SIGN: Software Implementierung durch Graphische Notation

Mark Richters Martin Gogolla Heino Gärtner

3. Dezember 1998

Zusammenfassung

Das studentische Projekt SIGN (*Software Implementierung durch graphische Notation*) wurde an der Universität Bremen von 1996 bis 1998 durchgeführt. Ziel des Projektes war die Entwicklung einer graphischen Modellierungssprache zur Unterstützung des objektorientierten Systementwurfs. Zur praktischen Umsetzung der Modellierungssprache wurde außerdem eine Entwicklungsumgebung entworfen und implementiert. Sowohl Sprache als auch das Entwicklungssystem wurden durchgehend objektorientiert entwickelt. Dabei wurde im Rahmen der Metamodellierung auch die zu entwerfende Sprache selbst eingesetzt. Dieses Papier faßt die Ziele, Methoden und Ergebnisse des Projektes zusammen.

1 Einleitung

Das Projekt SIGN (*Software Implementierung durch graphische Notation*) war ein studentisches Projekt, das an der Universität Bremen im Zeitraum von 1996 bis 1998 durchgeführt wurde. Teilgenommen haben daran 18 Studierende des Studiengangs Informatik sowie auf Seite der Betreuer ein Professor und zwei wissenschaftliche Mitarbeiter. Dieses Papier soll einen Überblick sowohl über die Inhalte, Ziele und Ergebnisse als auch über den Ablauf des Projektes geben.

Ein von den Studierenden verfaßter Abschlußbericht [SP98] dient dabei als Grundlage dieser Zusammenfassung, die wie folgt aufgebaut ist. In Abschnitt 2 werden zunächst die Projektziele, der thematische Hintergrund sowie ein grober Ablaufplan genannt. Abschnitt 3 skizziert kurz die wesentlichen Grundlagen objektorientierten Entwurfs, die in diesem Projekt von doppelter Bedeutung waren. Zum einen waren sie Ausgangspunkt für die inhaltliche Zielsetzung, zum anderen wurden sie konkret eingesetzt für die praktische Umsetzung in ein rechnergestütztes Werkzeug. Diese Arbeiten werden im Rahmen des Entwurfes im Abschnitt 4 vorgestellt. Das im Projekt erstellte Werkzeug deSIGN wird im Abschnitt 6 kurz erläutert. Abschließend folgen Ausführungen zu geplanten Erweiterungen des ursprünglichen Entwurfs (Abschnitt 7) und Anmerkungen zur Projektinfrastruktur (Abschnitt 8).

2 Projektziele und Verlauf

Ein wesentliches Merkmal studentischer Projekte in der Bremer Informatik ist es, daß die genaue Zielsetzung mit den Teilnehmenden gemeinsam erarbeitet wird. Ein inhaltlicher Rahmen war von den Veranstaltern vorgegeben und nannte als Schwerpunkt die Auseinandersetzung mit graphischen Entwurfsmethoden für Softwaresysteme unter besonderer Berücksichtigung von Informationssystemen. Neben der Erarbeitung und kritischen Diskussion der relevanten Literatur und der im Softwareentwurf eingesetzten Methoden war ein wesentlicher Bestandteil des Projektzieles auch die Erstellung eines konkreten Softwaresystems, in dem die erlernten Ansätze in ein praktisches Werkzeug umgesetzt werden sollten.

Zur Realisierung der geschilderten allgemeinen Projektziele ergab sich der folgende grobe Ablaufplan. In einer ersten Phase wurde im wesentlichen Wissensaneignung betrieben und eine Verfeinerung der Projektziele vorgenommen. Als Hauptziel wurde die Entwicklung einer graphischen Modellierungssprache (der SIGN-Modellierungssprache) für den Softwareentwurf festgeschrieben. Darüberhinaus sollte für die praktische Verwendung dieser Sprache ein Editor entwickelt werden, mit dessen Hilfe die Erstellung von Diagrammen in der Notation der SIGN-Modellierungssprache unterstützt werden sollte.

Nach der Festlegung der konkreten Projektziele folgte dann die Entwurfs- und Implementierungsphase. Der Entwurf wurde wiederum unterteilt in zwei Abschnitte, um bereits frühzeitig mit der Implementierung beginnen zu können. Abgeschlossen wurde das Projekt mit der Vorstellung der Arbeit im Rahmen des Projekttages, auf dem alle Projekte ihre Ergebnisse präsentierten und mit der Fertigstellung des Abschlußberichtes.

