Forschungsvorhaben
Objektorientierte Weiterentwicklung des Objektkatalogs im Straßen- und Verkehrswesen (OKSTRA)

Leitfaden zur objektorientierten Modellierung des OKSTRA

interactive instruments
Gesellschaft für Softwareentwicklung mbH
Trierer Straße 70-72
53115 Bonn

November 2004

Verantwortlich für den Inhalt:

Dipl.-Phys. Bernd Weidner
interactive instruments GmbH, Bonn

Redaktion: Bernd Weidner (interactive instruments)

Inhaltsverzeichnis

1 Überblick 9
1.1 Grundzüge der OKSTRA-Modellierung 9
1.2 Der objektorientierte OKSTRA 11
1.2.1 Daten- und dienstebasierte Systeme 12
1.2.2 Statische und objektorientierte Modelle 15
1.3 Zielsetzung des Leitfadens 17
1.4 Aufbau des Leitfadens 18
1.5 Glossar 21
1.6 Typebene und Instanzebene 24
2 Die fachliche Modellierung des objektorientierten OKSTRA 25
2.1 Vorbemerkungen 25
2.1.1 Modelle 25
2.1.2 Begriffsdefinitionen 27
2.1.3 Modelldokumentation 27
2.1.4 Begriffe zur Prozessmodellierung 27
2.2 Geschäftsbereichsanalyse 29
2.2.1 Aufgaben und Ziele 29
2.2.2 Welcher Bereich ist zu untersuchen? 30
2.2.3 Welche Prozesse gibt es im Bereich? 30
2.2.4 Wodurch können die Prozesse erschlossen werden? 30
2.2.5 Welche Informationen bearbeiten die Prozesse? 30
2.2.6 Welche anderen Bereiche berühren den gerade untersuchten? 31
2.3 Einzelprozessanalyse 31
2.3.1 Aufgaben und Ziele 31
2.3.2 Welche Aktivitäten gibt es im Prozess? 32
2.3.3 Wie wird der Prozess von außen beeinflusst? 32
2.3.4 Wie folgen die Aktivitäten aufeinander? 32
2.3.5 Was sind die Anwendungsfälle für das OKSTRA-Modell? 33
2.4 Dokumentation der Prozesse 34
2.5 Objektwelt 35
2.5.1 Aufgaben und Ziele 35
2.5.2 Welche Objekte tauchen in den Anwendungsfällen auf? 35
2.5.3 Wie lange leben die Objekte? 36
2.6 Operationen 36
2.6.1 Aufgaben und Ziele 36
2.6.2 Welche Verantwortlichkeiten nehmen die Objekte wahr? 36
2.6.3 Welche Nachrichten müssen zwischen den Objekten ausgetauscht werden? 36
2.6.4 Wie sehen die Operationen aus? 37
2.6.5 Was tun die Operationen? 37
2.7 Definition von Diensten 37
2.7.1 Aufgaben und Ziele 37

2.7.2 Welche Objekte arbeiten zusammen? ... 38
2.7.3 Welche Dienste sind erforderlich? ... 38
2.8 Review des Konzeptionsmodells ... 38
2.9 Formalisierung Bildung und Dokumentation des Modells ... 39
2.9.1 Aufgaben und Ziele ... 39
2.9.2 Objektmodell als Klassendiagramm formulieren ... 39
2.9.3 Modell publizieren ... 39
3 Begriffe der Objektorientierung für OKSTRA-Modelle ... 41
3.1 Ziele und Umfang ... 41
3.2 Objekte und Operationen ... 41
3.3 Identität, Lebenslauf und Instanzierung ... 43
3.4 Klassen und Exemplare ... 44
3.5 Klassenoperationen ... 44
3.6 Werte und Datentypen ... 45
3.7 Attribute und Assoziationen ... 45
3.8 Generalisierung und Spezialisierung ... 47
3.9 Polymorphie ... 48
3.10 Metaklassen ... 49
3.11 Typen und Schnittstellen ... 50
3.12 Anwendungen, Dienste und Komponenten ... 51
4 Bildung der Objektwelt ... 53
4.1 Objektbildung nach pragmatischen Kriterien ... 53
4.2 Objektbildung nach Aufgaben im Geschäftsprozess ... 54
4.3 Objektbildung nach Kriterien der Systemfunktion ... 55
4.4 Häufig anzutreffende Objektstrukturen ... 55
4.5 Hinweise zur Ermittlung der Operationen ... 56
5 Dokumentation mit UML ... 59
5.1 Einführung ... 59
5.2 Allgemeine Grundlagen ... 60
5.2.1 Beziehungen ... 61
5.2.2 Stereotype ... 62
5.3 Diagramme zur Prozessmodellierung ... 63
5.3.1 Aktivitätsdiagramme ... 63
5.3.2 Anwendungsfalldiagramme ... 64
5.4 Diagramme zur Objektmodellierung ... 65
5.4.1 Klassifizierer und Klassendiagramme ... 65
5.4.2 Paketdiagramme ... 72
5.4.3 Sequenzdiagramme ... 73
6 Formalisierung und Dokumentation der OKSTRA-Modelle ... 75
6.1 Modellarten ... 75
6.2 Inhalt der Modelle ... 75
6.2.1 Prozessmodelle ... 75
6.2.2 Objektmodelle ... 76

6.3 Semantische Dokumentation ... 78
6.4 Erstellung, Pflege und Verteilung der Modelle ... 78
7 Rahmenmodelle ... 81
7.1 Basisdatentypen ... 81
7.1.1 Einfache Datentypen ... 81
7.1.2 OKSTRA-Datentypen ... 81
7.1.3 Aufzählungen ... 81
7.1.4 UML-Diagramm Datentypen ... 81
7.2 Das OKSTRA-Objekt ... 82
7.3 Objekt-Identifikatoren und Namensräume ... 83
7.4 Ansammlungen ... 84
7.4.1 Mengen ... 85
7.4.2 Folgen ... 85
7.4.3 Tabellen ... 85
7.4.4 Callbacks ... 86
7.4.5 Darstellung von EXPRESS-Aggregaten ... 86
7.4.6 Akzessoren für multiple Attribute und :n Assoziationen ... 86
7.5 Verzeichnisse, Kataloge ... 89
7.6 Ereignisse ... 89
7.7 Fabriken ... 92
7.8 Serialisierung ... 93
7.9 Typsystem ... 94
7.10 Sicherheit ... 95
7.11 Persistenz ... 96
7.12 Transaktionen ... 97
7.13 Workflow ... 97
7.14 Raumbezug ... 98
7.15 Historisierung ... 98
7.16 Dokumente ... 99
8 Roadmap zur Modellierung des objektorientierten OKSTRA ... 101
8.1 Pflege des Leitfadens ... 101
8.1.1 Benennungsregeln für Objekte und Operationen ... 101
8.1.2 Festlegung der UML-Version ... 101
8.1.3 Festlegung der Dateiformate für die Publikation der Modelle ... 101
8.1.4 Festlegung der unterstützten Sprachbindungen ... 101
8.1.5 Evaluierung und Empfehlung von UML-Modellierungs-Werkzeugen ... 101
8.1.6 Regelungen für den Review-Prozess ... 101
8.2 Entwicklung der Rahmenmodelle ... 102
8.3 Fachliche Modellierung ... 102
9 Literaturverzeichnis ... 103
Anhang A Modellierung der Kostenermittlung ... 105
A.1 Einleitung ... 107
A.1.1 Vorbemerkungen ... 107

A.1.2 Motivation ... 107
A.1.3 Überblick über den Inhalt ... 108
A.2 Die Anwendungsfälle der Kostenermittlung ... 110
A.2.1 Definition des zentralen Anwendungsfalles *Kostenermittlung* ... 110
A.2.2 Die konkreten Anwendungsfälle zur Kostenermittlung ... 111
A.2.3 Anwendungsfälle zur Maßnahmenkonfiguration ... 111
A.2.4 Die Kostengliederung ... 112
A.2.5 Mengen- und Preisbestimmung ... 113
A.3 Das Objektmodell ... 117
A.3.1 Vorbemerkungen ... 117
A.3.2 Maßnahmen ... 117
A.3.3 Die Modellierung zum Anwendungsfall Kostenberechnung ... 122
A.3.4 Zur Problematik der existierenden Fachobjekte ... 136
A.3.5 Ereignisse ... 136
A.4 Die prototypische Realisierung als verteiltes System ... 138
A.4.1 Implementationstechnik ... 138
A.4.2 Komponentenbildung ... 141
A.4.3 Verteilung auf Standorte ... 149
A.5 Zusammenfassung ... 152
A.6 Literatur ... 155
Anhang B Modellierung von Fortführungsprozessen ... 157
B.1 Einleitung ... 159
B.1.1 Motivation ... 159
B.1.2 Ziele und Umfang ... 159
B.1.3 Vorgehensweise ... 159
B.2 Ergebnisse der Arbeitsphasen ... 161
B.2.1 Geschäftsbereichsanalyse ... 161
B.2.2 Einzelprozessanalyse ... 167
B.2.3 Bildung der Objektwelt ... 180
B.2.4 Operationen ... 192
B.2.5 Dienste ... 201
B.3 Zusammenfassung ... 205

Abbildungsverzeichnis

Abbildung 1. Lektüreplan. 20
Abbildung 2. Die OKSTRA-Modellierung. 26
Abbildung 3. Beziehungslinien in UML. 61
Abbildung 4. Generalisierung. 62
Abbildung 5. Muster Aktivitätsdiagramm. 63
Abbildung 6. Muster Anwendungsfalldiagramm. 65
Abbildung 7. Muster Klassendiagramm. 70
Abbildung 8. Beispiel Klassendiagramm. 71
Abbildung 9. Muster Paketdiagramm. 73
Abbildung 10. Muster Sequenzdiagramm. 74
Abbildung 11. Ansammlungen, OKSTRA-Typ, OKSTRA-Objekt. 88
Abbildung 12. Klassendiagramm Ereignisse. 91
Abbildung 13. Übermittlung von Signalen für Ereignisse. 92
Abbildung 14. Fabriken, Serialisierung. 94
Abbildung 15. Anwendungsfälle der Kostenermittlung. 113
Abbildung 16. Baukosten. 114
Abbildung 17. Grunderwerbskosten. 116
Abbildung 18. Baumaßnahme und Geschäftsbereiche. 119
Abbildung 19. Prototyp Sicherheitsmodell. 120
Abbildung 20. Baumaßnahmenverzeichnis. 121
Abbildung 21. Geschäftsbereich AKS. Kostenberechnungskatalog. 125
Abbildung 22. AKS- und Mengen-Interface für die Fachobjekte. 127
Abbildung 23. Sequenzdiagramm AKS-Berechnung. 130
Abbildung 24. Sequenzdiagramm Mengen und Preise. 132
Abbildung 25. Grunderwerbskostenberechnung. 133
Abbildung 26. Konfigurierbare Mengenberechnung. 135
Abbildung 27. Ereignisübermittlung. 137
Abbildung 28. Fachobjekte im Prototypen. 142
Abbildung 29. Komponenten des Prototypen. 144
Abbildung 30. Verteilung der Komponenten. 151
Abbildung 31. Leitungen erzeugen. 170
Abbildung 32. Fortführung Leitung. 171
Abbildung 33. Geometrieübernahme. 172
Abbildung 34. Anwendungsfälle Leitungen. 173
Abbildung 35. Anwendungsfälle Baumobjekte. 175
Abbildung 36. Baum neu eintragen/ändern. 177

Abbildung 37. Verallgemeinerte Anwendungsfälle und Akteure. 179
Abbildung 38. Leitungen. 181
Abbildung 39. Objektlebenslauf. 185
Abbildung 40. Datenerfassung. 186
Abbildung 41. Lageinformation. 187
Abbildung 42. Fachobjekt teilen. 188
Abbildung 43. Fachobjekte vereinigen. 189
Abbildung 44. TURIN erzeugt neuen Baum. 195
Abbildung 45. Verwendung des Fachanwendungskataloges. 196
Abbildung 46. Klassendiagramm Abfragen und Regisrtireungen. 203
Abbildung 47. Klassendiagramm zur Ablaufkontrolle. 204

1 Überblick

Dieser Leitfaden wendet sich an alle, die am Aufbau von *Modellen* für den *objektorientierten OKSTRA* mitwirken. Dies sind

- Fachleute des Straßen- und Verkehrswesens, die ihr Wissen und ihre Erfahrung einbringen, um dem OKSTRA eine solide fachliche Grundlage zu geben,
- Spezialisten der Informationstechnik (IT), die dem OKSTRA eine Gestalt geben, die für die Nutzung in praxisgerechten IT-Verfahren geeignet ist.

Viele der Passagen dieses Leitfadens richten sich vornehmlich an die IT-Spezialisten. Sie sind zur Hilfe für die Leser und Leserinnen, die diese Passagen überspringen möchten, grau hinterlegt.

Außerdem wurden Anmerkungen und Beispiele kleiner gesetzt und eingerückt.

Das Straßen- und Verkehrswesen ist ein großes und komplexes Fachgebiet, der Aufbau eines vollständigen objektorientierten OKSTRA daher eine anspruchsvolle Aufgabe. Der vorliegende Leitfaden beschreibt grundlegende Begriffe und Verfahrensvorgaben hierfür.

1.1 Grundzüge der OKSTRA-Modellierung

Der Objektkatalog für das Straßen- und Verkehrswesen – kurz OKSTRA – ist entstanden aus der Einsicht, dass eine wirkungsvolle Unterstützung der täglichen Aufgaben in diesem Fachgebiet durch informationstechnische Systeme es erfordert, die Strukturen und Formate der dabei zu verarbeitenden Informationen zu standardisieren.

Die gewünschte Standardisierung erstrebt im Einzelnen:

- Optimierung des Datenflusses: Daten werden in dem Format bereitgestellt, in der sie unmittelbar weiterverarbeitet werden können.
- Vermeidung von Medienbrüchen: Daten werden nach einem einheitlichen logischen Schema so bereitgestellt, dass keine Neu- oder Zusatzerfassungen derselben Sachverhalte nötig sind.
- Harmonisierung der Geschäftsprozesse: Wiederkehrende Aufgaben können nach denselben Mustern ablaufen und somit durch dieselben Softwareanwendungen unterstützt werden.
- Harmonisierungen von bereichsübergreifenden Objektdefinitionen: Die Sichten der Geschäftsprozesse von Verkehrsplanung, Straßenplanung, Straßenbau, Straßenbetrieb und Bewertung der Verkehrsqualität werden aufeinander abgestimmt und vereinigt, damit einmal erfasste Information möglichst vielfältig genutzt werden kann.
- einheitliche Berücksichtigung des geographischen Bezugs von Objekten.
- einheitliches Verfahren für den Zugriff auf und den Austausch von Straßen- und Verkehrsdaten: Dieselbe Softwarearchitektur kann für eine Vielzahl von Anwendungen genutzt werden.
- Interoperabilität zwischen unterschiedlichen IT-Landschaften bei Straßenbau- und Verkehrsverwaltungen, Ingenieurbüros sowie der Industrie: Die Nutzung der Daten ist nicht an bestimmte Produkte oder Produktfamilien gebunden.

Die Durchführung eines derartigen Standardisierungsprogramms setzt jedoch voraus, dass zunächst die fachlichen Konzepte selbst aufgefunden und ihre Beziehungen zueinander geklärt werden müssen.

Teilweise sind die Konzepte schon seit langem vorhanden und fest im Sprachgebrauch des Straßen- und Verkehrswesens verankert – z.B. die *Achse* –, in anderen Fällen sind sie erst (und manchmal mühsam) zu erabeiten – z.B. beim *Baum*.

Dieser Vorgang der *konzeptionellen Modellierung* oder *Realweltmodellierung* stützt sich auf das in den einschlägigen Regelwerken niedergelegte Wissen, auf den Erfahrungsschatz der Fachleute und die für das Fachgebiet typische Terminologie (die Fachsprache). Die Konzepte werden durch *Abstraktion* aus den real vorhandenen Dingen gebildet, in dem für eine Menge von Objekten deren typische, kennzeichnende Eigenschaften und Beziehungen ermittelt werden, soweit diese für das Straßen- und Verkehrswesen von Interesse sind. Die Konzepte und ihre Beziehungen werden mit Hilfe einer *formalen Sprache (Notation)* dokumentiert; es kann sich um eine textbasierte Darstellung (z.B. Express oder XML-Schema) oder um eine grafische (z.B. NIAM oder UML) handeln.

Die beschriebene Vorgehensweise ist nicht beschränkt auf das Fachgebiet Straßen- und Verkehrswesen, sondern typisch für Aufgaben der Organisation und Repräsentation von Wissen und Bedeutung in Informationssystemen (allen voran im World Wide Web), mit denen sich ein wachsender Zweig der Informatik beschäftigt. In deren Sprechweise bildet der OKSTRA eine Ontologie:

„Ontologien sind formale Modelle einer Anwendungsdomäne, die dazu dienen den Austausch und das Teilen von Wissen zu erleichtern. Auf der methodischen Seite werden Techniken der objektorientierten Modellierung konsequent so weiterentwickelt, dass die Modelle nicht bloß zur Strukturierung von Software dienen, sondern auch ein explizites Element der Benutzerschnittstelle darstellen und zur Laufzeit verwendet werden. Auf der soziokulturellen Seite erfordern Ontologien daher die Einigung einer Gruppe von Anwendern auf die jeweiligen Begriffe und deren Zusammenhänge."

aus: WIRTSCHAFTSINFORMATIK 43 (2001) 4, S. 393-396

„Eine Ontologie ist eine Sammlung von Konzepten und den Relationen zwischen eben diesen. Eine Ontologie muß formal definiert sein, damit sie ... computergestützt verarbeitet werden kann.", zitiert nach R. Schirdewan in seiner Diplomarbeit:

http://nats-www.informatik.uni-hamburg.de/~jum/research/odet/DiplomRoland2-Title.html

Zur Erläuterung: Die Anwendungsdomäne, von der im ersten Zitat die Rede ist, ist für den OKSTRA offenbar das Straßen- und Verkehrswesen. Zur geforderten formalen Definition werden NIAM, EXPRESS, GML und UML eingesetzt. Insbesondere die XML-Formalisierung als GML-Applikationsschema gestattet eine leichte Umsetzung in Elemente von webbasierten Benutzerschnittstellen – als Landkarte, in Objektbrowsern usw. Die angesprochene „Einigung einer Gruppe von Anwendern" ist seit jeher ein zentrales Anliegen des OKSTRA gewesen.

Weitere Definitionen:

http://beat.doebe.li/bibliothek/w00085.html

http://wiman.server.de/servlet/is/155/

Gute Einführungen zu Ontologien finden sich in:

http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/1999/M&O.pdf

http://www.math.tu-dresden.de/~rudolph/Dresden_Workshop.pdf

Eine aktuelle Liste von Ontologie-Projekten und –Ressourcen findet sich unter

http://www.cs.utexas.edu/users/mfkb/related.html,

darunter solche aus Medizin, Biologie, Landwirtschaft und Betriebswirtschaft.

Zahreiche weitere Internet-Ressourcen auf Deutsch finden sich bei einer Suche nach „Ontologie Domäne“ mit der Suchmaschine Google.

Die Verbindung der OKSTRA-Modellierung mit der Ontologieforschung erlaubt es, dort erarbeitete analytische Methoden und Darstellungstechniken für den OKSTRA nutzbar zu machen. Die Ontologieforschung versucht u.A.

- informationstechnisch präzise und praktikable Definitionen für die Begriff Ontologie, Anwendungsdomäne usw. zu geben,
- Methoden zur systematischen Konstruktion von Ontologien anzugeben („Leitfäden“ wie der vorliegende),
- Bedeutung unabhängig von Sprache formal und technisch zu organisieren

Beispiel: Wenn wir in Google nach "Lichtsignalanlage" suchen, dann finden wir keine Dokumente, in denen „Lichtzeichenanlage“, „Ampel“ oder „traffic light“ oder „feu“ (natürlich nur in der gewünschten Bedeutung!) ohne den Suchbegriff selbst vorkommen. Im Semantic Web stellt sich nun gerade die Aufgabe, die Inhalte im World Wide Web so zu organisieren, dass genau das möglich wird – die Suche nach Inhalten.

Aus den Konzeptmodellen werden in einem zweiten Schritt *Spezifikationsmodelle.* Hierzu wird die Konzeptmodellierung um Angaben ergänzt, die für die für die *Nutzung* eines dem Konzeptmodell entsprechenden technischen Informationssystems notwendig sind. Dazu werden die Elemente des Konzeptmodells auf Konstrukte eines informationstechnischen Modells abgebildet, z.B. auf ein Tabellenschema eines RDBMS, ein XML-Schema oder die Schnittstelle eines Web Service.

Der bestehende OKSTRA stellt sowohl Konzeptmodelle (in Form von NIAM-Diagrammen und freitextlichen Beschreibungen) als auch Spezifikationsmodelle bereit (als EXPRESS-, XML- und SQLDDL-Schemata).

1.2 Der objektorientierte OKSTRA

Der OKSTRA als Ontologie, d.h. als Beschreibung von Konzepten und Beziehungen, ist selbst nicht an eine bestimmte Modellierungstechnik gebunden. Bei der Umsetzung einer Ontologie in formale, standardisierte Spezifikationen, mit deren Hilfe praktisch einsetzbare, funktionierende Informationssysteme aufgebaut werden sollen, spielt die modelltechnische Sichtweise jedoch eine gewichtige Rolle.

Die sozusagen klassische Sichtweise stammt aus der Dateitechnik und, als Weiterentwicklung davon, der relationalen Datenbanktechnik. Sie ist jahrzehntelang überaus erfolgreich gewesen und spielt auch heute noch eine wichtige Rolle. Die Konzepte einer Ontologie werden in dieser Sichtweise auf die Beschreibung von Datensätzen und deren Teilen, den Datenfeldern, abgebildet. Datensätze repräsentieren identifizierbare Objekte, die Datenfelder deren Eigenschaften, entweder in Form von Attributwerten (Zahlen, Text usw.) oder von Referenzen auf andere Datensätze in Form sog. Schlüssel. Der aktuelle Bestand an Objekten in einer realisierten Datenbank wird zusammengefasst als Tabelle betrachtet und bezeichnet, die Struktur der Tabellen und ihrer Referenzen als Datenbankschema.

Das relationale Modell formalisiert den Umgang mit den Daten durch die Einführung (mengen-)mathematischer Operationen zwischen Tabellen. Die formale Sprache SQL dient zur Notation solcher Operationen. Durch die Tabellendarstellung treten bei der Umsetzung einer Ontologie jedoch notwendig Artefakte auf, denen keine Konzepte der zu Grunde liegenden Ontologie entsprechen.

> Die für die Abbildung von m:n-Relationen nötigen Zwischentabellen sind das bekannteste Beispiel für ein solches Artefakt.

Um solche Probleme zu umgehen, haben sich Modellierungstechniken etabliert, die näher an den ontologischen Beschreibungen liegen. Die bekanntesten sind das *Entity-Relationship*-Modell (ER-Modell) sowie die *Natural Language Information Analysis Method* (NIAM, auch unter dem Namen Objekt-Rollen-Modell (ORM) bekannt), die auch für den OKSTRA verwendet wird.

Die objektorientierte Sichtweise übersetzt die Konzepte und Beziehungen einer Ontologie in Objekte (präziser: *Objektklassen*) und *Operationen*, die mit den Objekten einer Objektklasse durchgeführt werden können. Eine Operation ist ein Schema für Nachrichten, auf die ein Objekt mit einer bestimmten Dienstleistung reagiert.

