do. Meta-Modellierung und Werkzeugkasten für die Lehre. *Softwaretechnik-Trends*, 15(4):32–37, November 1995.

[GJM91] C. Ghezzi, M. Jazayeri, und D. Mandrioli. *Fundamentals of Software Engineering*. Prentice-Hall, 1991.

[Gli96] M. Glinz. The Teacher: 'Concepts!' The Student: 'Tools!' – On the Number and Importance of Concepts, Methods, and Tools to be Taught in Software Engineering Education. *Softwaretechnik-Trends*, 16(1):32–34, Februar 1996.

[HK96] W. Hasselbring und A. Kröber. Anforderungsanalyse durch Kombination von OMT mit einem Ansatz zum Prototyping. In *Proc. GI-Fachtagung Softwaretechnik 96*, Softwaretechnik-Trends, Koblenz, September 1996. (im Druck).

[Int93] Interactive Development Environments. *Software through Pictures / Object Modeling Technique, Release 1*, 1993.

[JCJÖ92] I. Jacobson, M. Christerson, P. Jonsson, und G. Övergaard. *Object-Oriented Software Engineering – A Use Case Driven Approach*. Addison-Wesley, 1992.

[Mey88] B. Meyer. *Object-oriented Software Construction*. Prentice Hall, 1988.

[Obj95] Objectory AB, Kista (Schweden). *Objectory SE - User Guide and Reference, version 3.6*, 1995.

[Rat95] Rational Software Corporation, Santa Clara, CA. *Rational Rose User's Guide*, 1995.

[RMW+91] J. Rumbaugh, B. Michael, P. William, E. Frederick, und L. William. *Object-Oriented Modelling and Design*. Prentice Hall, 1991.

[SBC92] S. Stepney, R. Barden, und D. Cooper, Hrsg. *Object Orientation in Z*. Springer-Verlag, 1992.

[Sne95] H.M. Sneed. Objektorientiertes Testen. *Informatik-Spektrum*, 18:6–12, Februar 1995.

[SUvG95] J. Seemann, T. Urlaub, und J. Wolff von Gudenberg. XOMT – ein graphischer Editor für OMT-Objektmodelle. In *Proc. GI-Fachtagung Softwaretechnik 95*, Softwaretechnik-Trends 15/3, Seite 162, Braunschweig, Oktober 1995.

[Was89] A.I. Wassermann. Tool Integration in Software Engineering Environments. In F. Long, Hrsg., *Proc. Int. Workshop on Environments*, Band 467 der Reihe *Lecture Notes in Computer Science*, Seiten 137–149. Springer-Verlag, September 1989.