3 Objektorientierte Entwurfssprachen

Für die objektorientierte Analyse und den Entwurf von Softwaresystemen gibt es eine große Zahl unterschiedlicher Methoden und Sprachen. Ausgewählt für eine genauere Untersuchung wurden die Methoden OMT [RBP+91], OOD [CY91], Booch [Boo94], STATEMATE [HN96] und UML [Rat97a] (ein Ansatz, der sich erst während der Projektlaufzeit herauskristallisierte). Dabei wurden die Konzepte und Sprachmittel von OMT sehr ausführlich untersucht, während die Eigenschaften der anderen genannten Sprachen und Methoden hauptsächlich im Vergleich zu OMT betrachtet wurden.

In OMT *(Object Modeling Technique)* werden eine graphische Sprache und eine Entwurfsmethode zusammengefaßt. Ein System wird in OMT mit drei sich ergänzenden Modellen beschrieben, die eine statische Sicht (Objektmodell), eine dynamische Sicht (Dynamisches Modell) und eine funktionale Sicht (Funktionales Modell) beinhalten.

Objektmodell: Das Objektmodell beschreibt die Struktur eines Systems als eine Menge von Objekten. Objekte werden nach ihren gemeinsamen Eigenschaften in Klassen eingeteilt. Klassen definieren Attribute und Operationen von Objekten. Beziehungen werden durch Assoziationen modelliert. OMT bietet darüberhinaus Aggregationen, Generalisierungen und einige spezielle Konzepte wie z. B. Mehrfachvererbung an.

Dynamisches Modell: Das dynamische Modell beschreibt zeitliche Aspekte eines Systems. Zustandsdiagramme definieren mögliche Systemzustände und legen Übergänge zwischen den Zuständen fest. Das Modell basiert auf endlichen Automaten und fügt Konzepte wie hierarchische Zustände und Nebenläufigkeit hinzu.

Funktionales Modell: Das funktionale Modell beschreibt die Aspekte eines Systems, die sich durch die Transformation von Werten darstellen lassen. Berechnungen werden dabei unabhängig von zeitlichen und strukturellen Eigenschaften betrachtet. Für die Darstellung werden Datenflußdiagramme verwendet.

Es wird deutlich bei der Untersuchung von OMT, daß die Modelle in enger Beziehung zueinander stehen, denn jedes Modell beschreibt einen anderen Aspekt eines Systems, wobei wiederum auf Komponenten der anderen Modelle verwiesen wird. Das Objektmodell beschreibt die Datenstruktur, auf der das dynamische und funktionale Modell operieren können. Die Operationen in einem Objektmodell korrespondieren mit Ereignissen im dynamischen Modell und mit Funktionen in funktionalen Modell. Das dynamische

Modell beschreibt die Kontrollstruktur von Objekten. Entscheidungen, die einen Zustandsübergang betreffen, sind von Objektwerten abhängig. Ein Zustandsübergang wiederum stößt Aktionen an, die Objektwerte ändern und Operationen aufrufen. Das funktionale Modell beschreibt die von Operationen ausgeführten Berechnungen. Funktionen beziehen sich auf durch das Objektmodell gegebene Datenwerte.

Um die zahlreichen Modellierungsmittel von OMT besser zu verstehen, wurden eine Reihe von Fallstudien erstellt und diskutiert. Die Anwendungsgebiete dieser Fallstudien sind unter anderem eine Autovermietung, die Steuerung eines elektrischen Garagentors, die Organisation wissenschaftlicher Konferenzen und ein System zur Visualisierung von Netztopologien.

4 Entwurf einer Modellierungssprache

Im Sprachentwurf wurde die Syntax der SIGN-Modellierungssprache festgelegt. Der Entwurf basiert auf den Grundkonzepten objektorientierter Modellierungssprachen, wie sie in der Vorbereitungsphase herausgearbeitet wurden. Sie lehnt sich stark an die Modellierungssprache OMT an und berücksichtigt teilsweise Eigenschaften von OOD, Booch und UML. In einem ersten Schritt wurden dabei zunächst nur statische Aspekte, wie sie z. B. durch das OMT-Objektmodell gegeben sind, berücksichtigt. Die Sprache wird daher im folgenden auch *Basisnotation* genannt. Sie umfaßt die Konzepte: Klasse, Attribut, Operation und Methode, Assoziation, Kardinalität, Assoziationsattribut, Rollenname, Qualifikation (qualifizierte Assoziation), Aggregation, Generalisierung und Vererbung, Abstrakte Klasse, Spezialisierung, Mehrfachvererbung, Kandidatenschlüssel und Einschränkungen.