> In einer Implementierung eines objektorientierten Modells enthalten die Objekte daher Programme, die strukturierte Datenblöcke (entsprechend den Nachrichten) über einen Kommunikationskanal (z.B. Netzwerk) lesen und daraufhin innere Datenspeicher verändern und Nachrichten an andere Objekte übertragen.

1.2.1 Daten- und dienstebasierte Systeme

Der OKSTRA leistet in seiner jetzigen Form zweierlei:

- Er schafft durch die Konzeptmodellierung eine *standardisierte Objektbedeutung.* Seine Objektdefinitionen legen ein präzises und einheitliches Verständnis für auszutauschende Daten fest.
- Seine Spezifikationsmodelle beschreiben *offene, standardisierte Formate* für den Transport der Objektdaten.

Beide Leistungen ermöglichen eine erhebliche Verbesserung der Kommunikation zwischen Geschäftsprozessen, sowohl vertikal (entlang der Prozesskette) als auch horizontal (zwischen zwei Abläufen desselben Prozesses bei verschiedenen Bearbeitern), denn

- Es fallen auf Grund des standardisierten Formates keine unnötigen Konversionsvorgänge an.
- Es ist auf Grund der standardisierten Bedeutung sichergestellt, dass alle benötigten Daten auch tatsächlich vefügbar sind.

Dies löst aber nicht alle Austauschprobleme optimal. Die Hauptprobleme, die bei einer Kommunikation durch Datenaustausch entstehen, sind:

- **Aktualität**: Es kann nicht sichergestellt werden, dass die Daten zum Zeitpunkt ihrer Verwendung den aktuell bekannten Stand repräsentieren, sondern nur den zum Zeitpunkt der Erstellung des Austauschmediums.
- **Konsistenz**: Es kann nicht sichergestellt werden, dass ein und dasselbe Objekt bei allen Nutzern den gleichen Inhalt hat, da unterschiedliche und sogar widersprüchliche Bearbeitungsvorgänge auf verschiedenen Kopien abgelaufen sein können.

- **Redundanz**: Es müssen auf Grund der Schemaregeln u.U. viel mehr Daten ausgetauscht werden als erwünscht. Dies ist ein Problem vor allem bei langen Prozessketten, in denen jeder Prozess nur einen kleinen Teil der Daten bearbeitet, aber jeder einen anderen.

Durch objektorientierte Verfahren können diese Probleme weitgehend beseitigt werden.

Die Unterschiede von bisherigem und neuem Vorgehen und das Operationskonzept sollen an einem Beispiel erläutert werden.

> Für die Ermittlung der Grunderwerbskosten bei einem Straßenneubauprojekt werden u.a. die Erwerbsflächen zu allen vom Planungskorridor betroffenen Flurstücken benötigt. Mit der katasterführenden Stelle wird die Lieferung der Flurstücksdaten im betroffenen Gebiet vereinbart. Dabei müssen eine Reihe von Punkten abgesprochen werden, wie z.B. Inhalt und Bedeutung der Daten, das Format der Austauschdatei, das Referenz- und Koordinatensystem der Geometrie usw. Nachdem die Datei geliefert wurde, muss sie zuerst durch einen Konverter z.B. aus EDBS in eine Datei mit OKSTRA-Objekten übersetzt werden. Diese wird dann in ein OKSTRA-konformes Entwurfssystem eingelesen, das die benötigten Flächen durch Verschnitt des Korridors mit den Flurstücksflächen bildet und dabei von den betroffenen Flächen die Größen ausrechnet. Die Erwerbsflächen (modelliert im Fachbereich Grunderwerb) werden dann wieder als OKSTRA-Daten exportiert und dem Grunderwerbsprogramm übergeben, das dann endlich die Kosten ausrechnen kann.
>
> In einer objektorientierten verteilten Systemarchitektur spielt sich das Ganze wie folgt ab: Das Grunderwerbsprogramm (der Client) befragt einen Server, der Baumaßnahmen verwaltet, nach der Maßnahme, für die die Kosten zu berechnen sind. Der Server prüft die Berechtigung zum Zugriff auf das entsprechende Maßnahmenobjekt. Bei positivem Ausgang wird dieser gewährt. Der Client lässt sich von der Maßnahme das zugehörige Entwurfssystemobjekt nennen und befragt dieses nach der Korridorgeometrie. Danach wendet sich der Client mit dieser Geometrie an einen weiteren Server, der Zugriff auf die Geobasisdaten der Landesvermessung hat. Dieser Server führt damit eine geometrische Suchanfrage nach Flurstücken aus und stellt dem Client eine Menge von Flurstücks-Objekten zur Verfügung. Flurstücks-Objekte stellen eine Operation zur Verfügung, die zu einer gegebenen Geometrie die aus dem Flurstück herausgeschnittenen Geometrien liefert. Daraus werden dann Erwerbsflächenobjekte hergestellt, aus denen dann der Grunderwerbs-Client die Größe und nachfolgend die Kosten berechnen kann.
>
> Im Vergleich ergibt sich: Bei der herkömmlichen Methode per Datentransport müssen nacheinander drei Systeme aktiviert werden: Konverter, Entwurfssystem und Grunderwerbsprogramm. Der Fachbenutzer muss selbst auf den Verbleib der Dateien achten und entweder alle drei Systeme selbst bedienen können oder Kollegen mit den entsprechenden Kenntnissen bitten, die benötigten Daten zu erzeugen.
>
> Das „verteilte“ Szenario zeigt demgegenüber folgende Vorteile: Der Fachbenutzer hat es nur mit einer Applikation zu tun, nämlich seinem Grunderwerbsprogramm. Er braucht sich nicht um Organisationsfragen zu kümmern wie den Dateiverbleib, und er kann mit Ergebnissen rechnen, die dem aktuellen Stand genau entsprechen. Außerdem sind physisch weniger Daten übertragen worden, und es ergibt sich insgesamt eine erhebliche Beschleunigung des Geschäftsablaufs.

An dem Beispiel wird ein grundsätzlicher Paradigmenwechsel deutlich: Ausgetauscht werden nicht mehr die Daten eines Objekts, sondern die Objekte beantworten gezielt Fragen bzw. reagieren auf Anforderungen. Der Geschäftsablauf wird nicht mehr durch den Datenlieferanten bestimmt, sondern durch den Nutzer. Es erfolgt also ein Wandel von einem *datenbasierten* zu einem *dienstebasierten* Verarbeitungskonzept. Charakteristisch dafür ist, dass ein Objekt von einem anderen nur wissen muss, wofür dieses zuständig ist, nicht aber, wie es seine Aufgaben erfüllt.

Hierzu noch einige Anmerkungen: Der Übergang von datenbasierten zu dienstebasierten Systemen ist keineswegs etwas Neues. Historisch waren die ersten Dienste, die zentral bereit gestellt wurden, Dateidienste (Suchen, Öffnen usw. von Dateien), die entsprechenden Einheiten waren die Fileserver. Mit dem Aufkommen von Datenbanken wurden die zentralen Funktionen einer Datenbank: Einfügen, Ändern, Löschen und Abfragen von speziellen Datenbankservern als Dienst erledigt. Webserver schließlich bieten als Dienst die Bereitstellung von Dokumenten in einem standardisierten Format (HTML) an.

Allen diesen Beispielen gemeinsam ist, dass die angebotenen Dienste unspezifisch für Anwendungsbereiche sind – das Öffnen einer Datei ist keine typische Aufgabe des Straßenwesens. Anwendungsbezogen und damit ein Dienst im Sinne dieser Erörterung wäre dagegen z.B. ein Server, der die Kostenberechnung nach der AKS (Anweisung zur Kostenberechnung von Straßenbaumaßnahmen) erledigt. Nun gibt es bekanntlich eine ganze Reihe von Anwendungsprogrammen für solche fachlichen Aufgaben, und diese werden in lokalen Netzwerken zusammen mit den soeben genannten Infrastrukturdiensten eingesetzt. Solche Architekturen bleiben dennoch datenbasiert, denn es wird ja lediglich der Transport von Daten über ein Netzwerk statt über Datenträger abgewickelt, die Software für die Anwendung muss auf jedem Arbeitsplatz installiert werden, an dem die Aufgabe anfällt, und die Übermittlung der Daten von einer Anwendung zur nächsten geschieht weiterhin über die Aktivierung entsprechender Import- oder Exportfunktionen.

Die wichtigsten Vorteile von anwendungsorientierten, dienstebasierten Architekturen sind:

- ***Wirtschaftlichkeit***: Die Total Cost of Ownership (TCO), also die Gesamtkosten für Aufbau und Unterhalt eines Informationssystems, können durch Nutzung vielfältiger, dezentraler Dienste, von denen jeder *nur* die Daten und Programme pflegen muss, für die er selbst zuständig und verantwortlich ist, deutlich gesenkt werden.

  So müssen z.B. Katasterdaten bei Fortschreibungen nicht mehr durch die nutzende Stelle nachgepflegt werden, wenn sie jederzeit aktuell bei der Katasterverwaltung online abrufbar sind.

- ***Aktualität***: Jedes reale Objekt des bestehenden und geplanten Straßennetzes wird bei der dafür verantwortlichen Stelle eindeutig und vollständig repräsentiert und in seinem aktuell gültigen Zustand zur Verfügung gestellt. Aktualitätsprobleme durch mögliche unkoordinierte Pflege verschiedener Kopien einer Objektrepräsentation entfallen, ebenso wie Probleme durch die Existenz von mehreren Objektrepräsentationen, die widersprüchlich sind, weil sie nichts voneinander wissen. Änderungsvorgänge an einem Objekt können automatisch an alle Anwendungen gemeldet werden, die es verwenden.

- ***Sicherheit***: Die Daten eines Objektes können leicht vor unkontrolliertem und unberechtigtem Zugriff geschützt werden, da alle Zugriffe sich in Form von Operationen abspielen, die Zugriffsberechtigungen prüfen können. Die Operationen sorgen für Konsistenz (Informationen zu verschiedenen Themen sind miteinander verträglich) und Konformität (Informationen werden nach den Regeln der für sie verbindlichen Vorschriften erzeugt und weitergegeben).

  Wie bei allen technischen Systemen, so ist auch bei IT-Systemen, die Informationsdienste anbieten, die Qualität der angebotenen Leistung abhängig von den eingesetzten Ressourcen und ihrer Qualität. Je nach Wichtigkeit und Häufigkeit der Nutzung eines Dienstes sind unterschiedliche Anforderungen an Sicherheit, Verfügbarkeit, Aktualität der angebotenen Information und Antwortzeitverhalten gegeben, deren Einhaltung mit dem Betreiber der Dienste vertraglich zu vereinbaren ist.

Zur Realisierung der genannten Vorteile tragen entscheidend neuere Trends der Informationstechnologie bei, sowohl im Hinblick auf die Architektur der Anwendungen als auch auf die Art und Weise des Informationsaustausches zwischen ihnen. Kennzeichnend dafür sind:

- ***Komponenten***: Anwendungen werden nach dem Legoprinzip durch Zusammenschalten von Komponenten realisiert, die jeweils für einen ganz bestimmten, klar umrissenen Aufgabenbereich verantwortlich sind und von vielen unterschiedlichen Anwendungen genutzt werden können.
- ***Verteilte Systeme***: Die für eine Anwendung benötigten Komponenten müssen nicht alle auf demselben Computer installiert sein. Komponenten, die auf unterschiedlichen Systemen liegen, kommunizieren miteinander über Netzwerkverbindungen mithilfe standardisierter Protokolle, z.B. in Intranets.
- ***Internet-Technologie***: Die Benutzer verwenden am Arbeitsplatz zunehmend nicht mehr komplexe aufgabenspezifische Anwendungsprogramme, sondern sie arbeiten mit Informationsseiten innerhalb ihres Internet-Browsers, die über das Netz anwendungsabhängig bereitgestellt werden. Die Anwendungslogik verlagert sich vom Arbeitsplatz weg auf serverbasierte Dienste.
- ***Sicherheits-Technologie:*** Die Nutzung von Informationssystemen ist auf Grund administrativer, wirtschaftlicher und juristischer Vorgaben auf bestimmte Nutzerkreise, bestimmte Zugriffsarten und bestimmte Informationselemente zu beschränken. Informationen werden verschlüsselt übertragen, der Zugang zu Informationssystemen kann durch PKIs (Public Key Infrastructures) zentral und einheitlich verwaltungsübergreifend geregelt werden. Damit wird die kombinierte Nutzung von Informationsdiensten verschiedener öffentlicher und privatwirtschaftlicher Betreiber möglich.

1.2.2 Statische und objektorientierte Modelle

Die „statische“ Betrachtungsweise, auf der der bestehende OKSTRA basiert, bildet die reale Welt des Straßen- und Verkehrswesens auf eine Konzeptwelt von abstrakten Objekten ab. Die für das Fachgebiet wichtigen und charakteristischen Merkmale der realen Gegenstände übersetzen sich dabei in Objekt-*Attribute*, und die gegenseitigen Beziehungen in Objekt-*Relationen*.

> Die Verkehrszeichen der verkehrsregelnden Beschilderung haben eine *Bedeutung für den Verkehrsteilnehmer*, sowie eine *Lagebeziehung* zu einer Straße. Die Schemata des Bereichs 016 des OKSTRA präzisieren diese Konzepte in Form von Attributen und Relationen.

Attribute und Relationen sind in dieser Betrachtungsweise *Zustände* des Objekts, die nur von außen geändert werden können, z.B. wenn ein Schild ein paar Meter versetzt wird. Die „statischen“ Modelle sind also objektbasiert, betrachten die Objekte jedoch als passive Informationsträger.

Die *objektorientierte* Betrachtungsweise übernimmt zunächst den Objektbegriff der statischen Betrachtungsweise, erweitert ihn jedoch dahingehend, dass ein Objekt aktive, ausführbare Bestandteile (in Form von Programmcode) enthalten kann, die für das Objekt charakteristische Aufgaben lösen können. Der in diesem Leitfaden gewählte Zugang zur Objektorientierung betont die konzeptionelle Seite der Modellierung und geht von folgenden Grundsätzen aus:

- Jedes Objekt übernimmt klar definierte und begrenzte *Zuständigkeiten* oder *Verantwortlichkeiten*: Dem Objekt werden ganz bestimmte Aufgaben zugeordnet, die es erledigen soll.

  > Die *Abschnitte* des Netzknotenstationierungssystems sind (u.A.) für den Lagenachweis der Verbindungen zwischen Nullpunkten zuständig.

Wesentlich für ein objektorientiertes Modell ist, dass *alle* Aufgaben, die ein nach dem Modell gebautes System leisten soll, Objekten zugeordnet werden.

- Objekte kooperieren miteinander durch den *Austausch von Nachrichten.* Falls Objekte zur Lösung einer Aufgabe Teilaufgaben erledigen müssen, die nicht in ihre eigene Zuständigkeit fallen, *beauftragen* sie hierzu stattdessen die Objekte, die das können *(Delegationsprinzip)*. Dazu muss aber ein Objekt anderen Objekten Nachrichten mit den Einzelheiten solcher Aufträge zusenden können.

  Möchte ich den Gesamtverlauf einer Straße von einem Straßenobjekt wissen, so wird es der Reihe nach alle seine Abschnitte beauftragen, ihre Lage mitzuteilen, und dann wird es die Gesamtgeometrie daraus zusammensetzen.

- Um eine Nachricht senden zu können, muss die Existenz des Empfängers gesichert sein. Nötigenfalls ist der Empfänger also zunächst zu *erzeugen*. Der Empfang einer Nachricht kann den *Zustand des Empfängers* beeinflussen.

  Soll ein Abschnitt verlängert werden, so sendet man ihm eine Nachricht, die die Daten dazu (Geometrie usw.) enthält. Der Abschnitt wird daraufhin seinen Verlauf entsprechend verändern.

Die soeben ausgesprochenen Grundsätze sind ganz allgemeiner Natur und orientieren sich nicht an informationstechnischen Gesichtspunkten. Ausgehend davon wird eine konzeptionelle, objektorientierte Modellierung folgende Fragen zu beantworten haben:

- Was gibt es für Objekte?
- Welche Aufgaben gibt es?
- Wie sind die Aufgaben den Objekten zu zu ordnen?
- Welche Objekte kooperieren miteinander und welche Informationen sind dabei zu übertragen?

Für eine informationstechnische Umsetzung ergeben sich folgende Erfordernisse:

- Alle Objekte haben eine unverwechselbare und unveränderbare *Identität*, über die sie zum Übermitteln von Nachrichten adressiert werden. Sie wird durch den Empfang von Nachrichten nie geändert, und sie hat keine fachliche Bedeutung.

  Im bisherigen OKSTRA ist die Sicherstellung einer eindeutigen Objektidentität nach wie vor ein ungelöstes Problem.

- Objekte werden irgendwann *erzeugt*, sie besitzen innere Zustände, die sich ändern können, und sie können irgendwann *vernichtet* werden. Objekte haben also einen *Lebenslauf*. Die Art und Weise, *wie* ein Objekt seine Zustandsdaten speichert, wird nicht vorgegeben und ist grundsätzlich als irrelevant zu betrachten.

- Objekte bieten *Dienste* in Form von *Schnittstellen an*. Eine Schnittstelle definiert für das damit ausgestattete Objekt ein oder mehrere *Operationen***,** durch deren Verwendung die Nachrichten ausgetauscht werden**.** Ein Objekt ist von außen nur über Operationen zugänglich.

  Was diese Forderung beinhaltet, demonstrieren wir an einem Baumkataster. Dieses möge für Baumobjekte eine Operation *SetzeBaumart* anbieten. Die Baumart eines bestimmten Baumes darf dann nur durch Verwendung dieser Operation (ausgedrückt in der benutzten Programmiersprache) geändert werden. Das Systemkonzept für das Baumkataster hat sicherzustellen, dass andere Möglichkeiten ausgeschlossen sind. Die *Implementation* der Operation ist nicht Bestandteil des Modells und kann be-

liebig sein, z.B. als UPDATE in einer relationalen Datenbank oder als e-Mail an eine Mitarbeiterin, die die Baumkartei per Hand führt.

Eine Schnittstelle wird oft auch als *Vertrag* angesehen, der ein bestimmtes Verhalten zusichert.

Vertragsgemäßes Verhalten würde im Baumkataster-Beispiel z.B. bedeuten, dass eine Anfrage mit Hilfe einer *GibBaumart*-Operation die Baumart ausweist, die durch die letzte *SetzeBaumart*-Operation für den gewählten Baum eingetragen wurde.

Schnittstellen müssen zum einen *formal* spezifiziert werden, damit die Nutzer wissen, wie sie die Dienste eines Objektes ansprechen können, zum anderen muss es eine ausreichend detaillierte Beschreibung der fachlichen Leistung der Dienste geben.

Wie aus den Ausführungen dieses Abschnitts hervorgeht, sind objektorientierte Modelle besonders dazu geeignet, Aufgabenteilungen und Kommunikationsprozesse abzubilden. Sie können daher nicht nur Modelle für den Datenaustausch zwischen den Geschäftsprozessen des Straßen- und Verkehrswesens ausdrücken, sondern auch die Informationsverarbeitung in den Arbeitsschritten dieser Prozesse selbst.

Z.B. kann für eine Kostenberechnung nicht nur die Struktur der notwendigen Eingangsdaten (Mengenangaben, Einheitspreise) und Ergebnisse modelliert werden, sondern der gesamte Arbeitsablauf bis zu einer informationstechnisch umsetzbaren Form, die eine „Kostenberechnung auf Knopfdruck“ mit automatischer Aktualisierung bei Änderungen der Planung oder des Preisgefüges ermöglicht.

1.3 Zielsetzung des Leitfadens

Die objektorientierten Modelle des OKSTRA:

- dokumentieren die Geschäftsprozesse des Straßen- und Verkehrswesens in folgenden Aspekten:
  - Startbedingungen, z.B. das Vorhandensein der Planfeststellungsunterlagen,
  - Ablauf als Abfolge von durchzuführenden Arbeitsschritten,
  - Endprodukte, z.B. erstellte Pläne oder planfestgestellte Straße,
  - Kommunikation und Synchronisation mit anderen Geschäftsprozessen, z.B. Warten auf Bereitstellung von Mengenangaben für eine Kostenberechnung,
- beschreiben die in den Geschäftsprozessen benötigten und anfallenden Informationen in Form von standardisierten Objekten, z.B.
  - physisch realisierte Dinge (z.B. den Schichtaufbau einer Straße oder einen Baum),
  - geometrische und topologische Abstraktionen (z.B. Achsen, Abschnitte),
  - Dokumente (z.B. Ausschreibungsunterlagen),
  - Messergebnisse (z.B. Spurrinnentiefen)
  - Vorkommnisse (z.B. Unfälle)
  - Prozesse (z.B. Baumaßnahmen)
- beschreiben für jedes Objekt die Operationen, durch deren Verwendung bestimmte Aufgaben innerhalb der Prozesse abgewickelt werden können,

- stellen eine formale, automatisch durch informationstechnische Systeme (kurz IT-Systeme) nutzbare, standardisierte Darstellung für jedes Objekt bereit. Damit kann sowohl das Zusammenwirken existierender IT-Anwendungen im Hinblick auf die Geschäftsprozesse optimiert werden und ebenso können neue Anwendungen geschäftsprozessorientiert entworfen und realisiert werden.

1.4 Aufbau des Leitfadens

Kapitel 2 beschreibt unter dem Titel *Die objektorientierte Modellierung des OKSTRA* das Verfahren, aus einer Analyse der Geschäftsprozesse des Straßen- und Verkehrswesens ein für praktische Aufgaben brauchbares Modell einer dienstebasierten Informationsarchitektur zu entwickeln.

Kapitel 3 enthält *Definitionen und Erläuterungen zur Objektorientierung***,** um für den OKSTRA eine einheitliche und verbindliche technische Terminologie zu schaffen. Die Notwendigkeit ergibt sich daraus, dass in der wachsenden Flut an Publikationen - Lehrbücher, Aufsätze und Produktdokumentationen – zum Thema Objektorientierung die Zuordnung von Konzepten zu ihren Benennungen nicht einheitlich ist, was leicht zu Missverständnissen führen kann.

Kapitel 4 ist ein Überblick zur *Bildung von Klassen* und zur Auffindung von Operationen. Objektorientierte Modelle stellen alle Sachverhalte unter Bildung von Objektklassen dar, also nicht nur das Aussehen und die Verknüpfungen dauerhafter, anfassbarer Objekte, sondern auch Vorgänge, Verfahren usw. Andererseits fehlt in der technischen Literatur zumeist der Hinweis auf Klassenbildungsregeln einprägsamer Natur. Die Bildung der Objektwelt ist ein intuitiver und daher schwer präzisierbarer Prozess, bei dem es außer auf die korrekte Abbildung der Realität auch auf Faktoren wie Begrenzung von Komplexität, Verständlichkeit und Umsetzbarkeit ankommt. Das Kapitel gibt praktische Erfahrungen weiter.

In Kapitel 5 wird die *Benutzung der Modellierungssprache UML* (Unified Modeling Language) erläutert. Es wird die Anwendbarkeit verschiedener Diagrammtypen erklärt sowie der Einsatz bestimmter Modellierungselemente präzisiert.

In Kapitel 6 folgen Erläuterungen und Empfehlungen zur *Dokumentation der OKSTRA-Modelle*. Dies betrifft die Detaillierung der Modelle und die Bearbeitung der Modelle für den praktischen Gebrauch.

Kapitel 7 ist ein Katalog von *Rahmenmodellen*, die für den Aufbau des objektorientierten OKSTRA notwendig sind. Es werden in Form einer Anforderungsliste Problemfelder aufgezeigt, die unabhängig von den fachlichen Modellen zu bearbeiten sind.