Parallel zur Definition der Syntax der Basisnotation wurde ein Ansatz zur formalen Beschreibung ihrer Semantik beschrieben. Der Ansatz orientiert sich weitgehend an der formalen Definition eines erweiterten Entity-Relationship-Modells [Gog94].

5 Systementwurf

Ziel des Systementwurfes war die Ausarbeitung einer Spezifikation der graphischen Entwicklungsumgebung. Diese Umgebung – genannt deSIGN – dient als Editor für die Systemmodellierung mit der SIGN-Modellierungssprache. Der Entwurf wurde dabei in mehreren Phasen erstellt, die folgende Aufgaben umfaßten.

- Identifizieren zentraler Entwurfsprobleme und grundlegende Entwurfsentscheidungen.
- Festlegung der Gesamtarchitektur und Unterteilung in Schichten.
- Beschreibung der Struktur- und Datensicht mit Hilfe von Klassendiagrammen.
- Beschreibung der funktionalen und dynamischen Sicht mit Hilfe von Anwendungsfalldiagrammen (Use Cases), Datenflußdiagrammen und Aktivitätsdiagrammen.

Ausgangspunkt für die Modellierung war der Entwurf der SIGN-Modellierungssprache. Die rekursive Verwendung dieser Sprache für ihre eigene Definition sollte zu einem Metamodell führen, das unmittelbar für den Systementwurf geeignet war. Eine ähnliche Vorgehensweise findet sich beispielsweise auch bei der Definition des UML-Metamodells [Rat97b]. Während das Metamodell die abstrakten Kernkonzepte der Modellierungssprache beschreibt, existieren für die Darstellung eines Modells verschiedene Sichten in Form von Diagrammen. Für den Entwurf des Editors wurde daher konsequent zwischen sogenannten logischen Elementen (Teile des Metamodells) und Diagrammelementen für die graphische Repräsentation unterschieden. Eine Veränderung des Modells wird im Editor ausschließlich durch Manipulation auf der Diagrammebene hervorgerufen. Das Modell dient weiterhin als Grundlage für die Transformation in verschiedene andere Sprachen, wie etwa SQL und Java. Die graphische Darstellung (Layout und Präsentation) sollte sich weitgehend an der in OMT und UML üblichen Notation anlehnen.

Die Architektur der Entwicklungsumgebung ist durch eine Einteilung in drei Schichten bestimmt. Unterschieden werden dabei Benutzungsschnittstelle, Anwendungsgebiet und Datenverwaltung. Das Anwendungsgebiet umfaßt den graphischen Editor sowie einen Code-Generator für die Übersetzung in andere Sprachen (SQL und Java).

Alle Komponenten wurden schließlich mit der SIGN-Modellierungssprache entworfen und beschrieben. Der Entwurf der Benutzungsschnittstelle behandelt die Themenbereiche Sichten, Fenster, Kommandos, Dialoge. Es wurden eine Schnittstelle zwischen den Klassen des verwendeten graphischen Toolkits und der Klassen der Komponente Anwendungsgebiet beschrieben. Für die Anwendungskomponente werden das Metamodell und die Diagrammebene spezifiziert. Die persistente Speicherung von Modellen ermöglicht schließlich die Komponente Datenverwaltung.

6 Der SIGN-Editor

Zur Modellierung von Systemen mit der SIGN-Modellierungssprache wurde ein Werkzeug entwickelt, das die Bearbeitung von graphischen Systementwürfen ermöglicht. Dieses Werkzeug – der deSIGN-Editor – besitzt eine graphische Benutzungsschnittstelle für die direkte Manipulation von Diagrammen. Diese befinden sich jeweils auf einem Arbeitsblatt, welche in eigenen Fenstern dargestellt werden, während die verfügbaren Bearbeitungsfunktionen in einem separaten Fenster mit Menü und Schaltflächen angeboten werden (siehe Abb. 1).