Kapitel 8 beschreibt die Aufgabe „Modellierung des objektorientierten OKSTRA“ nochmal insgesamt unter Berücksichtigung nötiger Vorarbeiten.

Die Kenntnis des bestehenden OKSTRA ist eine notwendige Voraussetzung für das Verständnis des Textes, da sich der objektorientierte OKSTRA nicht als Ersatz, sondern als Weiterführung des existierenden OKSTRA-Regelwerkes versteht.

Für eine erfolgreiche Anwendung des Leitfadens ist die Kenntnis der Modellierungssprache UML (*Unified Modeling Language*) unabdingbar. Das Kapitel 5 erläutert zwar die wichtigsten Aspekte dieser Modellierungssprache, es ist jedoch nicht als umfassende Referenz oder gar als Lehrbuch konzipiert. Dies ist auch nicht notwendig, da es mittlerweile an die 100 deutschsprachige Titel zum Thema UML gibt, mit steigender Tendenz.

Lange Zeit richtete sich die zu UML verfügbare Literatur an Softwaredesigner und nicht an Mitwirkende bei konzeptionellen Modellierungen. Angeregt durch die neuere Version UML 2.0, die ein wesentlich stärkeres Gewicht auf die Aufgaben der Geschäftsprozessmodellierung legt, ist jedoch mit

> Bernd Oestereich:
> Objektorientierte Softwareentwicklung
> Analyse und Design mit der UML 2.0
>
> 6. völlig überarbeitete Auflage, Oldenbourg Wissenschaftsverlag, München 2004, ISBN 3-486-27266-7

ein ausgezeichnetes Buch verfügbar, das Verständlichkeit und Präzision vereinigt und ein Studium auf unterschiedlichem Vertiefungsniveau ermöglicht. Es wird daher in diesem Leitfaden als Standard-Referenzwerk eingesetzt..

Der Leitfaden ist in einigen Teilen nicht als abgeschlossen anzusehen. Vielmehr ist er regelmäßig fortzuschreiben, um die technische Weiterentwicklung und die Erkenntnisse aus der Modellierungstätigkeit zu berücksichtigen. Hierzu gibt das Kapitel 8 weitere Hinweise.

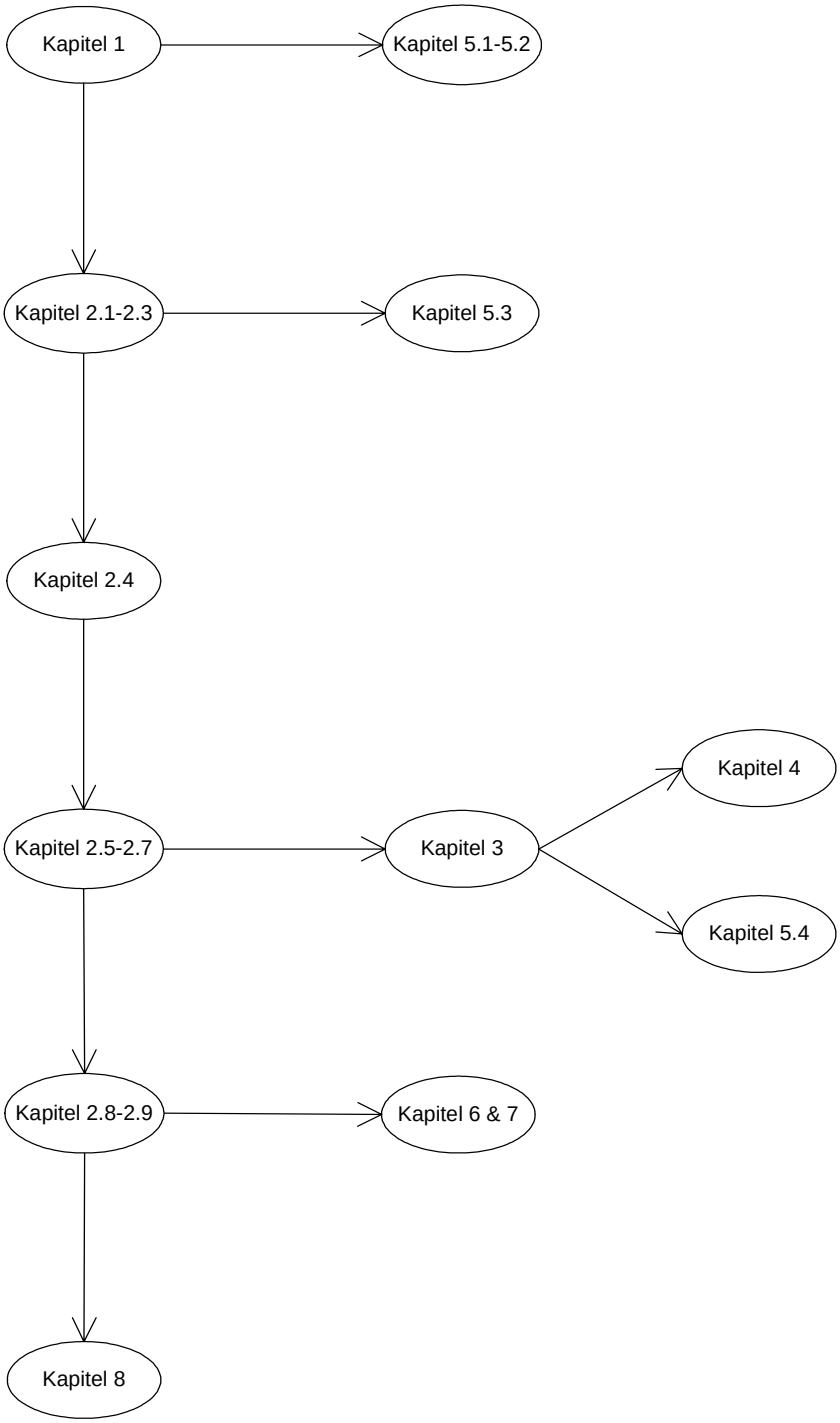


Abbildung 1. Lektüreplan.

1.5 Glossar

Das Glossar fasst alle wichtigen Begriffe des Leitfadens zur Modellierung in Kurzform zusammen. Jeder Eintrag enthält in der ersten Zeile den definierten Begriff, einen Verweis auf das Kapitel oder den Abschnitt, in er behandelt wird sowie eine Liste verwandter, ähnlicher oder ergänzender Begriffe; darunter folgt dann die Definition. Das Glossar stützt sich auf

- das Glossar aus dem oben empfohlenen Buch von B. Oesterreich [Oes 04]
- die Glossare der neuen UML-Spezifikationen [UML-I 04, UML-S 04].

Die Definitionen wurden für den Gebrauch in diesem Leitfaden präzisiert bzw. angepasst. Insbesondere wurden die andernorts üblichen Begriffe *Problembereich* bzw. *Domäne* auf den in [GP-Neu 02] eingeführten Begriff des *Geschäftsbereichs* bezogen, der sich ganz konkret auf das Vorhandensein einer Datenbasis bezieht. [Oes 04] definiert den Problembereich allgemein als „Anwendungsgebiet, innerhalb dessen die fachliche Modellierung stattfindet“. Ebenso wurde der Begriff *Akteur* mit konkretem Bezug auf Geschäftsbereiche definiert.

abgeleitete Klasse 3.9 Ableitung, Unterklasse, Subtyp
Eine a.K. hat eine oder mehrere Basisklassen

Aggregat 7 Kollektion
Eine Sammlung von Datenelementen.

Akteur 4.4 Rolle
Klassifizierer für Objekte, die die Datenbasis des untersuchten Geschäftsbereichs pflegen oder nutzen

Aktivität 2.3.1 Subprozess
Arbeitsschritt innerhalb eines Geschäftsprozesses

Akzessor 3.8
Operation zum Zugriff auf ein Attribut oder verknüpfte Objekte

Anwendungsfall 2.3.5
A. beschreiben die Interaktionen und Tätigkeiten von Akteuren in Geschäftsprozessen

Assoziation 3.8 Relation
Klassifizierer, dessen Exemplare die Zuordnung von Exemplaren von gegebenen Klassen ausdrücken

Attribut 3.8 Merkmal, Variable
Eigenschaft einer Klasse, die sich durch Annahme von Werten ausdrückt

Basisklasse 3.9 Oberklasse, Supertyp
Ergebnis einer Generalisierung von Klassen

Bestand
Die Gesamtheit der existierenden Exemplare einer Klasse

Datenbasis 2.2.5
Speicher zur dauerhaften Aufbewahrung von zusammengehörenden Datenelementen.

Datenelement
Ein Objekt, ein Aggregat oder ein Wert eines Datentyps.

Datentyp 3.7
Klassifizierer, dessen Exemplare schon und nur durch ihren Wert auseinandergehalten werden und die unbegrenzt existieren

Enumeration 7 Aufzählung
Ein Datentyp, dessen Exemplare einer Liste benannter Elemente ohne weitere Eigenschaften entstammen

Ereignis 2.1.4
Zeitlich lokalisiertes Vorkommnis, das den Ablauf eines Geschäftsprozesses beeinflusst

Exemplar 3.3 Instanz, Objekt
Ein Informationsträger mit Identität und Lebenslauf.

Folge (Aggregat) 7 Liste
Ein Aggregat, das eine Reihung seiner Elemente aufweist

Generalisierung 3.9 Abstraktion
Die Bildung einer Basisklasse zu einer oder mehreren Klassen durch Auswahl gemeinsamer Eigenschaften

Geschäftsbereich 2.1.4
Sammlung von Geschäftsprozessen, die eine gemeinsame Datenbasis pflegen

Geschäftsprozess 2.1.4
Eine Menge logisch verknüpfter Arbeitsschritte, die durchgeführt werden, um ein bestimmtes Geschäftsziel zu erreichen.

Geschäftsvorfall 2.1.4
Ablauf eines Geschäftsprozesses für eine aktuelle Situation..

Instanz 3.3 Exemplar
Realisierung eines Schemas

Instanzierung 3.3
Erzeugung einer Instanz nach dem zugehörigen Schema

Klasse 3.4
Klassifizierer, der durch eine Sammlung von Operationen definiert ist

Klassifizierer classifier
Eine Zusammenfassung von Exemplaren nach Gemeinsamkeiten

Konstrukt
Gedankliche Konstruktion, von daher auch: Sprachelement. In Abgrenzung zu Konzept und Objekt verwendet

Konzept 2.1.1 Fachkonzept
Für einen Geschäftsbereich kennzeichnende Begriffsbildung

Menge (Aggregat) 7
Eine Aggregat ohne weitere Struktur

Metaklasse 3.10
Eine Klasse, deren Exemplare Klassen sind

Methode 3.2
Implementation oder Implementationsvorschrift für eine Operation

Modell 2.1.1
Übersetzung von Begriffen und Sachverhalten eines Geschaftsbereichs in eine formale

Notation

Modellelement 2.1.1
Instanz eines Modellkonstruktes innerhalb eines Modells

Modellkonstrukt 2.1.1 Modellelementtyp
Konstrukt, das zum Aufbau eines Modells dient

Mutator 3.8
Operation zum Verändern eines Attributes

Nachricht 3.2 message
Information, die ein Exemplar an ein anderes Exemplar überträgt

Objekt 3.2
Ein Exemplar; verallgemeinert auch das Schema für eine Klasse von Exemplaren

Observator 3.8
Operation zum Erfragen eines Attributes

Ontologie 1.1
Sammlung von Konzepten und deren Beziehungen für einen Geschäftsbereich

Operation 3.2 Nachricht, Botschaft
Modellkonstrukt für die dynamische Funktion (das Verhalten) von Exemplaren

Parameter 3.2
Datenelement zur Steuerung einer Operation oder zur Aufnahme ihrer Ergebnisse.

Schema Klassifizierer
Ein Organisationsmuster für Informationsinhalte oder Abläufe

Schnittstelle 3.11 Interface
svw. Typ, insbesondere, wenn nur Operationen definiert sind

Signal 2.1.4 notification
Durch ein Ereignis ausgelöste Nachricht

Spezialisierung 3.9
Die Bildung einer Ableitung aus einer Klasse durch Hinzufügen von Eigenschaften

Szenario 2.3.5
Drehbuchartige Beschreibung eines Anwendungsfalles

Tabelle (Aggregat) 7
Aggregat, das den Zugriff auf seine Elemente über eindeutige Schlüssel zulässt

Typ 3.11
Klasse, die keine Methoden definiert.

Wert 3.7
Ein unveränderliches Exemplar unbegrenzter Lebensdauer

Workflow 7
Ein (teil-)automatisierter Geschäftsprozess

1.6 Typebene und Instanzebene

Die Konstrukte der Prozess- und Objektmodellierung lassen sich der Typebene oder der Instanzebene zuordnen. Die Konstrukte der Typebene beschreiben die Struktur bzw. das Verhalten der Konstrukte der Instanzebene, diese sind die Träger der Inhalte im aktiven, arbeitenden System.

Die Unterscheidung zwischen den beiden Ebenen ist wichtig. Allerdings wird gerade bei der Analyse von Dokumenten und der Durchführung von Interviews diese Trennung nicht immer strikt durchgehalten, weil sie den Autoren und Gesprächspartnern nicht bewußt ist. Deshalb stellt die folgende Tabelle die zusammengehörenden Konstrukte gegenüber:

Typebene	Instanzebene
Schema	Inhalt, Ablauf
Definition	Beispiel
Geschäftsprozess	Geschäftsvorfall
Akteur, Rolle	Person, Systeminstallation
Anwendungsfall	Szenario, Story
Klasse, Typ	Exemplar, Instanz
Operation	Ausführung
Attribut	Wert
Assoziation	Relationselement (Verknüpfung), Relation
Metaklasse	Klasse

2 Die fachliche Modellierung des objektorientierten OKSTRA

Das vorliegende Kapitel beschreibt die für die Erstellung der Modelle des objektorientierten OKSTRA notwendigen Arbeitsschritte.

Die meisten Beispiele für dieses Kapitel liefert die Modellierung der Kostenermittlung, wie sie im Anhang A beschrieben ist.

Der Gesamtablauf der Modellierung ist aus der Abbildung auf der folgenden Seite ersichtlich.

2.1 Vorbemerkungen

2.1.1 Modelle

Ein *Modell* übersetzt die Begriffe und Sachverhalte des untersuchten fachlichen Bereiches in eine formale Sprache, die als linearer Text oder grafisch notiert wird. Die Elemente einer solchen Sprache (die „Wortarten“ sozusagen) werden in diesem Text als *Modellkonstrukte* oder auch als *Modellelementtypen* bezeichnet, die Elemente eines Modells (die „Wörter eines Textes“) als *Modellelemente*.

Der OKSTRA kennt zwei Arten von Modellen: *Konzeptionsmodelle und Spezifikationsmodelle*.

Konzeptionsmodelle enthalten die Ontologie, d.h. die für den untersuchten fachlichen Bereich charakteristischen Begriffsbildungen (*Konzepte*): Gegenstände, Vorgänge, Ereignisse, Dokumente usw., ihre Eigenschaften sowie ihre Beziehungen zueinander. Konzeptionsmodelle werden auch als Realweltmodelle bezeichnet.

> Wichtige Beziehungstypen sind taxonomische (Einordnung in ein Schema von Ober- und Untergruppen durch Zusammenfassung bzw. Abgrenzung), merologische (Zerlegung in Bestandteile, Bildung von Zusammensetzungen) und topologische (räumliche Beziehungen wie Berührung, Verbindung, Überdeckung).

Konzeptionsmodelle geben das Fachwissen des modellierten Bereiches wieder; daher ist es notwendig, dass sie von den Fachleuten des Bereichs gelesen und verstanden werden können. Sie entstehen oft in mehreren Durchgängen durch fortschreitende Verfeinerung und Präzisierung der im Modell ausgedrückten Begriffsbildungen.

Der bisherige OKSTRA drückt die konzeptionellen Sachverhalte in der grafisch notierten NIAM-Sprache aus, deren wichtigste Modellkonstrukte die als Ellipsen gezeichneten *Objekte* und die als Kästchen dargestellten *Rollen* sind, welche die Objekte zueinander in Beziehung setzen. Die Modelle des objektorientierten OKSTRA verwenden als wesentliche Modellkonstrukte ebenfalls Objekte. Die Modellkonstrukte für die Eigenschaften und Beziehungen der Objekte heißen hier *Operationen*.

Das Spezifikationsmodell eines Informationssystems enthält die Vereinbarungen, die für eine sinnvolle informationstechnische Nutzung des Systems einzuhalten sind, z.B. Datenformate für den Informationstransport. Es entsteht aus einem Konzeptionsmodell dadurch, dass den Konzepten eine technisch realisierbare Repräsentation zugeordnet wird. Dies kann je nach verwendeter Technik auf verschiedene Art geschehen. So enthält der bisherige OKSTRA Spezifikationsmodelle in den Sprachen EXPRESS, SQLDDL und XML-Schema.

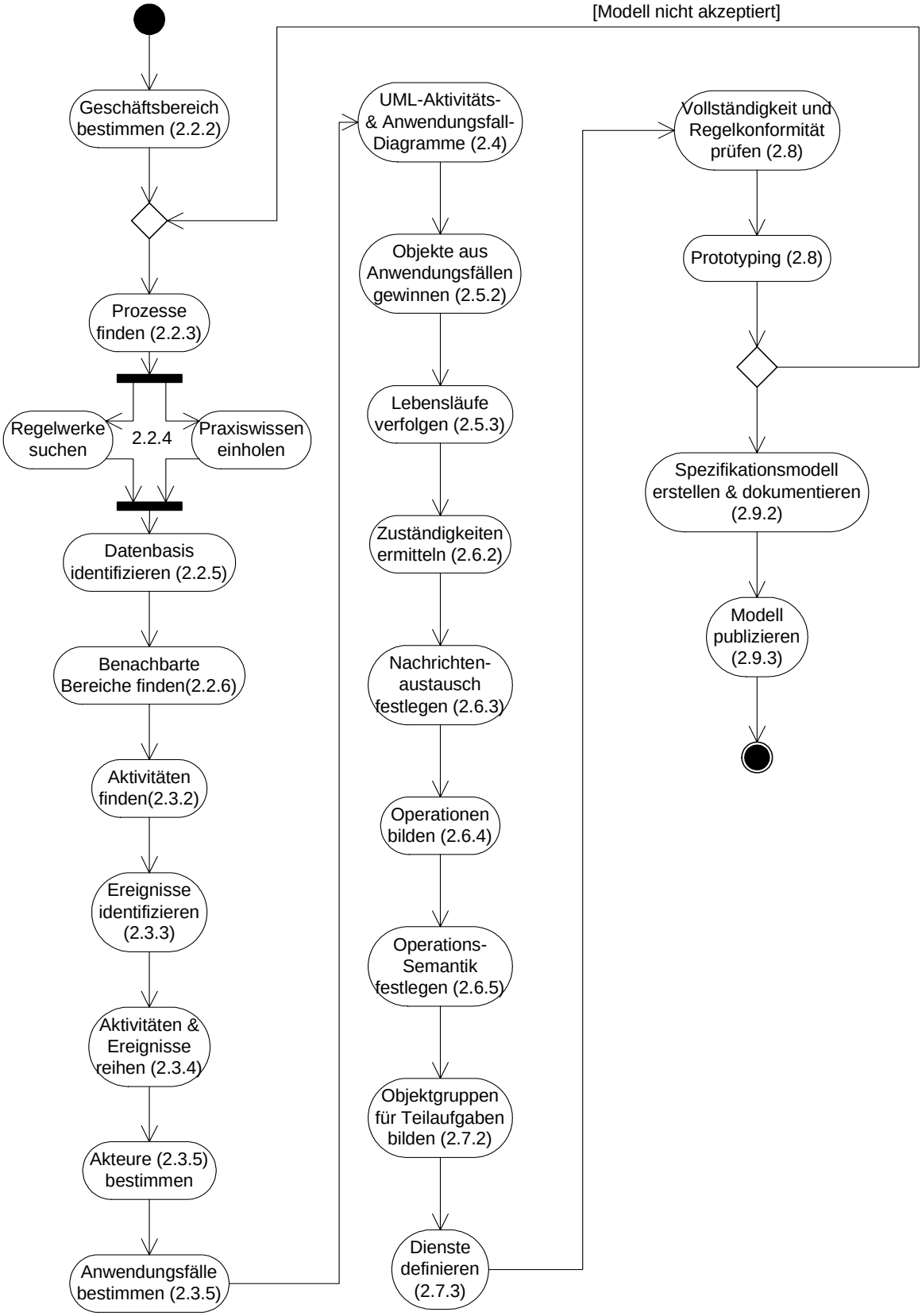


Abbildung 2. Die OKSTRA-Modellierung.

2.1.2 Begriffsdefinitionen

Um ein Modell zu schaffen, das eine eindeutige Interpretation erlaubt, müssen die Konzepte so präzise wie möglich definiert werden. Dies dient einerseits dazu, den zu modellierenden Bereich gegenüber anderen abzugrenzen, andererseits, um innerhalb des Modells Konflikte und Mehrdeutigkeiten auszuräumen.

Die Definitionen für die fachlichen Konzepte müssen in einem Glossar gesammelt dokumentiert werden. Sofern es fachlich verbindliche oder allgemein anerkannte Definitionen gibt, sind diese zu nutzen, wobei im Glossar die maßgebliche Quelle verzeichnet wird. Reichen die benutzten Quellen hierzu nicht hin, ist nötigenfalls ist eine eigene Definition vorzuschlagen und abzustimmen.

2.1.3 Modelldokumentation

Während der Modellierung sind die Ergebnisse der im Folgenden beschriebenen Modellierungsschritte zu dokumentieren:

- Als grafische Sprache für die Modelle wird UML (Unified Modeling Language) ab der Version 1.4 verwendet, und zwar mindestens in dem Umfang, wie in Kapitel 5 beschrieben.
- Die Nutzung weiterer Bestandteile von UML bleibt dabei unbenommen, sollte aber gerade in der Konzeptmodellierung mit Vorsicht und sparsam betrieben werden, denn in dieser Modellierungsphase liegt der Schwerpunkt auf Verständlichkeit und Vermittelbarkeit bei den Fachleuten.
- Für die Spezifikationsmodellierung kann und soll UML im durch die Modellierungsaufgabe geforderten Umfang eingesetzt werden, der den Rahmen von Kapitel 5 überschreiten kann. Es kommt hier ja darauf an, möglichst präzise Vorgaben für die Softwarehersteller zu liefern.
- Insbesondere für die Dokumentation der Geschäftsprozesse ist es angeraten, die Version UML 2.0 zu verwenden. Der Wechsel zur dieser Version sollte generell vollzogen werden, sobald ein verabschiedeter Standard für UML 2.0 vorliegt.
- Die Einschränkung des Sprachumfanges von UML soll die Einstiegsschwelle zum Lesen und Verstehen der Modelle für Interessenten ohne professionellen IT-Hintergrund herabsetzen, nicht aber Einschränkungen der Modelle hervorbringen.
- Die UML-Modelle sind, wo nötig, durch weiteren Text zu kommentieren und insbesondere mit Quellenangaben zu versehen.

2.1.4 Begriffe zur Prozessmodellierung

In den folgenden Abschnitten spielt bei der Beschreibung der Arbeitsschritte der Modellierung der Prozessbegriff eine zentrale Rolle. Daher sollen der Prozessbegriff, so wie er in diesem Leitfaden verwendet wird, sowie damit verbundene Begriffe in Form einiger Regeln präzisiert werden.

- Es gibt von einander abgrenzbare *Geschäftsbereiche*, die sich jeweils mit der Erfassung und Pflege von Daten eines fest umrissenen Arbeitsfeldes beschäftigen.

  Beispiele sind:

  - die geometrischen Entwurfsdaten der Planung,
  - die Kosten- und Abrechnungsdaten,

- die Inventardaten des bestehenden Straßennetzes,
- die Daten zur Dynamik des Verkehrsgeschehens,
- die Daten zur Beeinflussung der Umwelt durch Straße und Verkehr

- In jedem Geschäftsbereich gibt es *Geschäftsprozesse*, das sind Handlungsabläufe, die ein bestimmtes Ziel verfolgen, und zum Erreichen dieses Zieles die Daten des Geschäftsbereichs bearbeiten: Ein Geschäftsprozess ist „eine Menge logisch verknüpfter Arbeitsschritte, die durchgeführt werden, um ein bestimmtes Geschäftsziel zu erreichen“ (übersetzt aus einer Definition in Davenport & Short, Sloan Management Review, Sommer 1990, S. 11-27).

  Die Geschäftsprozesse des Bereichs *Kostenermittlung (Baumaßnahme)* bearbeiten die Kosten- und Abrechnungsdaten, um Bauleistungen finanzieren, vergeben und abrechnen zu können.

- Ein *Geschäftsvorfall* ist der Ablauf eines Geschäftsprozesses zu einem bestimmten Zeitpunkt und mit dann aktuellen Informationen.
- Die Handlungsabläufe der Geschäftsprozesse lassen sich in einzelne *Aktivitäten* zerlegen. Zu jeder Aktivität sind Informationen erforderlich, und sie können dem Geschäftsbereich neue oder geänderte Informationen zuführen.

  Zur Fortführung der Inventardaten des Straßennetzes gehört die Aktivität *Netztopologie pflegen*.

- Die Geschäftsprozesse eines Bereiches und solcher Bereiche, die seine Daten nutzen, beeinflussen sich als Folge von *Ereignissen*. Ereignisse treten an definierten Punkten eines Handlungsablaufs auf. Ihre Wirkung besteht im Anstoßen, Unterbrechen, Wiederaufnehmen und Beenden der Prozesse und in der Modifikation der Handlungsabläufe.

  Das Ereignis *Planung geändert* bewirkt eine *Kostenfortschreibung*.

  Das Eintreten eines Ereignisses innerhalb eines Geschäftsprozesses wird einem anderen Prozess durch ein *Signal* mitgeteilt.

- Die Informationsverarbeitung innerhalb der Prozesse wird durch *Dienste* geleistet. Dienste kommunizieren über *Schnittstellen* miteinander. Dienste können als *Komponenten* technischer Informationssysteme implementiert werden, die die Aktivitäten der Prozesse teilweise oder vollständig automatisieren.
- Dienste erfüllen ihre Verarbeitungsaufgabe durch das geordnete Zusammenwirken aktiver, dynamischer *Objekte*. Objekte tauschen untereinander *Nachrichten* aus, die über technische Kommunikationsverbindungen übermittelt werden. Die Objekte können *Zustände* einnehmen, die sich im Laufe der Zeit als Folge des Empfangs von Nachrichten ändern können. Objekte können erzeugt und vernichtet werden.

Ziel der Modellierung ist die Definition von Objekten und Diensten. Wie die Definitionen gewonnen werden können, ist Thema der folgenden Abschnitte des Kapitels.

Dokumentiert wird das Modell am zweckmäßigsten wie folgt:

Zu Beginn des Modellierungsprojektes wird ein Dokument angelegt, das mit dem Namen des untersuchten Geschäftsbereiches betitelt wird. Das Dokument wird in Kapitel aufgeteilt, die den Analyseschritten oberster Ordnung entsprechen (d.h. Geschäftsbereichsanalyse, Einzelprozessanalyse usw.).

An Hand der Schlüsselfragen, die im Folgenden die einzelnen Modellierungsschritte bezeichnen, kann eine Checkliste aufgestellt werden, die systematisch abgearbeitet werden kann.

2.2 Geschäftsbereichsanalyse

2.2.1 Aufgaben und Ziele

Die Geschäftsprozesse des Straßen- und Verkehrswesens lassen sich einer Reihe von Bereichen zuordnen. Dies wurde bereits beim Aufbau des bestehenden OKSTRA erkannt und berücksichtigt und konnte bei der Aufstellung des Geschäftsprozesskataloges (GP-Neu 02) bestätigt werden. Dort wurde der Begriff *Geschäftsbereich* als Gliederungseinheit für die Prozesse vorgeschlagen und auch Hinweise zum Abgrenzen der Geschäftsbereiche gegeben.

> Ein wichtiges Mittel zur Geschäftsprozessanalyse ist das Auffinden von Teilprozessen, die, abgesehen von unterschiedlichen Randbedingungen, in einem Gesamtprozess wiederkehrend dem gleichen Geschäftsziel dienen (*Faktorisierung).* Z.B. ist während Planung und Durchführung eines Bauvorhabens zu verschiedenen Zeitpunkten eine *Kostenermittlung* durchzuführen.
>
> **Hinweis**: Im Folgenden wird Geschäftsbereich oft zu Bereich abgekürzt.

Die zentrale Eigenschaft eines Geschäftsbereichs ist, dass seine Prozesse eine geschlossene Datenbasis bereitstellen, ergänzen und fortschreiben. Außer den Prozessen des Bereichs modifizieren keine anderen die Datenbasis.

Anderen Geschäftsbereichen steht die Datenbasis nur zur lesenden Nutzung zur Verfügung.

> Beispiele für diese Nutzung gibt es zuhauf: Der Straßenentwurf nutzt die Ergebnisse von Verkehrsanalysen und Umweltuntersuchungen, die Kostenermittlung benötigt die Mengenangaben, die sich aus den Objekten des Straßenentwurfs ergeben usw. usf.
>
> **Zu beachten**: Geschäftsbereich ist nicht gleichzusetzen mit Organisationseinheit eines Amtes oder Unternehmens!

Die Geschäftsbereichsanalyse ermittelt die Geschäftsprozesse eines Geschäftsbereiches. Sie ist der erste Schritt zur Auffindung der Objekte. Da die Geschäftsprozesse eines Bereichs sich alle auf dieselbe zu unterhaltende Datenbasis beziehen, kann die spätere detaillierte Analyse der Geschäftsprozesse die Struktur dieser Datenbasis offenlegen und damit die für den Geschäftsbereich typischen Objekte aufzeigen.

Am Ende der Geschäftsbereichsanalyse ist bekannt, welche Prozesse es in dem Geschäftsbereich gibt, aber noch nicht, wie sich diese im Einzelnen aus „logisch verknüpften Arbeitsschritten“ zusammensetzten.

Bei der Geschäftsbereichanalyse sind *alle* Prozesse zu ermitteln, unabhängig davon, ob sie durch IT-Verfahren unterstützt werden oder werden können. Da am Ende eines Prozesses so gut wie immer irgendwelche dokumentierten Ergebnisse stehen, die sich zumindest elektronisch speichern und übertragen lassen, gibt es praktisch keine Geschäftsprozesse, die man für die OKSTRA-Modellierung von vornherein ausschließen könnte.

Die Geschäftsbereichsanalyse gliedert sich in folgende Schritte:

2.2.2 Welcher Bereich ist zu untersuchen?

Die Geschäftsbereiche können einem entsprechend strukturierten Geschäftsprozesskatalog entnommen werden, der möglicherweise zuerst zu erstellen ist. (Als Beispiel und Anleitung diene (GP-Neu 02)). Ist eine Gliederung des Kataloges in Bereiche noch nicht vorgegeben, so muss diese zuerst erfolgen.

> Beispiel: der Geschäftsbereich *Kosten (Baumaßnahme)* umfasst alle Prozesse zur Schätzung, Kontrolle und Festsetzung von Kosten einer Baumaßnahme.

2.2.3 Welche Prozesse gibt es im Bereich?

Die zum ausgewählten Geschäftsbereich gehörenden Prozesse können ebenfalls unmittelbar dem Geschäftsprozesskatalog entnommen werden (z.B. *Kostenberechnung*), jedenfalls ist dies bei der Strukturierung des Kataloges anzustreben.

Liegt keine oder nur eine unvollständige Liste der Prozesse des Bereichs vor, so ist diese zu erstellen, und zwar, in dem man die Geschäftsziele ermittelt, die mit dem Bereich verbunden sind: Welchem Zweck dient der Geschäftsbereich, welche Teilziele gibt es, wie werden sie erreicht?

Dieser Schritt verlangt zunächst nur eine Stoffsammlung; die Abhängigkeiten und das Zusammenwirken der Prozesse werden an dieser Stelle noch nicht untersucht.

2.2.4 Wodurch können die Prozesse erschlossen werden?

Dieser Schritt sammelt das vorhandene Wissen über die Realwelt bezogen auf den Geschäftsbereich. Er ist zunächst nur eine Ressourcenbereitstellung, deren Umfang jedoch nicht unterschätzt werden darf. Als Informationsquellen zu den Geschäftsprozessen kommen in Frage (ohne erschöpfend sein zu wollen):

- Administrative und technische Regelwerke, z.B. Normen, Richtlinien, Gesetze usw. (Beispiel für die Kostenberechnung: Regelwerke *AKS 85, HOAI*)
- Verfahrensdokumentationen, z.B. Ablaufdiagramme, Arbeitsanweisungen, Schulungsunterlagen (auch Lehrbücher), Tagebücher und Verlaufsprotokolle für vergangene Projekte usw.
- Arbeitsergebnisse, z.B. Berichte, Pläne und Listen
- Verzeichnisse und Beschreibungen genutzter IT-Anwendungen
- Praxiswissen, das durch Befragen von Fachleuten mit Kompetenz für den Geschäftsbereich erschlossen werden muss. Diese Informationsquelle ist besonders wichtig, weil sie gängige Interpretationen der Regelungen sowie typische Probleme im Geschäftsablauf aufdecken hilft.

2.2.5 Welche Informationen bearbeiten die Prozesse?

Dieser Schritt beschreibt aus dem Wissen, das im vorigen Schritt gewonnen wurde, die Datenbasis, die durch die Prozesse erzeugt, fortgeschrieben und ausgewertet wird (z.B. die *Kostenaufstellungen* und *Preisdokumentationen* für die *Kostenberechnung*), und zwar unabhängig davon, wie die Daten tatsächlich gespeichert werden.

> Es kann sich im Einzelfall um Datensammlungen beliebiger Art handeln, also Akten, Pläne, Dateien, Datenbanken, Online-Ressourcen, multimediale Daten usw. usf.

Die Daten müssen in den Geschäftsprozessen regelmäßig bearbeitet und nicht nur passiv genutzt werden.

Zu beachten ist, dass an dieser Stelle die *Struktur* der Daten der Datenbasis noch nicht untersucht wird, dies bleibt später dem Schritt 2.5 vorbehalten.

2.2.6 Welche anderen Bereiche berühren den gerade untersuchten?

Genauer: Welche anderen Geschäftsbereiche liefern Eingangsdaten und welche nutzen die Ergebnisse der Prozesse des vorliegenden Bereichs? Der Schritt identifiziert also Kommunikationskanäle zu anderen Bereichen.

> Der Bereich Grunderwerb hat u.a. Kanäle zu: Straßenentwurf (die von der neuen Straße belegten Flächen), Umwelt (die benötigten Ausgleichsflächen), Bauorganisation (während des Baus benötigte Lager-und Zufahrtsflächen), Kostenermittlung.

Auch hier ist die Struktur der Kommunikation (was wird kommuniziert? in welcher Richtung?) noch nicht Untersuchungsgegenstand.

2.3 Einzelprozessanalyse

2.3.1 Aufgaben und Ziele

Die Einzelprozessanalyse nimmt sich jeweils einen Geschäftsprozess des ausgewählten Geschäftsbereichs vor, um die dazu notwendigen Arbeitsschritte zu bestimmen (z.B: die *Kostenberechnung*). Hierzu werden die im vorigen Schritt identifizierten Informationsquellen ausgewertet (z.B. die *AKS 85*). Der Begriff *Arbeitsschritt* wird wie folgt präzisiert:

Ausgehend von der Definition: Ein Geschäftsprozess ist „eine Menge logisch verknüpfter *Arbeitsschritte*, die durchgeführt werden, um ein bestimmtes Geschäftsziel zu erreichen“, wird von einem Arbeitsschritt gefordert:

- Ein Arbeitsschritt hat definierte Voraussetzungen (z.B. vorliegende Daten), und definierte Ergebnisse (z.B. Dokumente oder die Auswahl und den Anstoß des nächsten Arbeitsschrittes).
- Aufeinanderfolgende Arbeitsschritte sind miteinander logisch verknüpft, in dem die Ergebnisse des einen zu den Voraussetzungen des anderen beitragen. (Der Arbeitsschritt *Räumliche Teile bestimmen* strukuriert z.B. eine Kostenberechnung, er muss vor der Berechnung der Kosten selbst erfolgen)

Die Arbeitsschritte werden ab hier als *Aktivitäten* bezeichnet.

Aus der detaillierten Kenntnis der Aktivitäten kann später erschlossen werden, mit welchen Objekten es der Geschäftsprozess zu tun hat.

Die Abgrenzung von Prozess und Aktivität ist fließend. Als Abgrenzungskriterium kann gelten: Ein Prozess sollte als Ergebnis ein fertiges Produkt haben, das im Sinne der Geschäftstätigkeit nutzbringend verwendbar ist. Die prozess-orientierte Betrachtungsweise stellt also das Ziel oder Produkt in den Vordergrund. Die Betrachtungsweise nach Aktivitäten stellt dagegen den Weg zum Ziel in den Vordergrund, d.h. den Ablauf des Prozesses. Die beiden Sichtweisen schließen sich nicht aus, sondern ergänzen sich; so kann ein Prozess als Aktivität im Zusammenhang eines übergeordneten Prozesses angesehen werden.

Das Ergebnis der Einzelprozessanalyse ist eine detaillierte Beschreibung des Ablaufes des Prozesses. Außerdem ist danach bekannt, welche Leistungen ein IT-System anbieten muss, um den Prozess zu unterstützen.

Die Schritte der Einzelprozessanalyse sind:

2.3.2 Welche Aktivitäten gibt es im Prozess?

Dieser Schritt sammelt zunächst nur die Aktivitäten, ohne sich um die Reihenfolge zu kümmern. Aus den Informationsquellen ist teilweise nichts über die Reihenfolge zu erfahren, so dass sich zunächst eine Inventur anbietet.

2.3.3 Wie wird der Prozess von außen beeinflusst?

Ein Prozess kommt nicht von allein in Gang. Es sind äußere *Ereignisse*, die ihn anstoßen. Ebenso kann der Prozess durch äußeren Einfluss in seinem Ablauf beeinflusst, unterbrochen, wieder aufgenommen oder sogar vorzeitig abgebrochen werden.

Typische Ereignisse sind:

- Beginn und Ende eines Prozesses
- der Übergang von einem Stadium eines Prozesses in das nächste

  Zur Erläuterung: Prozesse durchlaufen (außer in den allereinfachsten Fällen) mehrere Stadien, die häufig durch typische Aktivitäten von einander abgegrenzt sind, z.B. bei Baumaßnahmen: Linienbestimmung, Planfeststellung, Vergabe.
- die Änderung eines Datenbestandes oder Objekts
- das Erreichen eines bestimmten Zeitpunktes (Datum, Uhrzeit)

  Der Standardisierungsvorschlag [BPMN 03] teilt Ereignisse nach ihrer Position im Prozess ein in Start-, Ende- und Zwischenereignisse, sowie weiter nach ihrem Auslöser, z.B. Eintreffen eines Signals, Erreichen einer Zeitmarke, Eintreten einer Bedingung (z.B. „Lagerbestand kleiner 5“), Storno, Fehler usw.

Durch *Signale* werden Ereignisse allen Prozessen mitgeteilt, die darauf reagieren müssen.

> Beispiel: Als Ergebnis eines Planfeststellungsverfahrens müssen evtl. neu hinzugekommene Umweltgesichtspunkte berücksichtigt werden; diese erzwingen dann eine Planungsänderung sowohl des Straßenentwurfs als auch des LBP; dies erzwingt wiederum eine Änderung des Grunderwerbs, was schließlich alles zusammen eine *Fortschreibung der Kostenberechnung* erforderlich macht.

In diesem Schritt der Modellierung erfolgt eine Inventur aller Ereignisse, die von außen auf den Prozess einwirken können sowie jener, die im Prozess selbst eintreten und ihrerseits andere Prozesse beeinflussen. Zu jedem äußeren Ereignis wird festgehalten, welche Auswirkungen es im untersuchten Prozess hat.

2.3.4 Wie folgen die Aktivitäten aufeinander?

Die Reihenfolge der Aktivitäten ergibt sich aus der Übergabe von Information zwischen ihnen. Weiß man, welche Aktivität die Ergebnisse welcher anderen benötigt, kann man die Arbeitschritte *anordnen*, wobei manche auch *parallel* zueinander ausgeführt werden können, wenn sie nicht aufeinander angewiesen sind. Oft machen auch Regelwerke und Verfahrensanweisungen Aussagen über einzuhaltende Reihenfolgen.

Die Untersuchung der Reihenfolgebeziehungen dient später der Ermittlung der Nachrichtenkanäle zwischen den Objekten. Der Informationstransport zwischen aufeinanderfol-

genden Aktivitäten schlägt sich im objektorientierten Modell so nieder, dass die Objekte, mit denen in den Aktivitäten hantiert wird, ihre Zustände ändern.

Die Reihenfolge der Arbeitsschritte kann durch die Ergebnisse vorheriger Schritte oder äußere Ereignisse modifiziert werden.

Dieser Schritt hat somit zur Aufgabe:

- Reihenfolgebeziehungen zu finden,
- Parallelitäten zu finden,
- Verzweigungen im Ablauf zu finden,
- Punkte im Ablauf zu finden, wo auf Ereignisse gewartet und eingegangen werden muss,
- Punkte im Ablauf zu finden, an denen Ereignisse eintreten, die anderen Prozessen signalisiert werden müssen.

  Beispiel für eine Reihenfolgebeziehung: *Zuerst* muss die Maßnahme in Teile und Lose gegliedert werden, *dann* kann die Berechnung der Kosten erfolgen.

2.3.5 Was sind die Anwendungsfälle für das OKSTRA-Modell?

Aus der Aufzählung und Anordnung der Aktivitäten allein lassen sich noch keine Objekte, und deren Kommunikationen herleiten. Um eine Aktivität erfolgreich durchzuführen, muss jemand bestimmte Handlungen ausführen, deren Summe schließlich zu Ziel und Ergebnis der Aktivität führen. In der Terminologie der objektorientierten Modellierung werden die „bestimmten Handlungen“ als *Anwendungsfälle* bezeichnet, die von *Akteuren* durchgeführt werden.

Die Ermittlung der Anwendungsfälle geht so vor sich, dass an Hand des in den vorigen Schritten gewonnenen Prozessablaufes die folgenden Fragen beantwortet werden:

- Wer ist am Prozess beteiligt?

  Dabei bezieht sich das „Wer?“ nicht auf einzelne Personen, sondern auf Gruppen von Teilnehmern in einer bestimmten Rolle, Funktion oder Aufgabe. Der modelltechnische Begriff hierfür ist *Akteur*, die Antwort auf die Frage liefert also alle Akteure im Prozess. Akteure versorgen die Datenbasis eines Geschäftsbereichs mit Daten oder nutzen sie zur Informationsgewinnung. Kandidaten für Akteure sind:

  - *Funktionsträger*, z.B Planer,
  - IT-*Systeme*, die eine bestimmte Aufgabe lösen helfen, z.B. Entwurfssystem
  - andere *Geschäftsbereiche*, z.B. Landschaftsplanung
  - externe *Ereignisse*, z.B. Fristablauf, Gerichtsentscheid

    Bezeichnungen wie „Ingenieurbüro“, „Straßenmeisterei“ usw. sind keine Funktions-, sondern Organisationsbegriffe; sie können als Erläuterung zum Funktionsbegriff hinzugefügt werden, dürfen aber nicht alleine stehen.

  Akteure werden entweder individuell bezeichnet (z.B. Bundesminister für Verkehr, KOSTRA usw.) oder generisch (z.B. Planer, Straßeninformationsbank, Planfeststellung). Individuelle Namen sind um eine Rollenbezeichnung zu ergänzen, falls die Rolle sich nicht aus dem Namen erschließen lässt.

- Was passiert während einer Aktivität im Detail?

Hierzu werden die Aktivitäten am besten in Form von Szenarien („Drehbüchern“) beschrieben. Manchmal liegen solche fertig vor, z.B. als Handbücher für die Bedienung einschlägiger Anwendungsprogramme, administrative Verfahrensanweisungen oder aus der Praxis geborene „Kochrezepte“ für bestimmte Aufgaben. Ist dies nicht der Fall, müssen die Szenarien zuerst durch Befragen oder Beobachten gewonnen werden. In den Drehbüchern wird die Aktivität durch Sätze beschrieben, in denen die Akteure etwas tun, und zwar in Bezug auf irgendwelche Dinge.

Beispiele:

Der Sachbearbeiter für die Netzfortführung teilt einen Abschnitt an einem neuen Nullpunkt.

Zur Kostenberechnung entnimmt der Planer einer Preisdokumentation einen Preis für eine bestimmte Leistung, abhängig von der benötigten Menge.

Als Beschreibungen für Anwendungsfälle werden alle solche Sätze betrachtet, die für die die Modellierung des OKSTRA relevant sein sollen.

Existieren bereits IT-Anwendungen, die eine Aktivität unterstützen, so können Anwendungsfälle auch aus den Bedien-Dialogen der Anwendung abgeleitet werden, allerdings sollte hierbei immer eine darin eingearbeitete Person um Erläuterung gebeten werden.

Hier sind folgende Anmerkungen zu machen:

- Anwendungsfälle sind nicht unbedingt einer bestimmten Reihenfolge unterworfen, das unterscheidet sie von Aktivitäten.
- Aktivitäten können mit Anwendungsfällen zusammenfallen, müssen das aber nicht.
- Derselbe Anwendungsfall kann in verschiedenen Aktivitäten vorkommen.

Die Menge der gefundenen Anwendungsfälle wird dann weiter strukuriert:

- Einander ähnliche Anwendungsfälle können als *spezialisierte Varianten* eines generalisierten Anwendungsfalles modelliert werden.
- Anwendungsfälle, die als fester Bestandteil mehrerer anderer Anwendungsfälle auftreten können, werden mit diesen verknüpft. (Siehe unter ***<<include>>***, Kap. 5)

2.4 Dokumentation der Prozesse

Das Ergebnis der vorherigen Analyseschritte wird in Form von *UML-Aktivitäts- und UML-Anwendungsfalldiagrammen* dargestellt (siehe Kap. 5 und Anhang A). Dabei sind Prozesse, Aktivitäten, Ereignisse, Anwendungsfälle und Akteure in angemessenem Umfang textlich zu erläutern, wenn möglich im Diagramm, ansonsten durch einen Verweis im Diagramm auf erläuternden Fließtext. Insbesondere gehört zur Kommentierung ein Hinweis auf die zugrundeliegenden *Quellen*.

Es empfiehlt sich, die Diagramme schon während der Analyseschritte anzulegen und zu ergänzen und in diesem Schritt nochmals in ihrer Gesamtheit zu prüfen und zu redigieren

Anmerkung: Aus der Erkenntnis, dass für einen erfoglreichen Ensatz von IT in Organisationen die Geschäftsprozessmodellierung von zentrale Bedeutung ist, sind in jüngster Zeit mindestens 7 Sprachen und Notationen entstanden und z.T. zur Standardisierung eingereicht worden. Einen Überblick gibt:

http://www.ebpml.org/status.htm

Ob und wann eine Konvergenz dieser Ansätze erfolgt, bleibt abzuwarten. Es wird dringend empfohlen, die Standardisierungsaktivitäten in diesem Bereich zu beobachten. Für den objektorientierten OKSTRA empfiehlt sich bis auf weiteres die Verwendung von UML 2.0, weil diese eine integrierte Modellierung der Prozesse und der Objektwelten erlaubt.