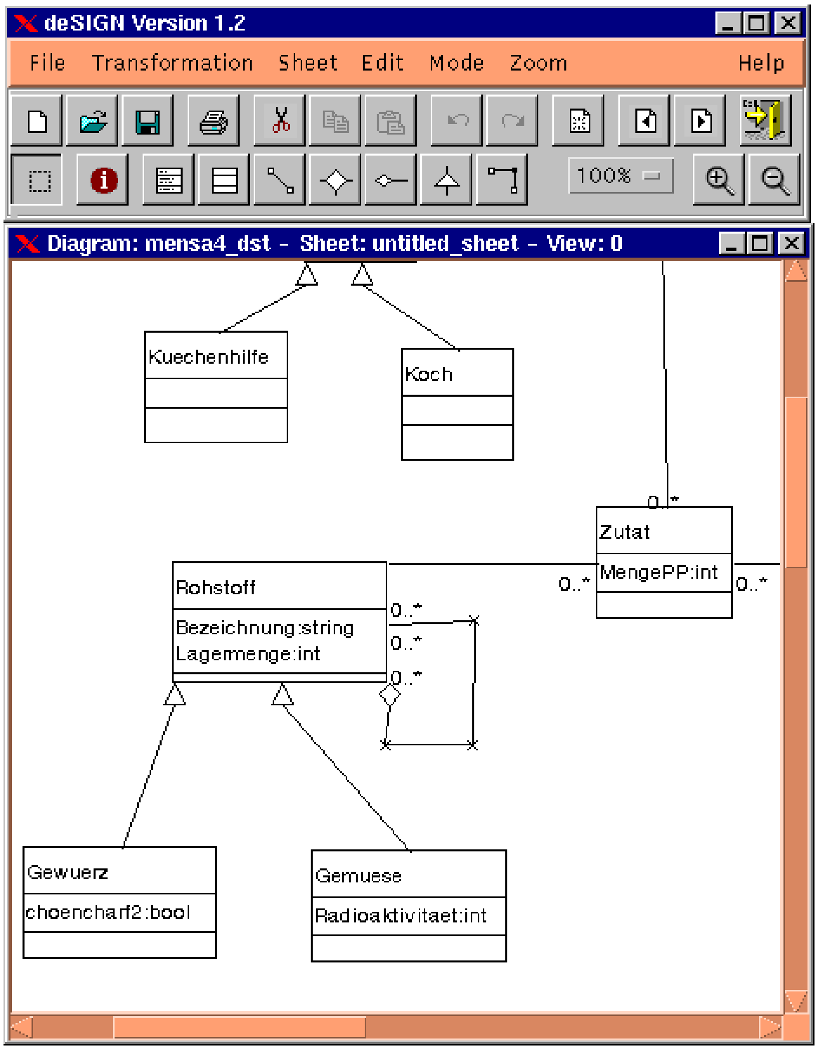

Abbildung 1: Ausschnitt eines Klassendiagramms bearbeitet im SIGN-Editor

Die Funktionen zur Bearbeitung umfassen das Erstellen neuer Diagramme, das Laden, Speichern und Drucken vorhandener Diagramme sowie umfangreiche Funktionen zum Einfügen und Verändern von Diagrammelementen. Der Befehl "Transformation" erlaubt die automatische Übersetzung eines Modells in verschiedene Zielsprachen, z. B. nach Java.

7 Erweiterungen

In einer zweiten Entwurfsphase wurden Erweiterungen des ersten Entwurfs erarbeitet, die im wesentlichen dynamische und funktionale Aspekte der Modellierung betreffen. Neben den notwendigen Erweiterungen im Sprachentwurf wurde auch der Systementwurf für den Editor ergänzt. Diese Erweiterungen sind nicht mehr im deSIGN-Editor implementiert worden, jedoch wurde für die Erstellung der Klassendiagramme im strukturellen Entwurf bereits der Editor verwendet und so ein Teil der Qualitätssicherung geleistet.

Als Erweiterungen wurden drei neue Modellierungsmittel eingeführt.

- Statecharts für den dynamischen Anteil eines Systems.
- Activity-Charts für die funktionale Sicht eines Systems.
- Constraints für die Einschränkung der zulässigen Systemzustände.

Statecharts. In Anlehnung an OMT, STATEMATE und UML wurde die SIGN-Modellierungssprache konzeptionell erweitert. Als Alternative zu den behandelten und untersuchten Modellen von OMT sind die Konzepte von STATEMATE untersucht und erarbeitet worden. STATEMATE ist ein Tool, das den Harelschen Ansatz verwendet [HN96]. Die einzelnen Modelle und deren Konzepte sind formeller beschrieben als die von OMT, wobei STATEMATE im Gegensatz zu OMT nicht objektorientiert ausgerichtet ist. In STATEMATE werden die Statecharts benutzt, um Kontrollaktivitäten in den Activity-Charts zu beschreiben. Diese Aktivitäten bilden das Verhaltensgerüst eines STATEMATE-Modells. Die Darstellungsform der Statecharts basiert auf Zustandsdiagrammen, enthält jedoch weitere Konzepte wie z. B. Hierarchien und Ausdrücke.