2.5 Objektwelt

2.5.1 Aufgaben und Ziele

Die Ergebnisse der Prozessmodellierung der vorigen Abschnitte können nun zur Bildung der *Objektwelt* für die einzelnen Prozesse herangezogen werden. Die Notwendigkeit eines solchen Schrittes ergibt sich einfach aus der Tatsache, dass objektorientierte Modelle die in der Realität aufgefundenen Konzepte in Klassen gleich strukturierter Objekte und Nachrichtenkanäle zwischen Objekten übersetzen.

Ziel des Schrittes ist, alle die Begrifflichkeiten aus der Prozessbeschreibung herauszufiltern, denen eine eigenständige Existenz zugestanden wird. Wirklich scharfe Kriterien hierfür gibt es nicht, und im Einzelfall können durchaus Ermessensspielräume vorliegen, die zu alternativen Modellansätzen führen. In solchen Fällen ist es zweckmäßig, die Alternativen parallel weiterzuverfolgen und am Ende schließlich die zu wählen, die möglichst natürlich und verständlich wirkt und eine vergleichsweise geringe Komplexität aufweist. Eine zu reichhaltige Objektwelt birgt die Gefahr, dass u.U. sehr viele Nachrichtenkanäle zwischen den Objekten nötig werden, während eine zu arme Objektwelt die Tendenz hat, die Objekte zu überlasten: die Zuständigkeiten sind nicht klar getrennt auf die Objekte verteilt, Objekte nehmen Aufgaben wahr, die eigentlich nicht zusammengehören.

Am Ende des Schrittes sind alle Objekte bekannt, mit denen der Prozess umgeht.

2.5.2 Welche Objekte tauchen in den Anwendungsfällen auf?

Die Antwort ergibt sich aus den Quellen, auf die die Anwendungsfälle verweisen, also aus den Ergebnissen von Schritt 2.3.5. Dabei werden zunächst die Begriffe ausgesondert, die auf Akteure verweisen:

im Beispiel oben: *Sachbearbeiter für die Netzfortführung* und *Planer*

Auf den übrigen setzt eine inhaltliche Analyse auf, die zunächst die vorkommenden typischen und wiederkehrenden Begrifflichkeiten (Konstrukte) herausfiltert:

im Beispiel oben: *Preisdokumentation, Preis, Leistung, Menge*

Aus diesen sind dann diejenigen auszuwählen, die Objekte werden sollen. Das Kapitel 4 nennt hierzu eine Reihe von Kriterien:

im Beispiel: Preisdokumentation und Leistung, denn eine Preisdokumentation hat eine Identität (z.B. als Datei)

Für die spätere Bildung der Dienste ist es nötig, zu wissen, welche Objekte gemeinsam für einen Anwendungsfall zuständig sind, deshalb muss hier auch die Zuordnung der Objekte zu den Anwendungsfällen dokumentiert werden.

Zusätzlich sollten an dieser Stelle *Anforderungen*, die zu den Anwendungsfällen gehören, in Anforderungen an die Objekte umgesetzt werden.

Beispiele für Anforderungen: Genauigkeit von Ergebnissen, Dauerhaftigkeit der Speicherung, Leistungsfähigkeit der Verarbeitung

2.5.3 Wie lange leben die Objekte?

Objekte haben einen Lebenslauf. In einer dienste-basierten Architektur werden Objekte von bestimmten Diensten neu erzeugt, von anderen verändert und möglicherweise später von wieder anderen vernichtet, wenn sie keine brauchbare Information mehr liefern können.

Dieser Schritt untersucht daher, bei welchen Anwendungsfällen Objekte neu entstehen, verändert und beseitigt werden. Objektlebensläufe lassen sich ebenfalls in UML-Aktivitätsdiagrammen darstellen. Eine andere Technik, die sich jedoch weniger gut für die Kommunikation mit Nicht-IT-Fachleuten eignet, verwendet UML-Zustandsdiagramme.

2.6 Operationen

2.6.1 Aufgaben und Ziele

Das Merkmal eines objektorientierten Modells ist, dass die Objekte darin Nachrichten austauschen, die Informationen erfragen, die inneren Zustände der Objekte ändern oder zur Neuerzeugung oder Beseitigung von Objekten führen. Der Austausch von Nachrichten wird formal durch *Operationen* dargestellt. Eine Operation wird zumeist im Kontext eines einzelnen Objektexemplars *ausgeführt*. Dabei kann sie die Funktionen, die andere Objekte bereitstellen, dadurch nutzen, dass sie deren Operationen anspricht. Dabei wird die Operationen mit Parametern versorgt, die genauer spezifizieren, was die Operation tun soll. Nachdem die angesprochene Operation durchgeführt wurde, kann sie dem anfordernden Objekt ein Ergebnis mitteilen.

Alle in den vorherigen Schritten aufgefundenen Objekte müssen daher in diesem Schritt mit Operationen ausgestattet werden, so wie beim OKSTRA in seiner statischen Form die Objekte mit Attributen und Relationen ausgerüstet werden. Die Details des Operationsbegriffes und sein Zusammenhang mit den klassischen Attribut- und Relationsbegriffen sind in Kapitel 3 dargestellt.

2.6.2 Welche Verantwortlichkeiten nehmen die Objekte wahr?

Aus dem vorher erarbeiteten Prozessmodell werden zunächst die Verantwortlichkeiten der Objekte ermittelt. Dieser Begriff ist analog zum Erteilen von Zuständigkeiten in einer Organisation zu sehen: Ein Objekt ist für eine ganz bestimmte Aufgabe (oder einen Aufgabenkreis) zuständig. Andere Objekte lösen diese Aufgabe nie selbst, auch wenn sie bei Verfolgung ihrer eigenen Zuständigkeiten zur Lösung ansteht. In so einem Fall wird die Aufgabe an das Objekt *delegiert*, das sie lösen kann.

> Beispiel: Im Gesamtzusammenhang einer Kostenberechnung sind einige Objekte dafür zuständig, Informationen über die Leistung zu erteilen, zu der sie etwas beitragen, sowie über ihren quantitativen Anteil daran.

2.6.3 Welche Nachrichten müssen zwischen den Objekten ausgetauscht werden?

Wie gesagt, bedeutet die Verteilung von Zuständigkeiten an die Objekte: Wenn immer eine bestimmte Teilaufgabe innerhalb des Informationssystems ansteht, wird ein hierfür zuständiges Objekt angesprochen. Um dieses herauszufinden, werden die Aufgaben der Objekte wiederum in Form von Szenarien analysiert. Ein Szenario ist diesmal eine detaillierte Beschreibung des Handlungsablaufes für einen einzelnen Anwendungsfall, wobei

die Handelnden aber nicht Personen, sondern die Objekte sind. Die vorher beschriebenen Schritte 2.2.4, 2.2.5 und 2.4.2 liefern das Ausgangsmaterial für die Analyse.

> Beispiel: In der Kostenberechnung ist zu einer bestimmten Leistungsposition eine Menge zu ermitteln. Das Objekt, das die Berechnung durchführt, muss also den Objekten, die die realen Gegenstände zur Leistungsposition (wie z.B. Rohrleitungen) repräsentieren, eine Nachricht senden, die um Angabe der Menge bittet.

Daneben ist die Erzeugung und Vernichtung von Objekten zu berücksichtigen. Erst wenn ein Objekt existiert, kann es zum Empfänger von Nachrichten werden und seine Verantwortlichkeit auch wahrnehmen. Es ist also sicherzustellen, dass vor dem Bearbeiten einer Aufgabe ein dafür zuständiges Objekt existiert. Die Grundlagen für diese Untersuchung stammt aus 2.5.3. Hilfsmittel für diesen Schritt sind die UML-Sequenzdiagramme, die Übermittlungsszenarien für Nachrichten veranschaulichen. Aber auch UML-Aktivitätsdiagramme können eingesetzt werden.

2.6.4 Wie sehen die Operationen aus?

Hier wird festgestellt, welche Informationen jede Nachricht enthalten muss, damit das Empfängerobjekt beim Empfang seiner Verantwortlichkeit auch nachkommen kann. Verlangt die Nachricht eine Auskunft, ist zudem der Inhalt der Antwort festzulegen.

> Beispiel: Für das obige Beispiel wird z.B. die Leistungsposition als KBK- oder STLK-Nummer übermittelt und die Menge, z.B. die Länge der Rohrleitung, kommt als Antwort zurück.

Mit diesem Schritt sind die Operationen *formal* festgelegt.

2.6.5 Was tun die Operationen?

In diesem letzten Schritt der Konzeptionsmodellierung ist festzulegen, was jede Operation an Informationsverarbeitung leisten muss, d.h. welche Berechnungen, Speicherfunktionen, Interaktionen mit dem Benutzer etc. Der Detaillierungsgrad kann dabei sehr unterschiedlich sein. Im Allgemeinen sollte das Wie der Verarbeitung nicht vorgegeben werden, um die Implementationsfreiheit nicht einzuschränken, es sind aber Fälle denkbar, wo die Einhaltung bestimmter rechnerischer Verfahren von den zugrundeliegenden Regelwerken zwingend vorgeschrieben ist, d.h. Anforderungen an die Objekte sollten in den Funktionsbeschreibungen für die Operationen berücksichtigt werden.

> Beispiel: Für das obige Beispiel wird etwa gefordert, dass die benötigte Menge aus der Geometrie der Leitung und mit einer bestimmten Toleranz nach oben zu berechnen ist.

Mit diesem Schritt sind die Operationen *funktional (semantisch)* festgelegt.

2.7 Definition von Diensten

2.7.1 Aufgaben und Ziele

Typischerweise wird ein Anwendungsfall durch Zusammenarbeit mehrerer Objekte erledigt. Zudem ergibt sich aus der Strukturierung der Anwendungsfälle u.U., das bestimmte Teilaufgaben in mehreren Anwendungsfällen zu lösen sind. Zweckmäßig werden dann die Objekte, die zusammen solche Teilaufgaben abarbeiten, als höhere Einheit betrachtet werden, als sogenannte *Kollaborationen*. Stellt man die Leistungen einer Kollaboration in Form ihrer Operationen als ganzes gebündelt zur Verfügung, erhält man einen *Dienst* zu Lösung einer bestimmten Aufgabe.

Das Auffinden der Kollaborationen und das Zusammenfassen der relevanten Operationen ist die Aufgabe dieses Modellierungsschrittes.

2.7.2 Welche Objekte arbeiten zusammen?

In diesem Schritt wird zunächst das Zusammenwirken der Objekte für die einzelnen Anwendungsfälle durchgespielt und in Form von UML-Sequenzdiagrammen dokumentiert, wobei die Szenarien aus 2.6.3 die Grundlage bilden. Dabei bilden sich Gruppen von Objekten, die gemeinsam bestimmte Teilaufgaben lösen.

> Beispiel: Bei der Kostenberechnung arbeiten die Objekte AKS_Berechnung, AKS_Teil, KBK und KBK_Position Objekte zusammen, um eine Berechnung zu erstellen. (Siehe Anhang A).

2.7.3 Welche Dienste sind erforderlich?

Hierzu werden die Operationen der Objekte einer Kollaboration zusammengefasst, wobei zwischen Objekten der Kollaboration allein wirksame Operationen als dienstinterne Angelegenheit wegfallen.

Technisch wird der Dienst als ein neues Objekt eingeführt. Die ursprünglichen Operationen der Kollaboration werden als solche des Dienstobjektes umformuliert, wobei bei jeder Operation deren definierendes Objekt nunmehr als Parameter eingeführt werden muss. Hinzuzufügen sind i.d.R. Operationen zum Erzeugen und Vernichten von Objekten der Kollaboration (Fabriken, s. Kapitel 7).

> Beispiel: Im Kostenberechnungsmodell bilden der AKS_Manager, die zusammen einen Dienst.
>
> Das Objekt AKS_Berechnung enthält eine Operation Berechne(), die vom Objekt AKS_Manager verwendet wird, um einerseits Neuberechnungen durchzuführen, andererseits existierende Berechnungen zu aktualisieren. Da die Operation nur vom AKS_Manager benötigt wird, kann sie in der Schnittselle des Dienstes entfallen.
>
> Anders die Operation Präsentiere(). Sie wird benötigt, um eine Berechnung auszudrucken. Auf der Dienstschnittstelle erscheint sie als Präsentiere(B), wobei B die Objekt-Identifikation für die zu druckende Berechnung ist.

2.8 Review des Konzeptionsmodells

Hier endet die konzeptionelle Modellierung und es beginnt die Spezifikationsmodellierung. Sie wird im Regelfall von den IT-Modellierern allein bestritten, daher ist es erforderlich, dass das vorliegende Konzeptionsmodell von den Fachexperten verstanden und akzeptiert wird.

Die erarbeiteten formalen Modelle müssen deshalb vorher daraufhin überprüft werden, ob sie die untersuchten Geschäftsprozesse vollständig und widerspruchsfrei wiedergeben. Dies geschieht am besten durch ein fachkundiges *Review*-Team, das das Modell aus Kenntnis der Quellen und der täglichen Praxis begutachtet. Die Ergebnisse des Reviews werden in einer *Mängelliste* festgehalten, die die gefundenen Mängel identifiziert und nach Möglichkeit Korrekturvorschläge macht.

Zum Review muss geprüft werden

- ob alle Geschäftsprozesse berücksichtigt wurden,
- ob die gefundenen Aktivitäten die Prozesse vollständig und regelkonform darstellen,
- ob die Anwendungsfälle die Aktivitäten sinnvoll unterstützen (z.B. durch Beseitigung von *Medienbrüchen*; siehe hierzu Kapitel 1)

- ob die gefundenen Dienste mit ihren Operationen die Abwicklung der Anwendungsfälle erlauben. Dies kann z.B. durch Erstellen einer Testimplemenation (Prototyping) erfolgen.

Sofern für die Geschäftsprozesse keine verbindlichen, bundeseinheitlichen Regeln vorliegen, ist damit zu rechnen, dass es regionale Varianten gibt. Diese sind in den Review mit einzubeziehen.

Die aus dem Review resultierenden Korrekturen werden, beginnend bei Schritt 2.2.3, in das Modell eingearbeitet, worauf dann der Reviewschritt wiederholt wird.

2.9 Formalisierung Bildung und Dokumentation des Modells

2.9.1 Aufgaben und Ziele

Dieser Schritt dokumentiert das Objektmodell vollständig als Klassendiagramm und mit funktionalen Beschreibungen. Die Anforderungen hierzu finden sich im Kapitel 6. Es bringt das Modell außerdem in eine Form, die unmittelbar zur Software-Entwicklung eingesetzt werden kann.

2.9.2 Objektmodell als Klassendiagramm formulieren

Die gefundenen Objekte werden mit den Operationen, mit denen sie ausgestattet wurden, zu Objekttypen. Diese werden als UML-Klassendiagramme gemäß Kapitel 5 dokumentiert. Dabei werden

- Attribute und Operationen formal beschrieben,
- Assoziationen gebildet, sofern sie nicht bereits im Konzeptionsmodell formuliert wurden, und formal beschrieben
- gemeinsame Funktionen in Basisklassen ausgegliedert, sofern sie nicht bereits im Konzeptionsmodell formuliert wurden

Zu den Begriffen Operation, Attribut etc. und ihren Zusammenhängen siehe Kapitel 3.

Hinweis: Paare von Gib-Setze-Operationen können gemäß Kapitel 3 als Attribute bzw. Assoziationen dargestellt werden, häufig ist es jedoch gerade bei Assoziationen klarer, die Operationen in der Notation beizubehalten und die Assoziation als Hinweis darauf zu notieren, dass sich die verbundenen Objekte kennen müssen. (Im Modell müssen wenigstens immer dann Assoziationen zwischen Klassen eingetragen werden, wenn Objekte der einen Klasse Operationen der anderen Klasse verwenden.)

Es existiert jetzt ein Bauplan für OKSTRA-konforme Dienste.

2.9.3 Modell publizieren

Das nunmehr fertige Modell in UML und freiem Text wird in eine IT-gerecht transportable und weiterverarbeitbare Form gebracht, die es gestattet, daraus modellgerechte Implementationen zu schaffen. Hierzu gehören im Einzelnen folgende Aufgaben:

- Ausgabe des gesamten Modells in einem standardisierten Format, z.B. im XMI (XML Metadata Interchange Format), so dass es mit Software-Entwicklungswerkzeugen dargestellt und weiterverarbeitet werden kann.

- Ausgabe der grafischen Darstellung und der freien, erläuternden Texte in standardisiertem Format, so dass daraus Dokumente hergestellt werden können (z.B. als SVG-Grafik-, HTML- oder PDF-Dateien)
- Herstellung von Sprachbindungen: Umsetzung des Modells in programmiersprachenspezifische Schemata, z.B. in Java, C++, C#, XML-Schema, WSDL (Web Service Description Language) oder IDL (Interface Definition Language)

Diese Aufgaben sind nicht unabhängig voneinander zu sehen. Sprachbindungen können z.B. durch Entwicklungswerkzeuge automatisch erzeugt werden. Andererseits gibt es keine Möglichkeit, aus einer XMI-Darstellung auf eine eindeutige Weise die ursprüngliche grafische Präsentation wiederzugewinnen. UML 2.0 Diagram Interchange erweitert die inhaltliche XMI-Repräsentation um eine zusätzliche Wiedergabe der Grafik. Näheres zu diesem Thema in Kapitel 6.

3 Begriffe der Objektorientierung für OKSTRA-Modelle

3.1 Ziele und Umfang

Die im folgenden gegebenen Definitionen für die Begriffe zum objektorientierten Modellieren wurden speziell im Hinblick auf die Modellierungsaufgabe OKSTRA getroffen. Ziele bei ihrer Auswahl und Definition sind:

- *Kompatibilität* mit dem bestehenden OKSTRA
- *Sparsamkeit* der Begriffe
- *Unabhängigkeit* von Programmiersprachen oder Implementations-Technologien
- Schwerpunkt auf der *Konzeptmodellierung*

Die objektorientierten Modelle für den OKSTRA sollen ausschließlich mit den hier eingeführten Konstrukten formuliert werden.

Der Begriff Modell, so wie er hier verwendet wird, beschreibt die Eigenschaften eines Informationssystems, die unabhängig von den Inhalten einzelner Installationen des Systems sind. In der konventionallen Datenbankmodellierung entspricht dem Modellbegriff daher der Begriff des Schemas. Modelle bilden Konzepte des zu Grunde liegenden fachlichen Problembereichs auf formale Konstrukte, wie z.B. Klassen, ab. Die Welt der bei der Modellierung verwendeten Konstrukt-Typen läßt sich selbst wieder als Modell einer höheren Ebene auffassen. Solche höheren Modelle werden *Metamodelle* genannt. Beispiele für Metamodelle mit einigen ihrer Konstrukte sind:

- NIAM mit Objekt und Rolle
- ER mit Entityset und Relationship
- das Relationale Modell mit Tabelle, Zeile, Spalte
- UML mit Klasse, Operation, Attribut, Assoziation

In diesem Sinne beschreibt das Kapitel 3 auf informelle Weise das Metamodell für den objektorientierten OKSTRA.

3.2 Objekte und Operationen

***Leitsatz**: Ein objektorientiertes Objektmodell für den OKSTRA beschreibt Objekte durch die für sie verfügbaren Operationen.*

Ein *Objekt* im Sinne der Objektorientierung besteht aus einem Bündel von *Operationen* sowie möglicherweise aus einem *Zustand.* Eine Operation ist gegeben durch:

- den *Namen* der Operation,
- die *formale* Spezifikation: Zahl, Art und Gültigkeitsbedingungen für Parameter- und Antwortobjekte (s.u.),
- die *funktionale Spezifikation*: Was leistet die Operation für den Geschäftsbereich? Statt von funktionaler spricht man auch von *semantischer* Spezifikation, *semantischer Beschreibung* oder schlicht der *Semantik* der Operation.

Eine Operation kann im Kontext des Objektes *ausgeführt* werden. Die Ausführung erfolgt gemäß der funktionalen Spezifikation der Operation; sie kann sowohl vom augenblickli-

chen Zustand des Objektes als auch von zusätzlich angebenen weiteren Objekten, den *Parametern,* abhängen. Die Ausführung einer Operation eines Objektes wird angestoßen durch das *Empfangen* einer *Nachricht*, die von einem anderen Objekt oder einem Akteur *gesendet* wurde.

> Der Mechanismus der Nachrichtenübermittlung bleibt offen, er wird bei einer Realisierung des Systems festgelegt. Einige Möglichkeiten der Übertragung sind: Prozeduraufrufe innerhalb eines Betriebssystem-Prozesses, Kommunikation zwischen solchen Prozessen, Netzwerkkommunikationen beliebiger Art.

Der Inhalt der Nachricht besteht aus dem Namen der gewünschten Operation sowie eventuell den für diese Ausführung zu verwendenden Parametern. Während der Ausführung kann die Operation:

- den Objektzustand ändern,
- Nachrichten an andere Objekte senden,
- die empfangene Nachricht durch Übermittlung eines Objektes beantworten.

Der Zustand eines Objektes kann sich *ausschließlich* als Wirkung irgendeiner seiner Operationen ändern.

> Als rudimentäres Beispiel sei die funktionale Spezifikation einer Operation für *Fahrzeug*-Objekte betrachtet:
>
> > ***BerechneNeueGeschwindigkeit**:* Aus der bestehenden Geschwindigkeit des Fahrzeugs, der an der Position des Fahrzeugs geltenden Höchstgeschwindigkeit sowie der Geschwindigkeit des vorausfahrenden Fahrzeugs wird eine neue Geschwindigkeit berechnet nach der Formel ....
>
> Dieser semantischen Beschreibung ist zu entnehmen,
>
> - dass das Objekt *Fahrzeug* als Zustand u.A. eine *Position* und eine *Geschwindigkeit* hat,
> - dass die Geschwindigkeit von Fahrzeugen über eine Operation erfragbar sein muss,
> - dass die Position des Fahrzeugs als Parameter für die Bestimmung einer Höchstgeschwindigkeit durch eine Operation eines hier nicht näher bezeichneten Objekts fungieren muss,
> - dass sich der Zustand des Objektes durch die Operation ändern kann.

Die funktionale Spezifikation beschreibt u.A. alle vor, während und nach der Ausführung der Operation einzuhaltenden, vom Geschäftsbereich geforderten *Bedingungen* sowie die Reaktion auf die Verletzung solcher Bedingungen.

Nicht zur Definition einer Operation gehört, *wie* die funktionale Spezifikation erfüllt werden soll, also z.B. welches Verfahren zur Berechnung der „Formel“ in obigen Beispiel eingesetzt werden soll oder wie der Objektzustand strukturiert werden soll. Eine solche Angabe würde eine *Methode* definieren*.* Zu einer Operation können oft viele verschiedene, funktional gleichwertige Methoden angegeben werden.

> Ein weiteres Beispiel soll den Unterschied klarmachen: Wir betrachten *Dreiecks*-Objekte der euklidischen Ebene. Für ein derartiges Objekt kann man eine Operation *GibFlächeninhalt* fordern:
>
> > **GibFlächeninhalt**. Die Operation teilt den Flächeninhalt des Dreiecks mit.
>
> Bekanntlich gibt es dazu mehrere Verfahren, elementargeometrische wie z.B. direkt aus den Eckpunkt-Koordinaten, über die Heronische Formel, über Grundlinie und Höhe, und auch „esoterische“, wie z.B. Monte-Carlo-Integration. Wird die obige Spezifikation wie folgt geändert:

GibFlächeninhalt. Die Operation teilt den Flächeninhalt des Dreiecks mit, *berechnet aus der Heronischen Formel.*

liegt eine Methodenspezifikation vor.

Nun möchte man aber immer erreichen, dass sich eine Implementation *vertragsgemäß* verhält, und dazu muss es möglich sein, überprüfbare Qualitätsanforderungen in die funktionale Spezifikation einer Operation aufzunehmen:

GibFlächeninhalt. Die Operation teilt den *der Heronischen Formel entsprechenden* Flächeninhalt des Dreiecks mit.

ist keine Methodenspezifikation, denn sie gestattet ja den Einsatz eines beliebigen Verfahrens, sofern nur die Qualitätsanforderung erfüllt wird.

Die Formulierung

GibFlächeninhalt. Die Operation teilt den Flächeninhalt des Dreiecks mit*. Nach den „Regeln für die elektronische Dreiecksberechnung im Straßenbau“ ist die Heronische Formel zu verwenden.*

enthält eine Anforderung, die durch ein (hier fiktives) Regelwerk sanktioniert ist. Solche Operationsbeschreibungen müssen erlaubt sein, da sonst Regelkonformität nicht gewährleistet werden kann.

Das letzte Beispiel gibt Anlass zu der Regel:

Die funktionale Spezifikation einer Operation darf auch implementationsorientierte Verfahrensanweisungen (z.B. Algorithmen) fordern, sofern dies durch bindende Regelwerke des modellierten Geschäftsbereichs erzwungen wird.

Eine Operation, zu der keine Methode angegeben werden kann oder soll, heißt *abstrakte Operation*.

***Leitsatz**: Der objektorientierte OKSTRA definiert nur Operationen, keine Methoden, d.h. alle Operationen sind abstrakte Operationen.*

3.3 Identität, Lebenslauf und Instanzierung

***Leitsatz**: Jedes Objekt ist zu jeder Zeit von allen anderen verschieden. Es kann erzeugt und vernichtet werden und seinen Zustand verändern.*

Damit ein Objekt von anderen Objekten eine Nachricht empfangen kann, muss es identifizierbar sein. Jedes Objekt benötigt deshalb eine eindeutige, unveränderliche *Identität*. Zwei Objekte sind auch dann voneinander verschieden, wenn sie sich im gleichen Zustand befinden.

Für die allermeisten Anwendungen sind außerdem Systeme erforderlich, bei denen die benötigte Zahl von Objekten veränderlich ist. Das bedeutet, dass Objekte andere Objekte *erzeugen* und möglicherweise *vernichten* können müssen. So erhält jedes Objekt einen individuellen *Lebenslauf*, während dessen sich sein Zustand durch die Aktivität seiner Operationen ändern kann.

Wir betrachten eine Verkehrssimulation auf der Basis einzelner Fahrzeuge. Ein *Verkehrszellen*-Objekt erzeugt hierzu ein neues *Fahrt*-Objekt und speist es über eine Operation für *Abschnitts*-Objekte in einen Abschnitt der Verkehrszelle ein. Das Fahrtobjekt simuliert die Fahrt eines Fahrzeuges durch das Straßennetz. Während des Lebenslaufes des Fahrt-Objektes ändert sich die Zuordnung Fahrtobjekt-

Abschnitt, in dem an jedem passierten *Netzknoten* entschieden wird, welcher Abschnitt als nächstes zu befahren ist. Ist die simulierte Fahrt vorbei, wird das Objekt von dem Verkehrszellen-Objekt des zuletzt erreichten Abschnitts registriert und vernichtet.

Den Erzeugungsakt eines Objektes nennt man *Instanzierung*.

Die Identität eines Objektes wird nicht zu seinem Zustand gezählt. Operationen ändern die Identität eines Objekts nicht.

Die Zahl und Beschaffenheit der Operationen eines Objektes ändert sich nicht.

Wird die gleiche Nachricht an zwei Objekte unterschiedlicher Identität gesendet, die sich im gleichen Zustand befinden, so werden die gleichen Antworten erzeugt.

3.4 Klassen und Exemplare

Sieht man von der Identität der Objekte und ihrem Zustand ab (von dem wir nur wissen, dass er sich ändern kann und dass er Auswirkungen auf die Ergebnisse von Operationen haben kann), so bleibt als eigentlich kennzeichnendes Merkmal für Objekte nur die Menge der Operationen übrig, über die sie verfügen. Dies gibt Anlass zu der Definition:

***Leitsatz**: Die Spezifikation eines zusammengehörigen Satzes von Operationen für Objekte bildet eine Klasse.*

Die Klasse der *Fahrzeug*-Objekte des obigen Beispiels kann etwa (wieder als unvollständiges Beispiel!) spezifiziert werden durch die Operationen:

- *GibGeschwindigkeit*
- *BerechneNeueGeschwindigkeit*
- *GibFahrzeugAbstand*

Da individueller Zustand und Identität der Objekte Eigenschaften sind, die erst während der Aktivität eines Informationssystems eine Rolle spielen, wird besonders im Rahmen konzeptioneller Modelle oft kein Unterschied zwischen Objekt und Klasse gemacht. Dies ist auch der bisherige Sprachgebrauch für den OKSTRA (*Objekt*katalog für...!).

Eine Revision dieser Sprachgewohnheit ist kaum durchsetzbar und wird daher nicht angestrebt. Falls es notwendig ist, einzelne Objekte in ihrer Individualität zu betrachten, sollte konsequent von *Exemplaren* und von *(Objekt-)Klassen* statt von Objekten gesprochen werden und der Gebrauch des Begriffs Objekt ohne Qualifikation ganz vermieden werden.

3.5 Klassenoperationen

***Leitsatz**: Klassenoperationen sind Operationen, die unabhängig von Existenz, Identität und Zustand der Exemplare einer Klasse sind,*

Manchmal ist es notwendig, Operationen zu definieren

- die unabhängig von der Existenz von Objekten einer Klasse sind, oder
- die unabhängig vom Zustand eines Objektes sind.

Zweckmäßigerweise rechnet man solche Operationen der Klasse selbst zu und nicht ihren Objekten. Man nennt sie *Klassenoperationen* im Kontrast zu dem Normalfall der *Objektoperationen*.

Beispiele: Die Erzeugung eines neuen Objekts einer Klasse oder die Ermittlung der aktuellen Anzahl aller Objekte einer Klasse sind offenbar keinem einzelnen Objekt, sondern nur der Klasse zuzuordnen.

3.6 Werte und Datentypen

Intuitive Definitionen für diese beiden Begriffe werden im folgenden Abschnitt bei der Behandlung von Attributen gegeben.

***Leitsatz**: Ein Wert ist ein Exemplar, das keinen Zustand und unendliche Lebendauer hat.* (Dies ist die objektorientierte Definition von Konstanten.)

Beispiele sind die Zahl 5304, die Zeichenkette „OKSTRA“ oder die Koordinate (7°52´, 51°12´).

***Leitsatz**: Ein Datentyp ist eine Klasse für Werte.*

Beispiel: Positive ganze Zahlen. 13, 1604, 9999 sind Objekte dieses Datentyps, 0 ist es nicht.

Die Spezifikation eines Datentyps gibt Bedingungen für die Zugehörigkeit eines Wertes an. Wie andere Klassen auch, spezifiziert ein Datentyp Operationen, im Beispiel kämen dafür etwa Addition, Multiplikation und Vergleich in Frage. Operationen für einen Datentyp können die Exemplare des Datentyps (die Werte) nicht ändern.

3.7 Attribute und Assoziationen

Die „reine Lehre“ der Objektorientierung kommt ohne die aus der klassischen Datenbanktechnik stammenden Begriffe *Attribut* und *Relation* aus. Aus mehreren Gründen ist eine Betrachtung dieser Begriffe jedoch sinnvoll:

- um leichter eine systematische Strategie für die Übernahme des Modells des bisherigen Modells des OKSTRA aufzubauen,
- in Hinblick auf die in der Unified Modeling Language UML verwendete Terminologie,
- um Klarheit und Verständlichkeit der Modelle zu fördern, und
- für Realisierungen, die nicht nur durch Operationen beschrieben werden können, z.B. Festlegungen für Datenaustauschformate, sind Attribute nicht verzichtbar.

Es folgen zunächst intuitive Definitionen für beide Begriffe, die für die konzeptionelle Modellierung ausreichen, und danach präzisierte Fassungen, die sich für die Umsetzung innerhalb eines Informationssystems eignen

Ein *Attribut* ist eine Eigenschaft eines Objektes, die sich als Zahlenwert (z.B. *3,14*), qualitatives Merkmal (z.B. *grün*) oder Text (z.B. „*A. Stein*“) ausdrücken lässt. Die Unterscheidung zwischen diesen Möglichkeiten Zahl, Text usw. konstituiert den *Datentyp* des Attributs. Der *Wert* eines Attributs entstammt einem für den Datentyp charakteristischen zulässigen Bereich. Neben *einfachen* Datentypen sind auch *zusammengesetzte* möglich, z.B. Koordinaten aus *zwei* oder *drei* Zahlenwerten, oder *Datumsangaben* aus Tageszahl, Monatszahl und Jahreszahl. Attribute lassen sich *abfragen* und/oder *setzen*, so dass sich jedes Attribut als zusammengehörende Abfrage- und Setz-Operationen auffassen lässt.

Der Begriff Relation ist in seiner Verwendung in der IT-Welt vieldeutig, weswegen es sich anbietet, einen neuen, weniger vorbelasteten Begriff zu verwenden: *Assoziation.* Eine Assoziation zwischen zwei Objektklassen besteht darin, dass den Exemplaren der einen Klasse Exemplare der anderen Klasse zugeordnet werden, so dass jedes Exemplar einer

der beiden Klassen weiß, ob und welche der Exemplare der anderen Klasse seine Partner sind.

> So sind einem Abschnitt seine begrenzenden Netzknoten zugeordnet, und einem Netzknoten die von ihm ausgehenden Abschnitte.

Zuordnungen (auch als Verknüpfungen bezeichnet) zwischen Exemplaren lassen sich neu bilden, auflösen und abfragen, so dass sich auch Assoziationen durch Abfrage- und Setzoperationen in den beteiligten Klassen darstellen lassen.

Da mehrere Assoziationen zwischen zwei Klassen bestehen können und auch eine Assoziation einer Klasse mit sich selbst vorkommen kann, ist die Angabe der beteiligten Klassen nicht ausreichend. Es muss vielmehr gesagt und benannt werden, *als was* die Exemplare ihre Partner *betrachten*. Dies nennt man die *Rolle* einer Assoziation; es ist übrigens den NIAM-Bewanderten bekannt als Eintrag in einem Relations-Kästchen

> Zur Verdeutlichung wieder Beispiele:
>
> *Abschnitt_oder_Ast*-Objekte und *Brücken* können *zwei* Assoziationen bilden: Ein Abschnitt_oder_Ast *führt über* eine Brücke, und: Ein Abschnitt_oder_Ast *führt unter* einer Brücke hindurch.
>
> Ein Nullpunkt kann mit einem *Abschnitt_oder_Ast* in der Rolle *Anfangsnullpunkt* oder in der Rolle *Endnullpunkt* assoziert sein.

Nun zur Präzisierung dieser Begriffe:

***Leitsatz**: Attribute und Assoziationsrollen sind äquivalente Begriffe. Beide werden implizit als durch Akzessor-Operationen repräsentiert angesehen.*

Die beiden Begriffsbildungen Attribut und Relation werden auf den Operationsbegriff zurückgeführt.

Ein *Attribut* ist gegeben durch ein Paar von Operationen, den *Akzessoren*. Die eine, der *Observator,* nimmt keine Parameter an und erteilt einen *Wert* als *Antwort*. Zwei unmittelbar aufeinanderfolgende Ausführungen des Observators liefern stets die gleiche Antwort. Unmittelbar bedeutet, dass zwischen den beiden Operationsausführungen keine andere Nachricht das Objekt erreicht hat. Der mitgeteilte Wert liegt zudem garantiert im *Gültigkeitsbereich* des Attributs, das ist die Menge der zulässigen Attributwerte.

Die zweite Operation, der *Mutator*, verlangt als *Parameter* einen Wert; nach ihrer Ausführung teilt eine unmittelbar danach ausgeführte Observator-Operation wieder genau diesen Wert als Antwort mit, sofern der Wert im Gültigkeitsbereich des Attributs lag. Die Mutator-Operation darf fehlen, dann ist das Attribut *unveränderlich*.

Statt Observator und Mutator werden oft die Bezeichnungen Get-Set-Operationen bzw. eingedeutscht Gib-Setze-Operationen verwendet.

Für den Relationsbegriff sind die Verhältnisse etwas verwickelter. Es muss unterschieden werden zwischen

- der Relation
- dem Relationselement (auch Zuordnung oder Verknüpfung, engl. *link,* genannt)
- dem Schema für die Relation

Zunächst zum *Relationsbegriff*: Mathematisch ist eine zweistellige *Relation* zwischen zwei Mengen A und B eine Teilmenge der geordneten Paare (a,b) mit a aus A und b aus B. Ein

einzelnes Paar ist ein *Relationselement*. Für n-stellige Relationen sind die Relationselemente n-Tupel (a,b,c,d...) mit entsprechenden Mengen A,B,C,D...

> Der Inhalt einer Tabelle in einem relationalen Datenbanksystem (RDBMS) stellt damit eine n-stellige Relation dar, wenn die Tabelle n Spalten hat. Ein Relationselement ist eine Tabellenzeile.

Oft wird aber der Begriff Relation stattdessen für das hinter ihrer Bildung stehende *Konzept* gebraucht, so in der Sprechweise „die Relation: Abschnitt beginnt bei Nullpunkt". *Dieser* Begriff wird zweckmäßig anders benannt, nämlich als *Assoziation*, er drückt aus, dass Exemplare der teilnehmenden Klassen als Relationselemente in einer Relation vorkommen können. Die für Relationen übliche Terminologie, wie Stelligkeit (Zahl der beteiligten Klassen) und Kardinalität wird sinngemäß auf Assoziationen angewendet.

Eine Assoziation wird nun ebenfalls auf Akzessoren abgebildet. Zunächst gehören zu einer Assoziation *Assoziationsrollen*. Eine Assoziationsrolle benennt die Sicht, die die Objekte einer Klasse auf die Objekte einer assoziierten Klasse haben.

> Beispiel: Zwischen *Unternehmen* und *Personen* besteht eine Assoziation, die die Mitarbeit einer Person in einem Unternehmen ausdrückt. *Unternehmen*-Objekte sehen *Personen* über diese Assoziation in der Assoziationsrolle *Arbeitnehmer*, umgekehrt sehen *Personen Unternehmen* als *Arbeitgeber*.

Für die Operationalisierung von Assoziationen sind nur die Assoziationsrollen wichtig. Jede Assoziationsrolle wird analog zu einem Attribut als durch Akzessoren gegeben behandelt. Die für Attribute gegebene Definition von Observatoren und Mutatoren wird wörtlich auf Assoziationsrollen übertragen, jedoch sind statt Werten von Datentypen *Exemplare beliebiger Klassen* als Antwort bzw. Parameter zugelassen. Das bei den Attributen geforderte Verhalten bei unmittelbar aufeinander folgenden Ausführungen der Akzessor-Funktionen gilt auch hier. Der dabei verwendete Gleichheitsbegriff bezieht sich bei Assoziationrollen auf die Gleichheit der Objektidentitäten, nicht jedoch auf die der Objektzustände.

Da für Attribute Objektidentität und Wertgleichheit dasselbe sind, existiert in der operationalen Betrachtungsweise zwischen Attribut und Assoziationsrolle kein Unterschied.

Der hier definierte Attribut- bzw. Assoziationsrollenbegriff ist auch unter der Bezeichnung *abstraktes Attribut* anzutreffen.

In UML 2.0 wird er durch das Konstrukt der *property* abgedeckt, wobei im Gegensatz zu diesem Leitfaden allerdings keine implizite Abbildung auf Akzessor-Operationen gefordert wird, da viele Programmiersprachen solche *properties* von sich aus unterstützen.

3.8 Generalisierung und Spezialisierung

***Leitsatz**: Eine Klasse B ist eine Generalisierung einer Klasse A, wenn in allen Operationen, die einen Parameter der Klasse B verlangen, auch Exemplare der Klasse A für diesen Parameter zulässig sein sollen.*

Aus einer Klasse B kann eine neue Klasse A erzeugt werden, in dem z.B. zu den existierenden Operationen von B neue hinzu definiert werden. Die Objekte von A unterstützen dann sowohl die Operationen von B als auch die neu hinzu gekommenen. Aber es gibt auch andere Möglichkeiten. So können von den Exemplaren der abgeleiteten Klasse zusätzliche Eigenschaften gefordert werden.

> Beispiel: Kreise sind eine Spezialisierung von Ellipsen, in dem die Gleichheit der Halbachsen gefordert wird.

Man sagt dann, dass A eine aus B *abgeleitete Klasse* oder eine *Ableitung* oder *Spezialisierung* ist, und umgekehrt, dass B eine *Basisklasse* zu oder *Generalisierung* von A ist.

Offensichtlich können aus einer Klasse mehrere verschiedene Klassen abgeleitet werden, d.h. viele Klassen können dieselbe Basisklasse haben.

> *Radfahrer, Fußgänger, Kraftfahrzeuge, Straßenbahnen* etc. sind abgeleitete Klassen zur Basisklasse *Verkehrsteilnehmer*.

In Fällen wie in diesem Beispiel wird die Klasseneinteilung durch ein implizit vorhandenes Attribut erzeugt, das auf den Exemplaren der abgeleiteten Klassen einen konstanten Wert hat. Dies ist typisch für Begriffshierarchien (Taxonomien).

> Im Beispiel ist das implizite Attribut das Fortbewegungsmittel.

Ob solche Taxonomien im Modell in eine entsprechende Hierarchie abgeleiteter Klassen aufgelöst oder ob die Einteilung durch explizite Klassifizierungsattribute („Objektart“) vorgenommen werden kann, hängt davon ab, ob die Operationen für die verschiedenen Unterarten dieselben sein können oder nicht (was sich nicht nur auf Benennung und Parametrisierung, sondern auch auf die Funktion bezieht!).

Die Übernahme der Operationen einer Basisklasse durch ihre Ableitungen bezeichnet man als *Vererbung*.

In der Praxis hat man oft den umgekehrten Fall mehrerer existierender Klassen, bei denen sich Gemeinsamkeiten im Bestand der Operationen (Attribute, Assoziationen) ergeben. Diese Gemeinsamkeiten kann man dann in eine Basisklasse übertragen.

> So ergab sich im OKSTRA die Basisklasse Abschnitt_oder_Ast zu den Klassen Abschnitt und Ast.

Eine Ableitung kann auch aus *mehreren* existierenden Klassen gewonnen werden, indem man fordert, dass sie alle Operationen dieser Klassen zusammen (plus eventuell zusätzliche neue) unterstützt. Eine Klasse kann daher mehrere Basisklassen haben. In diesem Fall spricht man von *Mehrfachvererbung*.

Die Klassen eines Modells lassen sich bezüglich der Ableitungsbeziehung in einer *Klassenhierarchie* darstellen. Die Begriffe Basisklasse-Generalisierung und Ableitung-Spezialisierung werden dann auch auf Paare von Klassen angewendet, wenn es innerhalb der Hierarchie eine Folge von mehreren verbundenen Ableitungen zwischen den Klassen gibt. Will man besonders ausdrücken, dass eine Ableitung über Zwischenstufen erfolgte, so spricht man von *indirekter* Ableitung, Generalisierung usw

3.9 Polymorphie

***Leitsatz**: Operationen, die fachlich analoge Funktionen haben, heißen polymorph zueinander. Sie erhalten denselben Namen.*

Oft taucht das gleiche Konzept bei der Beschreibung unterschiedlicher Objekte auf.

> Als Beispiel sei etwa die *Länge* genannt, die sich z.B. für die Klassen *Abschnitt*, *Strecke*, *Route* des OKSTRA definieren ließe, und die für all diese Objekte deren Längenausdehnung mißt, wie sie z.B. für eine Längenstatistik gebraucht wird. Für alle genannten Objekte kann daher eine Operation *GibLänge*() eingeführt werden.

Zweckmäßig werden Operationen, die für Objekte verschiedener Klassen dasselbe leisten, gleich benannt und parametrisiert.

Solche .namensgleiche, fachlich analoge Operationen verschiedener Klassen nennt man *polymorph.*

Das Auftreten polymorpher Operationen zeigt an, dass die beteiligten Klassen eine Verwandschaft aufweisen, die eventuell zu einer neuen Abstraktion führt.

> Alle Objekte, für die die Operation *GibLänge* sinnvoll definiert werden kann, kann man als Mitglieder einer neuen abstrakten Klasse *LängenausgedehntesNetzobjekt* auffassen, der als typische Eigenschaft gerade diese Operation zugeordnet wird. Alle daraus abgeleiteten Klassen (z.B. Strecke, Route, Teilnetz) müssen dann diese Operation unterstützen.

Der Vorteil einer solchen Modellierung zeigt sich in folgendem Beispiel:

> In einem GIS soll die Länge eines per Mausklick ausgewählten Teiles des Straßennetzes angezeigt werden. Das Unterprogramm des GIS, das diese Aufgabe erledigen soll, muss dann nur das angeklickte Objekt daraufhin überprüfen, ob es ein *LängenausgedehntesNetzobjekt* ist.
>
> Ohne die Abstraktion *LängenausgedehntesNetzobjekt* müsste getestet werden, ob es sich um einen Abschnitt oder eine Strecke oder oder ... handelt. Und kommt eine neue Objektart mit Längenausdehnung hinzu, müsste auch das GIS-Programm geändert werden. Wird diese neue Klasse jedoch aus *LängenausgedehntesNetzobjekt* abgeleitet, ist nur dafür zu sorgen, dass dabei die *GibLänge*-Operation implementiert wird, und das GIS wird dann ohne Änderung auch für die neuen Objekte die Länge anzeigen.

Eine andere, verwandte Art von Polymorphie gestattet unterschiedliche Ausgangsbedingungen für ansonsten fachlich analoge Operationen.

> Beispiel:
>
> ErzeugeAbschnitt(N1,N2) erzeugt einen neuen Abschnitt zu zwei Netzknoten N1 und N2.
>
> ErzeugeAbschnitt(A1,A2) erzeugt einen neuen Abschnitt durch Verschmelzung der Abschnitte A1 und A2.

3.10 Metaklassen

***Leitsatz**: Eine Metaklasse ist eine Klasse, deren Exemplare selbst Klassen sind.*

Hin und wieder ist es zweckmäßig, Klassen selbst als Objekte zu betrachten, z.B. um sie als Parameter für oder als Antworten auf Nachrichten versenden zu können. In solchen Fällen können Klassen dann als Exemplare einer *Metaklasse* aufgefasst werden. Beispiele finden sich unten im Kapitel über die OKSTRA-Rahmenmodelle bei den Ereignissen und beim Typsystem.