Activity-Charts. Die Sprache der Activity-Charts in STATEMATE beschreibt die funktionale Sicht auf ein System. Die funktionale Beschreibung eines Systems gibt detailliert die Funktionen oder Aktivitäten an, die das

System ausführen kann, und die Informationen, die zwischen diesen Funktionen "fließen". Man kann die Activity-Charts als ein mehrstufiges Datenflußdiagramm verstehen, das Eingabe und Ausgabe der Daten beschreibt. Ein Activity-Chart kann mehrere Aktivitäten beinhalten, die hierarchisch zerlegt werden können.

Object Constraint Language Für die Einschränkung möglicher Systemzustände wird eine Sprache zur Formulierung von Integritätsbedingungen betrachtet. Diese basiert im wesentlichen auf der mit UML eingeführten OCL *(Object Constraint Language)*. OCL erlaubt die Formulierung von Zusicherungen und Anfragen im Kontext von Klassenmodellen und Operationen. Sie ergänzt damit die Möglichkeiten der graphischen Modellierung um eine formale textuelle Sprache für Bedingungen, die sich in der Regel nicht durch Diagramme ausdrücken lassen.

8 Infrastruktur des Projektes

Für die Durchführung des Projektes waren neben der rein inhaltlichen Arbeit eine Reihe organisatorischer Tätigkeiten erforderlich. Aufgaben des Projektmanagements wurden z. B. von einer Arbeitsgruppe übernommen, Protokolle von Besprechungen mußten angefertigt werden, Plenumssitzungen mußten geleitet werden. Weiterhin wurden eine Reihe von technischen Hilfsmitteln eingesetzt. Die informelle Kommunikation wurde mittels eines E-mail-Verteilers unterstützt und sowohl archivierte als auch aktuelle Informationen über WWW zur Verfügung gestellt.[1]

Alle im Laufe des Projektes entstandenen Dokumente und Programme wurden mit dem Versionskontrollsystem CVS [Ber90] verwaltet. Sämtliche Dokumente (Protokolle, Ausarbeitungen, Projektbericht, etc.) wurden mit SGML *(Standard Generalized Markup Language)* [GR90] erstellt, um Einheitlichkeit und Flexibilität in der Präsentation und Weiterverarbeitung zu erzielen.

9 Fazit

Das Projekt SIGN befaßte sich während seiner zweijährigen Laufzeit mit graphischen Sprachen für die objektorientierte Modellierung von Informationssystemen. Ausgangspunkt für die Entwicklung einer neuen Sprache war

[1]siehe Projekt-Homepage: `http://www.informatik.uni-bremen.de/grp/stp/home.html`

die Untersuchung von bereits etablierten Sprachen wie OMT und UML. Die hauptsächlich konzeptionelle Arbeit, die für den Sprachentwurf erforderlich war, wurde ergänzt durch die Entwicklung und Implementierung eines konkreten Systems für die Erstellung und Bearbeitung von Diagrammen in der selbstentwickelten SIGN-Modellierungssprache.

Literatur

[Ber90] Brian Berliner. CVS II: Parallelizing software development. In USENIX Association, editor, *Proceedings of the Winter 1990 USENIX Conference, January 22–26, 1990, Washington, DC, USA*, pages 341–352, Berkeley, CA, USA, January 1990. USENIX.

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

[CY91] Peter Coad and Edward Yourdon. *Object-Oriented Design*. Yourdon Press, 1991.

[Gog94] Martin Gogolla. *An Extended Entity-Relationship Model – Fundamentals and Pragmatics*, volume 767 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994.

[GR90] Charles F. Goldfarb and Yuri Rubinsky. *The SGML handbook*. Clarendon Press, Oxford, UK, 1990.

[HN96] David Harel and Amnon Naamad. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996.

[Rat97a] Rational Software Corporation, Unified Modeling Language (UML) version 1.1, `http://www.rational.com`, 1997.

[Rat97b] UML Semantics, 1997. (Published as part of [Rat97a]).

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs (NJ), 1991.

[SP98] SIGN-Projekt. SIGN–Projektbericht, October 1998. Abschlußbericht des studentischen Projektes SIGN (1996-1998), Universität Bremen, Fachbereich Informatik.