> Den Unterschied zwischen Basisklassen und Metaklassen machen wir noch an einem Alltagsbeispiel klar.
>
> Alle Pkws gruppieren wir nach ihrer Marke in Klassen, alle BMWs kommen also in die Klasse *BMW-Pkw*, alle Renaults in die Klasse *Renault-Pkw,* usw. Zu diesen Klassen gibt es offensichtlich eine Basisklasse, nämlich *Pkw*, die von den Unterschieden der Fahrzeuge einzelner Marken abstrahiert. Das Auto, das gerade an uns vorbeifuhr, identifizieren wir als *BMW-Pkw,* und als solches *ist es* ein *Pkw.*
>
> Andererseits können wir die Automarken als Instanzen einer Klasse *Marke* begreifen. Da wir die Automarken jedoch selbst als Klassen modelliert haben, ist *Marke* eine *Klasse von Klassen*, eben eine *Metaklasse.* Von dem vorbeifahrenden Auto können wir *nicht* sagen, dass es eine Marke *ist.* Die Metaklasse Marke verwenden wir z.B., um Markenembleme zu modellieren. Dazu bekommt *Marke* ein Attribut *Emblem*. Jede Klasse belegt dieses Attribut mit einem *einzigen* Wert, eben dem für BMW, Re-

nault usw. typischen Emblem. Alle Pkws bekommen damit genau das ihrer Marke entsprechende Emblem zugeteilt.

3.11 Typen und Schnittstellen

In vielen Programmentwicklungssystemen werden neben dem Begriff Klasse die Begriffe *Typ* und *Interface* verwendet.

***Leitsatz**: Für den OKSTRA sind die Begriffe Klasse und Typ synonym, die Klassen des OKSTRA sind allesamt Typen.*

Eine *Schnittstelle* (ein *Interface)* ist eine Klasse, zu der keine eigenen Exemplare gebildet werden können, sondern nur zu daraus abgeleiteten Klassen (wie z.B. zu *Abschnitt_oder_Ast* nur Exemplare als *Abschnitt* oder aber als *Ast* existieren können).

Eine genauere Erklärung dieser Sachverhalte und einige weitere Sprechweisen folgt für Interessierte.

Ein *Typ* ist gegeben durch eine Klasse, die nur Operationen, Attribute und Assoziationen spezifiziert, die von Objekten anderer Klassen verwendet werden können (sog. sichtbare Operationen usw.). Im Allgemeinen dürfen Modelle zusätzlich Auskunft über *innere* Zustände und Operationen von Klassen geben, also solche, die nur den Methoden der Klasse zugänglich sind. Für ausführungsreife Software-Baupläne ist das auch nötig.

Für den OKSTRA wird hingegen hier vereinbart, dass Attribute und Assoziationen *abstrakt* sind, d.h. durch Akzessor-Operationen repräsentiert werden. Zudem werden keine Methoden angegeben, damit entfällt die Notwendigkeit, die innere Struktur einer Klasse zu beschreiben. Also erfüllen alle Klassen des OKSTRA die Bedingung für Typen, woraus sich der obige Leitsatz ergibt.

Weitere Regeln und Sprechweisen für Typen:

Typen dürfen nur von Typen abgeleitet werden.

Ein Exemplar ist *vom Typ* T oder *gehört zum Typ* T, wenn es alle Operationen, die T spezifiziert, unterstützt.

Beispiel: Wir definieren (s. Beispiel in 3.9) einen Typ *LängenausgedehnteNetzobjekte* mit einer Operation G*ibLänge()*. Wenn nun die Objekte der Klassen *Straße, Abschnitt* und *Ast* alle die Fähigkeit bekommen sollen, ihre Länge mitzuteilen, leiten wir sie von diesem Typ ab. Bei *Abschnitt* und *Ast* geschieht dies indirekt über den Typ *Abschnitt_oder_ Ast*.

Ein *Subtyp* eines Typen T ist ein Typ, der aus T (indirekt) abgeleitet wurde.

Ein Typ heißt *instanzierbar*, wenn zu ihm Exemplare gebildet werden können, die zu keinem seiner Subtypen, folglich nur zu ihm selbst, gehören. OKSTRA-Modelle müssen die Instanzierbarkeit der in ihnen auftauchenden Typen angeben.

Abschnitt_oder_Ast ist nicht instanzierbar, *Abschnitt* und *Ast* sind es.

Leitsatz: *Schnittstelle bzw. Interface* ist gleichbedeutend ist mit *nicht instanzierbarer Typ.*

Der Begriff *Interface* wird in den relevanten IT-Standards unterschiedlich definiert.

Der UML-Standard 1.4 definiert *Interfaces* als abstrakte Klassen, die nur abstrakte Operationen definieren. Attribute und Assoziationen sind für Interfaces nicht zugelassen.

Der IDL-Standard (ISO/IEC 14750) für die verteilte Systemarchitektur CORBA setzt Interfaces aus Datentypen, Operationen und Attributen zusammen, wobei Attribute implizit Akzessor-Operationen definieren.

Für den objektorientierten OKSTRA wurde die Typ-Definition des UML-Standard 1.4 durch die Attribut-Interpretation im Interface-Begriff von IDL ergänzt und zum hier gegebenen Typ-Begriff zusammengeführt.

Der so eingeführte Typbegriff ist das angemessene Modellierungskonstrukt für den objektorientierten OKSTRA:

- Er gestattet eine Modellierung mit Hilfe von Attributen und Assoziationen.
- Er ist mit dem Subtype/Supertype-Konstrukt aus EXPRESS kompatibel.
- Durch die vollständige Operationalisierung von Attributen und Assoziationen ist die Unabhängigkeit von Implementierungstechniken gewährleistet.
- In der Praxis werden die Begriffe *Typ* und *Interface* fast immer als synonym betrachtet, so dass eine einheitliche Definition naheliegt.
- Die Berücksichtigung des IDL–Standards gestattet es, für den objektorientierten OKSTRA wo nötig auch IDL-Beschreibungen zu erstellen. Dies ist z.B. wichtig, wenn das XML-Modell des OKSTRA um eine entsprechende Definition für dazu passende DOM-Schnittstellen ergänzt werden soll. Diese sind laut DOM-Spezifikation in IDL zu formulieren. (DOM: Domain Object Model).
- Der kommende Standard UML 2.0 lässt für Interfaces *properties* zu, die zu Attributen bzw. Assoziationsrollen gehören können.

3.12 Anwendungen, Dienste und Komponenten

Anwendungen sind IT-Systeme, um eine Anzahl fachlich zusammenhängender Anwendungsfälle zu bearbeiten. Eine Anwendung trennt üblicherweise

- Informationsdarstellung (Präsentation) und
- Informationsbereitstellung (Geschäftslogik)

voneinander. Das gewährleistet, dass dieselbe Information für den Nutzer unterschiedlich ausgegeben werden kann – z.B. als Liste, Diagramm, thematische Landkarte, Exportdatei.

Die Informationsbereitstellung wird häufig in kleinere Systemeinheiten zerlegt, die hochradig fachspezifisch arbeiten können oder aber sehr allgemeine Aufgaben wahrnehmen (wie z.B. Datenbanken). Eine solche Zerlegung bietet sich insbesondere dann an, wenn solche Systemeinheiten von mehreren Anwendungen genutzt werden können. Die Systemeinheiten zur Informationsbereitstellung nennt dieser Leitfaden *Dienste*.

***Leitsatz**: Dienste unterstützen Geschäftsprozesse, in dem sie Funktionen für zusammengehörige fachliche Anwendungsfälle über Schnittstellen zugänglich machen.*

Dienste sind Systemeinheiten, die intern aus Klassen aufgebaut sind, die zusammenarbeiten, um Anwendungsfälle für eine eng durch fachlichen Zusammenhang begrenzte Zahl von Objektklassen zu realisieren. Dienste können selbst andere Dienste nutzen, sie kommunizieren dazu untereinander über Schnittstellen. Wichtige Designkriterien für die Definition von Diensten sind:

- Wiederverwendbarkeit in verschiedenen Prozessen
- Reduzierung der Zahl von Kommunikationskanälen
- Unterstützung der Trennung von Inhalt und Darstellung von Informationen

Die Modellierung zur Kostenermittlung ergab einen Satz von 5 Klassen, die für die eigentliche Aufgabe *Kostenberechnung* eingesetzt werden. Diese bilden den zum Anwendungsfall *Kostenberechnung* gehörenden Dienst.

Die Bündelung von Programmcode für Klassen, die zusammenarbeiten, führt zu *Komponenten*.

Die Klassen zur Kostenberechnung nach der AKS wurden in 3 Komponenten zusammengefasst: OO-AKS-Manager zur Durchführung und Speicherung der Kostenberechnungen, OO-KBK-Prototyp zur Pflege des zugrundezulegenden Kostenberechnungskataloges, und OO-KBK-Preisdoku zur Pflege einer Preisdokumentation.

Eine Komponente kann mehrere Dienste zur Verfügung stellen. Während Dienste fachlich motivierte Zusammenstellungen von Klassen sind und somit einen Bezug zu den Geschäftsprozessen haben, für die sie ihre Dienstleistung erbringen, werden Komponenten häufig nach pragmatischen Kriterien zusammengestellt, z.B. leichter Austausch von Versionen, physische Konzentration usw.

Ein Handy ist eine Komponente mit Diensten zum Telefonieren, Adressbuchverwaltung, Terminkalender, Taschenrechner, Spielesammlung, Internetbrowser usw.

4 Bildung der Objektwelt

Dieses Kapitel gibt zusätzliche Hinweise zur Auffindung und Ausstattung von Klassen, Akteuren und Operationen in den Modellierungsschritten (2.5) und (2.6). Das Rohmaterial sind (s. 2.2.4):

- Anwendungsfallbeschreibungen (Szenarien), z.B. aus Interviews zur Praxis, Dienstanweisungen usw.
- Texte von Regelwerken, z.B. Gesetze, Normen, Richtlinien, Verordnungen usw.
- Beispiel von Arbeitsergebnissen

Aus der Analyse dieser Quellen ergeben sich Begriffe für die Dinge, die im untersuchten Geschäftsbereich vorkommen und in das Modell eingehen sollen. Im Folgenden werden diese Begriffe als Konstrukt bezeichnet.

Es werden drei Bildungsansätze vorgestellt, die sich keineswegs ausschließen, sondern ergänzen.

4.1 Objektbildung nach pragmatischen Kriterien

Es werden hier pragmatische Kriterien zusammengestellt, wann ein Konstrukt aus einer Tätigkeitsbeschreibung oder Handlungsanweisung zu einer *Klasse* werden sollte (es müssen nicht alle erfüllt sein):

(1) Das Konstrukt ist *instanzierbar* und die *Exemplare* können eindeutig *identifiziert* werden. So ist z.B. das Konstrukt *Radfahrer* instanzierbar (die Exemplare sind die Personen, wenn sie mit dem Rad unterwegs sind). Eindeutig identifiziert werden können sie z.B. durch ihren Namen und ihre Anschrift (so hofft man wenigstens). Exemplare werden auch *Objekte* genannt.

(2) Die Exemplare des Konstrukts haben einen *Lebenslauf*. Ein *Radfahrer*-Objekt entsteht, wenn eine Person ihr Fahrrad besteigt, und vergeht, wenn sie absteigt. Konzeptionell ist damit ein Radfahrer z.B. etwas anderes als ein *Fahrradbesitzer.* Ein solches Objekt entsteht, wenn eine Person ein Fahrrad kauft (oder klaut), und vergeht, wenn sie das Rad irgendwie wieder los wird.

(3) Das Konstrukt ist Mitglied einer *hierarchischen Begriffsordnung (Taxonomie)*. Radfahrer, Fußgänger, Kraftfahrzeuge, Straßenbahnen etc. sind Unterarten (sogenannte abgeleitete Klassen) zum Konstrukt *Verkehrsteilnehmer*. *Kraftfahrzeuge* können weiter in *Pkw, Lkw, Busse, Motorräder* usw. untergliedert werden. Weitere typische Fälle sind Organisationseinheiten und Verwaltungsgliederungen.

(4) Das Konstrukt steht in einer *Bestandteil-Beziehung (merologische Relation)* zu einem anderen Konstrukt. Die Beziehung braucht nicht ständig zu bestehen. Eine *Straßenbahn* als *Verkehrsteilnehmer* besteht z.B. aus *Fahrzeug*, *Fahrer/Fahrerin* und den *Passagieren*.

(5) Das Konstrukt zeigt *dynamisches Verhalten.* Es hat Zustände, die sich im Lauf der Zeit ändern und an Bedingungen gebunden sein können, und es sind Operationen auf dem Konstrukt definierbar. Unsere Straßenbahn z.B. *hält* oder *fährt.* Beim *Aussteigen* wird ein *Passagier* vernichtet und ein *Fußgänger* entsteht. Das *Aussteigen* ist außerdem nur möglich, wenn die *Straßenbahn hält*.

(6) Das Konstrukt charakterisiert eine bestimmten *Typus*. Um das klarzumachen, betrachte man den *Verkehrsteilnehmer Bus*. Davon bilde man die zwei Ableitungen *Linienbus* und *Reisebus*. *Straßenbahnen* und *Linienbusse* sind nun zusätzlich dadurch charakterisiert, dass sie vom Typus *öffentliche Verkehrsmittel* sind. Eine charakteristische Eigenschaft des Typus ist, dass die darunter fallenden Objekte an ausgezeichneten Punkten, den *Haltestellen, halten*, um *Passagiere ein-* und *aussteigen* zu lassen.

(7) Das Konstrukt ist *in Raum und/oder Zeit lokalisierbar*. Eine Haltestelle *steht* an einem bestimmten Ort, und eine Fahrt *beginnt* und *endet* zu einer im Fahrplan festgelegten Zeit

Nach diesem Kriterium sind natürlich auch *Verspätungen* Objekte ☺.

(8) Das Konstrukt steht in *topologischer Beziehung* zu anderen Konstrukten. Ein *Kraftfahrzeug* steht auf einem *Parkplatz*.

(9) Das Konstrukt erfüllt eine *definierte Aufgabe in einem Geschäftsprozess*. *Öffentliche Verkehrsmittel* gehören zu ÖPNV-Betrieben, deren *Verkehrsleitung* u.a. die Beziehung zwischen *Fahrer/Fahrerin* und *Fahrzeug* herstellt und festlegt, wann ein *öffentliches Verkehrsmittel* an einer *Haltestelle halten* soll. Die Bildung solcher Klassen wird im nächsten Abschnitt genauer erläutert.

4.2 Objektbildung nach Aufgaben im Geschäftsprozess

Eine allgemein anerkannte Definition, was Geschäftsobjekte sind und welche Unterarten es davon gibt, existiert, trotz mannigfacher Bemühungen, bis heute nicht (wie eine selbst flüchtige Suche im Internet zeigt). Die hier vorgeschlagenen Kategorien sind ein Versuch. Es sind Fälle denkbar, in denen Konstrukte je nach Sichtweise sowohl der einen als auch der anderen Kategorie zurechenbar sind. Definitionen für *business entity object* und andere Geschäftsobjektarten finden sich reichlich im Internet, sie können über eine Suche nach dem Begriff schnell erschlossen werden.

(1) Ein Konstrukt repräsentiert einen *Gegenstand,* der zur Durchführung eines *Geschäftsprozesses* als *Ressource* benötigt wird, oder von ihm als *Produkt* hergestellt oder verändert wird. Solche Objekte werden hier mit dem Begriff *Fachobjekt* bezeichnet. Die Definition entspricht am ehesten den gängigen von *business entity object*.
Beispiele: Die meisten Klassen des bestehenden OKSTRA gehören hierzu, z.B. *Trasse* (wird in der Planung hergestellt), *Abschnitt* (wird bei der Bestandspflege angelegt und verändert), *Zählstelle* (wird als Datenlieferant für Verkehrsuntersuchungen etc. benötigt).

(2) Ein Konstrukt repräsentiert einen *Vorgang* während des Geschäftsablaufes. Diese Objekte *werden Prozessobjekte* genannt. Die Definition entspricht der von *business process object*. Wichtig ist, das im Konstrukt die zeitliche Verfolgung des Ablaufes in irgendeiner Form modelliert wird.
Beispiele: *Maßnahme* und als Teile davon z.B. *Untersuchung*, *Ausschreibung* etc.

(3) Ein Konstrukt repräsentiert ein *Signal*, das einen Prozess von außen beeinflußt und *Zustandsänderungen* in Prozess- oder Fachobjekten initiiert. Solche Objekte heißen *Ereignisobjekte*. (*business event objects*). Ein Ereignisobjekt ist immer zeitgebunden, da ein Ereignis definitionsgemäß zu einem bestimmten Zeitpunkt eintritt. Zumeist wird neben dem Zeitpunkt auch übermittelt, wer oder was das Ereignis ausgelöst hat.

Beispiele: *PlanfeststellungsverfahrenEingeleitet, AngebotEingegangen, ZahlungErfolgt, FürDenVerkehrFreigegeben*

(4) Ein Konstrukt repräsentiert ein *Schema* oder *Verfahren*, nach dem ein Prozess abläuft. Diese Objekte sollen *Regelungsobjekte* heißen (*business rule objects,* die Übersetzung Regelobjekt wurde wegen möglicher Missverständnisse verworfen). Mit Hilfe von Regelungsobjekten kann der Ablauf eines Prozesses geplant und konfiguriert werden.
Beispiele: *REB-Verfahren, Kostenberechnungskatalog, Zeichenvorschrift*

Für den oben genannten Fall einer Überschneidung der Zuordnung diene die Klasse *Maßnahme*. Während der Bedarfsplanung werden *Maßnahmen* definiert und sind daher als Produkt des Prozesses *Bedarfsplanung* Fachobjekte. Wird die Maßnahme dann tatsächlich in Angriff genommen, handelt es sich in den Prozessen der *Bauplanung* und *Baudurchführung* eindeutig um ein Prozessobjekt.

4.3 Objektbildung nach Kriterien der Systemfunktion

Die folgenden Kriterien gehen von der Funktion aus, die Objekte innerhalb des Gesamtsystems in Bezug auf andere Objekte haben können. Die Beispiele sind bekannten Betriebssystemfunktionen entnommen. Objekte mit:

(1) *Fabrikfunktion* stellen Objekte anderer Klassen her oder bauen sie um (z.B. Editor),

(2) *Vermittlerfunktion* organisieren die Übermittlung von Nachrichten zwischen Objekten (z.B. Verbindung, Mailbox),

(3) *Verzeichnisfunktion* erlauben die Suche nach anderen Objekten (z.B. Ordner, Katalog),

(4) *Schemafunktion* beschreiben den Aufbau und die Beziehungen von Klassen (z.B. Tabellenschema),

(5) *Überwachungsfunktion* gestatten oder verhindern den Zugriff auf andere Objekte (z.B. Berechtigung).

4.4 Häufig anzutreffende Objektstrukturen

Dieser Abschnitt stellt einige typische Strukturen vor, die zur Bildung von Klassen führen. Die Beispiele sind dem Bereich ÖPNV entnommen.

(1) Organisationseinheiten und –hierarchien (oft mit Raumbezug in Form von Zuständigkeitsbereichen kombiniert)
Abteilungen, Verkehrsverbundräume, Tarifzonen

(2) Netze
Linienwege, Umstiegsstellen, Liniennetz

(3) Arbeitsabläufe und ihre Schritte
Fahrkartenverkauf, Fahrzeugwartung

(4) Beobachtungen und Messungen
Fahrgastzahlen, Umsätze

(5) Dokumente und Dokumentverweise (Zitate)
Fahrkarten, Liniennetzpläne

(6) Kataloge und ihre Einträge

Fahrpläne

(7) Vorkommnisse

Halt an einer Haltestelle

4.5 Hinweise zur Ermittlung der Operationen

Die Zuweisung von Operationen zu den Klassen ist eine Aufgabe, die nicht zwingend zu einem einzigen gültigen Design führt.

Grundprinzip ist die Vereinbarung von *Verantwortlichkeiten* für jeden Typ. Diese Vereinbarung ist von der Aufgabe abhängig, die das System hat, das modelliert wird.

Beispiele: Der Typ *AKS_Fachobjekt* ist verantwortlich dafür, dass Fachobjekte Auskunft über die zur AKS-Kostenberechnung benötigten Mengen geben können.

Der Typ AKS_Preisdoku ist dafür verantwortlich, die Pflege und Befragung einer Preisdokumentation zu ermöglichen. Es sollen mengenabhängige Preise möglich sein, und sie müssen den Positionen des KBK zuzuordnen sein.

Ein Verkehrsteilnehmer-Objekt in einer Simulation. Es vermag jederzeit Auskunft über seinen Ort und seine Geschwindigkeit zu geben. Es reagiert auf Mitteilung von Geschwindigkeitsbeschränkungen, Mitteilung neuer Wetterdaten und Annäherung anderer Verkehrsteilnehmer.

Die Zerlegung eines Arbeitsablaufs in einzelne Schritte in einem Szenario hilft, die Verantwortlichkeiten und Operationen zu finden. Hier ein Beispiel:

Um die Einzelpositionen der AKS-Berechnung zu füllen, kann man sich entweder jede Position vornehmen und alle zugehörigen Fachobjekte raussuchen, oder man nimmt sich alle Fachobjekte nacheinander vor und schaut nach, zu welcher Position jedes gehört. Das erste Verfahren ist sehr aufwändig, weil man für jede Position jedesmal alle Objekte betrachten muss. Dieser Nachteil entfällt beim anderen Weg. Unser Szenario beginnt also:

Für jedes Fachobjekt tue folgendes:

Befrage das Fachobjekt, zu welcher Position es gehört.

Nun brauchen wir die Menge zur Position:

Befrage Fachobjekt, welche Menge es zur Position beisteuert.

Aggregiere erhaltene Menge.

Sind wir mit den Fachobjekten fertig, brauchen wir die Preise für die Positionen:

Für jede Position tue folgendes:

Befrage die Preisdokumentation nach dem Einheitspreis, der für die errechnete Gesamtmenge gilt.

Multipliziere Gesamtmenge mit Einheitspreis.

Es werden also notwendig:

- eine Operation für Fachobjekte, die die betroffene Position mitteilt
- eine Operation für Fachobjekte, die die Menge zur Position mitteilt
- eine Operation zur Preisdokumentation, die einen Einheitspreis zu einer Menge mitteilt
- eine Operation für Ansammlungen von Fachobjekten, die alle Fachobjekte der Ansammlung nacheinander aufzählt

- eine Operation für Ansammlungen von Positionen, die alle Positionen nacheinander aufzählt

Szenarien können auch in Form von UML-Sequenzdiagrammen formuliert werden.

Bei der Organisation der Klassenbildung hilft oft das CRC-Verfahren (Class-Responsibility-Collaboration):

Auf eine Karteikarte wird oben die Bezeichnung des Objektes (= Name der Klasse) geschrieben. Beim Durchgehen durch die Arbeitsabläufe wird für jede Anforderung ein Verantwortlichkeitseintrag links gemacht und ein Kollaborationseintrag (Klassenname) rechts.

In unserem vorigen Beispiel sieht das so aus (links Verantwortlichkeiten, rechts „Kollaborateure“):

Position	
Rechnet Preis zur Position aus	Fachobjekt Preisdoku Ansammlung

Fachobjekt	
Kann KBK-Nummer angeben, zu der es beiträgt. Kann Menge zur KBK-Nummer angeben	

Preisdoku	
Kann Einheitspreis zu Menge angeben	

Ansammlung	
Kann Fachobjekte oder aber Positionen ansammeln Kann angesammelte Elemente aufzählen	

5 Dokumentation mit UML

Dieses Kapitel beschreibt, wie die Modelle für den objektorientierten OKSTRA mit Hilfe der Unified Modeling Language UML dokumentiert werden sollen.

5.1 Einführung

Hier ist zunächst der Ort, die Unterschiede der Modellierungssprachen NIAM(ORM) und UML zu diskutieren, die jeweiligen Vorzüge und Nachteile anzuführen und die Entscheidung für UML zu begründen.

NIAM versteht sich als Sprache für die *konzeptionelle Datenmodellierung* in einer Zusammenarbeit von Fachleuten eines Anwendungsbereichs und Modellierern. Wie auch die Arbeit am bestehenden OKSTRA-Modell gezeigt hat, ist die NIAM-Notation für diese Aufgabe sehr geeignet.

UML hat den Anspruch, eine universelle Modellierungssprache sowohl für die Abbildung von Prozessen, die Datenmodellierung und den objektorientierten Entwurf zu sein. Diesem Anspruch wird sie auch gerecht, allerdings mit unterschiedlichem Erfolg. Für die konzeptionelle Datenmodellierung bietet NIAM eine wesentlich übersichtlichere und intuitivere Terminologie und Notation als UML.

Für den objektorientierten Entwurf dagegen müsste man die NIAM-Notation schon arg vergewaltigen, und dabei würden Übersichtlichkeit und Verständlichkeit verlorengehen. Insbesondere Operationen sind in NIAM mit den vorhandenen Modellkonstrukten kaum darstellbar. Die Darstellung von Operationen gelingt in UML besser, ist aber auch nicht frei von Schwächen, so würde man sich eine grafische Umsetzung des Operationskonstruktes wünschen und bessere Mittel zur verbalen Darstellung der Semantik der Operationen. Die Umsetzung objektorientierter Modellkonstrukte in eine intuitive Notation ist in den UML-*Klassendiagrammen* durchaus als halbherzig zu bezeichnen.

Die Abbildung von Prozessen gelingt in UML dagegen leicht. *Anwendungs*- und *Aktivitätsdiagramme* haben einen hohen Anschaulichkeitswert und sind für die kooperative Modellierung zwischen IT- und Fachbereichsexperten ähnlich gut geeignet wie NIAM-Diagramme. *Sequenzdiagramme* sind zwar deutlich abstrakter, erfüllen ihren Zweck, die Abfolge von Nachrichten zwischen Objekten exemplarisch darzustellen, dennoch relativ gut, insbesondere, wenn sie nicht zu groß werden.

UML versucht, ein Modell durch verschiedene Sichten zu beschreiben und verwendet dafür verschiedene Diagrammtypen. Die Existenz verschiedener Diagrammtypen macht die Notation umfangreicher, das Erlernen dauert länger. Der Ausdruck des Zusammenhanges der durch die Diagramme ausgedrückten Sichten ist zudem auch nicht immer intuitiv.

Dass die UML trotz einiger Schwächen dennoch als Modellierungssprache verwendet werden soll, hat folgende Gründe:

Erstens ist der Ansatz, möglichst viele Modellaspekte in *einer* Sprache zu bündeln, auch dann sinnvoll, wenn der Zusammenhang der Aspekte zz. nicht immer gelungen präsentiert wird. Er ist einer Methodik vorzuziehen, die die Aspekte eines Modells verschiedenen Darstellungsweisen überlässt, die *ohne jeden formalen Bezug* nebeneinander verwendet werden.

Zweitens ist die UML ein *internationaler Standard*. Es gibt daher sehr viele Werkzeuge, Bücher und Internet-Ressourcen dazu. UML wird an Universitäten und Fachhochschulen

gelehrt und auf breiter Front in der Softwareindustrie eingesetzt. Die praktische Umsetzung der OKSTRA-Modelle wird daher durch UML sehr erleichtert.

Drittens lässt der UML-Standard sich durch eigene, der Modellierungsaufgabe angepasste Konstrukte erweitern, und diese Erweiterungen der Modellierungssprache lassen sich zusammen mit dem Modell standardisiert übertragen.

Mit der kommenden Version 2.0 des UML-Standards wird außerdem die Modellierung von Geschäftsprozessen erheblich besser unterstützt.

Dieses Kapitel ist keine vollständige Einführung in UML. Der beschriebene Teil von UML enthält das, was zum Verstehen konzeptioneller Modelle des OKSTRA durch Fachleute notwendig ist, die keinen professionellen Hintergrund in Modellierung und Analyse von Geschäftsabläufen haben.

Als weiterführende Literatur wird besonders hingewiesen auf [Oes 03] und [Oes 04].

5.2 Allgemeine Grundlagen

Modellelemente sind die Einheiten, aus denen Modelle aufgebaut werden. Jedes Modellelement gehört einem Modellelementtyp an; statt Modellelementtyp verwendet dieser Leitfaden den Begriff *Modellkonstrukt*. Modellkonstrukte von NIAM sind z.B. Entitäten (dargestellt durch Ellipsen), Rollen (dargestellt durch ein Kästchen) und Relationen (aneinandergeklebte Kästchen mit Verbindungen zu Entitäten).

UML ist wie NIAM eine grafische Notation. Alle zentralen Modellkonstrukte haben eine grafische Darstellung. Der UML-Standard definiert für alle Modellkonstrukte die grafische oder textuelle Syntax, das Zusammenwirken der Konstrukte und die Bedeutung, die sie für den Aufbau eines Systems haben, das nach einem UML-Bauplan gebaut wird.

UML gestattet die Modellierung eines Systems aus verschiedenen Sichten heraus. *Strukturelle* Sichten beschreiben den Aufbau des modellierten Systems aus Bausteinen und wie diese Bausteine miteinander verbunden sind; *verhaltensorientierte* Sichten beschreiben die Systemabläufe, den Lebenslauf der Bausteine und den Austausch von Nachrichten zwischen ihnen.

UML besitzt deshalb viel mehr Modellkonstrukte als NIAM. Dies führt dazu, dass die „Lernkurve“ deutlich steiler ist. Um diesen Effekt abzumildern, ist es sinnvoll, aus dieser Reichhaltigkeit eine Auswahl von Modellkonstrukten zu treffen, die als unentbehrlich angesehen werden.

Grundlage für diese Auswahl sind die Anforderungen aus dem Modellierungsprozess nach Kapitel 2, die Definitionen des Kapitels 3, und die benötigten Modellarten nach Kapitel 6.

UML-Syntax wird nicht streng formal angeschrieben. Als einzige Zeichen zur Definition von Syntax werden eckige Klammern [ ] benutzt, um zu kennzeichnen, dass der eingeschlossene Bestandteil ausfallen kann. Beispiele für Syntax ist in „“ eingeschlossen.

Die Diagramme sind durch *Muster* illustriert, bei denen die Namen dazu benutzt werden, die Bezeichnungen bzw. Bedeutungen der Modellkonstrukte anzuzeigen.

Beispiele für Diagramme finden sich in den Anhängen.

5.2.1 Beziehungen

Es werden die in UML definierten Beziehungen in ihrer unveränderten Bedeutung benutzt:

Assoziationen.

Dependenzen. Ein- oder zweiseitige Beziehung zwischen Elementen. Eine Änderung an dem einen Element *kann* sich auf das andere auswirken.

Generalisierung und ***Spezialisierung.*** Aus einem Basiselement wird durch zusätzliche Eigenschaften ein *spezialisiertes (abgeleitetes)* Element. Generalisierung ist die Umkehrung: durch Weglassen von Eigenschaften entsteht ein allgemeiner verwendbares Element.

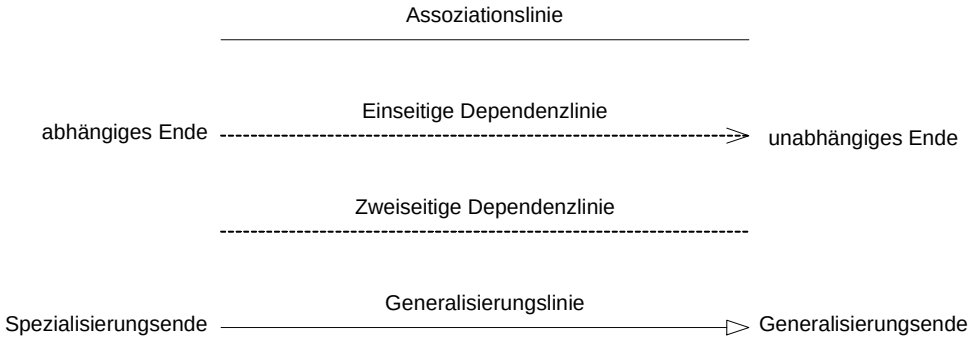


Abbildung 3. Beziehungslinien in UML.

Daneben gibt es eine als *Realisierung* bezeichnete Beziehung zwischen einem *spezifizierenden* und einem *realisierenden* Element. Das realisierende Element sichert zu, die in der Spezifikation verlangten Eigenschaften aufzuweisen. Diese Beziehung wird nicht benötigt. (Ihre Notation: wie Generalisierung, aber mit gestrichelter Linie)

Für die Beziehungen lassen sich *Sprechweisen* finden, die einem die Modellzusammenhänge leichter klar machen. Diese Sprechweisen sind meist diagrammabhängig und werden deshalb im Folgenden bei der Beschreibung der Diagrammtypen gegeben.

Für die Generalisierungsbeziehung kann man in der Regel die Sprechweise ***„ist ein“*** oder ***„ist eine Art von“*** verwenden. Hierzu einige Beispiele auf der folgenden Seite.

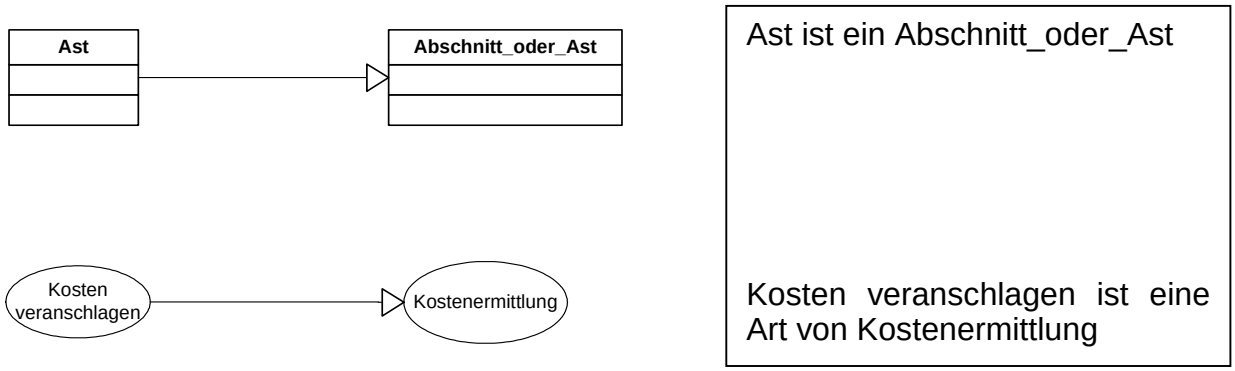


Abbildung 4. Generalisierung.

Hinweis: Bei ungeschickter Wahl der Namen der durch einen Generalisierungspfeil verbundenen Modellelemente kann es geschehen, dass die genannten Sprechweisen zu unverständlichen Sätzen führen! Daher sollten die Namen so gewählt werden, dass sie sich der Sprechweise anpassen.

5.2.2 Stereotype

Die UML-Modellkonstrukte können durch *Stereotype* modifiziert werden. Ein Stereotyp ist ein *Hinweis* darauf, dass die damit gekennzeichnete Modellelemente *zusätzliche* Eigenschaften oder besondere *Interpretationen* und *Verwendungsmöglichkeiten* haben.

> Die Bezeichnung ist etwas unglücklich gewählt, weil „Stereotyp“ außerhalb der Fachsprache die Bedeutung „Klischee“ hat. Es gibt aber leider keine eingängige andere Übersetzung des UML-Fachbegriffes.

Stereotype werden durch einen Namen bezeichnet, der in **<<>>** eingeschlossen ist. UML definiert einige Stereotype mit fester Bedeutung, die im UML-Standard bzw. einschlägiger Literatur nachzuschlagen sind (z.B. <<type>>, siehe unten). Daneben kann jedes Modell selbst eigene Stereotype definieren (wie z.B. <<okstra>>).

> Das Stereotyp <<okstra>> z.B. ist ein Hinweis darauf, dass eine damit gekennzeichnete Klasse eine Definition im bestehenden OKSTRA hat.

Zur Definition eigener Stereotype bietet sich als praktikabelste Lösung eine Tabelle an, die den Namen des Stereotyps und seine Beschreibung enthält. Die Beschreibung muss die Modellkonstrukte enthalten, auf die das Stereotyp angewendet werden kann, sowie die Bedeutung (also die zusätzlichen Eigenschaften, von denen oben die Rede war). Bei Verwendung eines geeigneten Modellierungs-Werkzeugs wird die Verwaltung der Stereotype mit unterstützt.

> Beispiel für eine Beschreibung: „<<okstra>> ist auf Klassen anwendbar. Wird eine Klasse so gekennzeichnet, so ist sie im bestehenden OKSTRA definiert und ihr dort niedergelegtes Schema wird gemäß den Interpretationsregeln des Leitfadens zur objektorientierten Modellierung des OKSTRA interpretiert. <<okstra>> impliziert die Eigenschaften für <<type>>.“

Stereotypen sind spezialisierbar, d.h. neue Stereotypen können aus vorhandenen abgeleitet werden, in dem zusätzliche Eigenschaften gefordert werden.

> Zur formal korrekten Definition von Stereotypen benötigt man das UML-Metamodell, dann kann die Definition selbst in UML erfolgen. Sie erfolgt durch Spezialisierung aus vorhandenen Stereotypen oder Klassen des UML-Metamodells.

5.3 Diagramme zur Prozessmodellierung

5.3.1 Aktivitätsdiagramme

Aktivitätsdiagramme dienen zur Darstellung von *Abläufen* von oder in Prozessen. Gegenüber UML 1.4 gibt es in UML 2.0 viele Änderungen bei diesem Diagrammtyp. Der Leitfaden empfiehlt die Verwendung von UML 2.0, weil sie die Modellierung von Geschäftsprozessen wesentlich besser unterstützt. Aktivitätsdiagramme die bestehen im Wesentlichen aus folgenden Elementen:

Aktionen beschreiben Schritte innerhalb eines Prozesses, die in sich abgeschlossen sind und definierte Ein- und Ausgangsbedingungen haben (z.B. Daten, die vorliegen müssen, Ergebnisse, die zu produzieren sind).

Kontrollfluss: Eine Aktion löst nach ihrem Ende eine oder mehrere Folgeaktionen aus. Dies wird durch durchgezogene Flusspfeile angezeigt. Die Übergänge können beschriftet sein, die Beschriftung gibt das Ereignis an, das dazu führt, dass die Zielaktion des Pfeiles ausgelöst wird. Gehen mehrere Kontrollflüsse von einer Aktion aus, so starten diese gleichzeitig; treffen mehrere ein, so beginnt die Aktion dann, wenn alle angekommen sind.

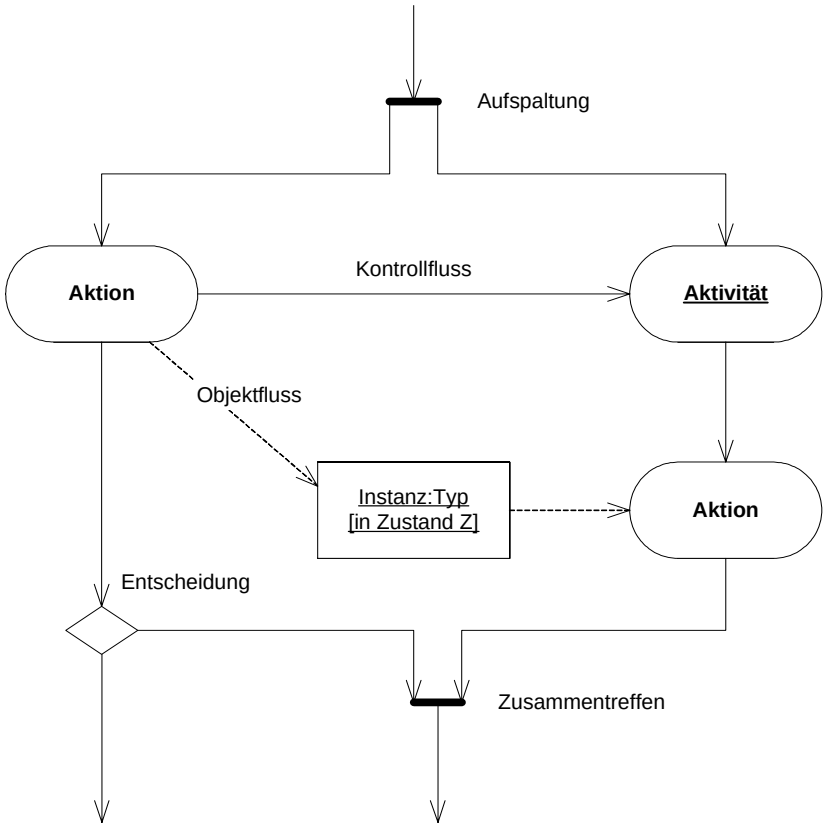


Abbildung 5. Muster Aktivitätsdiagramm.

Objekte: Der Informationstransport wird durch *Objekte* angezeigt, die in den Kontrollfluss eingefügt werden. D.h. ein Flusspfeil von einer Aktion zu einem Objekt bedeutet, dass die Aktion dieses Objekt als Ausgabe produziert hat, während ein Flusspfeil von einem Objekt zu einer Aktion anzeigt, dass die Aktion das Objekt als Eingabe konsumiert.

UML hat eine Notation für Objekte als prototypische Exemplare innerhalb von Prozessen. Sie werden durch Rechtecke dargestellt, in die der Name des Objektes unterstrichen eingetragen wird. Der Name kann durch einen Typ qualifiziert werden:

Name[:Typ] Beispiel: „L:Lichtsignalanlage“

Außerdem kann unterhalb des Namens in eckigen Klammern angegeben werden, dass sich das Objekt in einem bestimmten Zustand befindet, z.B. „[blinkend]“

Übergänge können durch ***Aufspaltungen*** und ***Verzweigungen*** modifiziert werden, um parallele und bedingte Abläufe anzuzeigen. Eine Aufspaltung hat einen Eingang, auf die der Pfeil zeigt, und mehrere Ausgänge. Die Aktionen entlang der Ausgangslinien können

parallel ablaufen. Ein ***Zusammentreffen*** hat umgekehrt mehrere Eingangspfeile und einen Ausgang. Die Aktionen entlang der einlaufenden Linien müssen alle beendet sein, bevor der Ablauf fortgesetzt wird.

Beispiele für Aktivitätsdiagramme finden sich im Anhang B sowie in [GP-Neu 02], Abschnitt 3.5.5.

Die Syntax und Semantik für Aktivitätsdiagramme in UML 2.0 ist sehr reichhaltig und wird hier nicht annähernd vollständig behandelt. Zusätzliche Information findet sich in [Oes 04], die vollständige Beschreibung findet sich im Standardisierungsdokument [UML-S 04], Kapitel 12.

5.3.2 Anwendungsfalldiagramme

Anwendungsfalldiagramme zeigen die Beziehung zwischen den Nutzern eines Systems oder den Beteiligten an einem Geschäftsprozess und den Leistungen, die das System oder der Prozess erbringt. Sie haben als Elemente:

Akteure. Akteure stehen für Klassen, die die Datenbasis des zu modellierenden Geschäftsbereichs pflegen oder nutzen. Typischerweise werden sie durch ihre Zuständigkeiten gekennzeichnet. Akteure können durch *Generalisierungsbeziehungen* verbunden sein, da es sich ja um eine besondere Art von Klassen handelt. Der spezialisierte Akteur hat dann die Zuständigkeiten des generalisierten, plus zusätzliche eigene.

Beispiel: Ein Akteur *Planer* könnte in *Straßenplaner* und *Landschaftsplaner* spezialisiert werden.

Anwendungsfälle: Sie beschreiben einen geschäftlichen Ablauf (Geschäftsprozess oder Teil davon) oder die von außen wahrnehmbaren Leistungen eines Systems. Auch Anwendungsfälle können spezialisiert oder generalisiert werden.

Beispiel: Ein Anwendungsfall *Zahlung* könnte in *Barzahlung* und *Kartenzahlung* spezialisiert werden.

Inklusions-Dependenzen - ***<<include>>***. Inklusionen beschreiben das Herausfaktorisieren von gemeinsamem Verhalten in untergeordnete Anwendungsfälle. Der Anwendungsfall, auf den der Pfeil zeigt, ist immer und zwingend Teil des Anwendungsfalles, von dem der Pfeil ausgeht. In Pfeilrichtung kann man Inklusionen lesen als: ***„erfordert zwingend“.***

Außerdem sind ***ungekennzeichnete Dependenzlinien*** zulässig in den Lesarten ***„nutzt, verwendet“***.

Falls UML-Werkzeuge, die Verwendung ungekennzeichneter Dependenzlinien verbieten, können zusätzliche Stereotypen definiert werden, die die Art der Abhängigkeit genauer beschreiben.

Generalisierungen. Siehe unter Akteure und Anwendungsfälle.

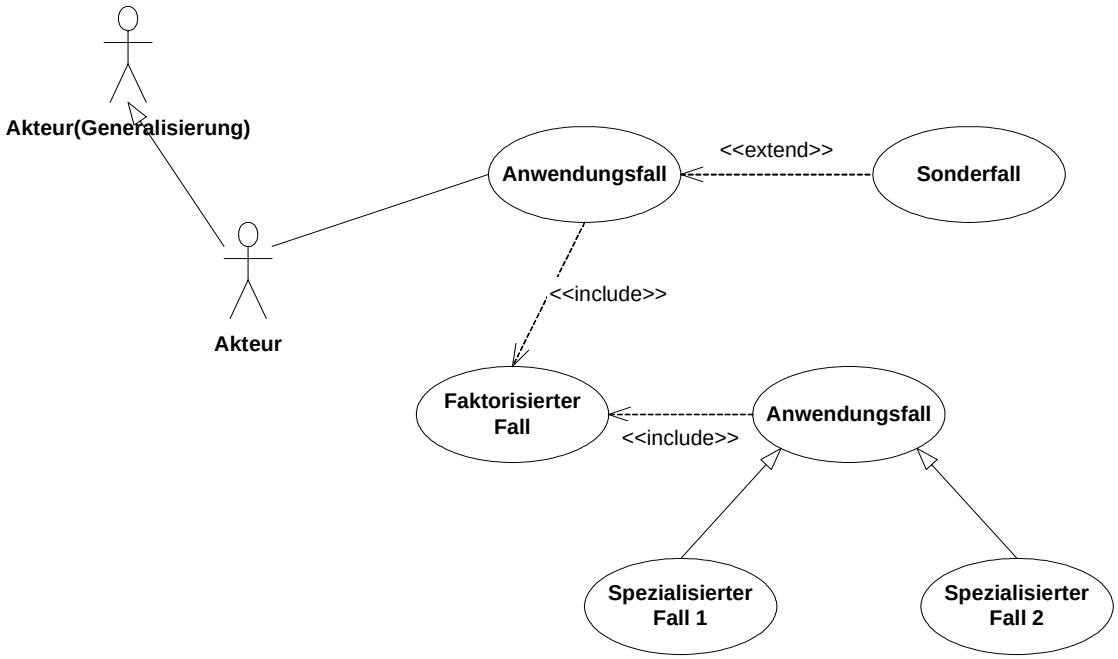


Abbildung 6. Muster Anwendungsfalldiagramm.

Anwendungsfalldiagramme und Aktivitätsdiagramme ergänzen sich. So können Szenarien für Anwendungsfälle oft durch Aktivitätsdiagramme ausgedrückt werden: Die zeitliche Abwicklung des Anwendungsfalls wird durch eine Abfolge von Aktionen und Aktivitäten ausgedrückt.

Umgekehrt können aus einem Aktivitätsdiagramm oft Anwendungsfälle gewonnen werden: Wird der Ablauf eines Geschäftsprozesses durch ein Aktivitätsdiagramm modelliert, so entsprechen den gezeigten Aktionen vielfach Anwendungsfälle.

5.4 Diagramme zur Objektmodellierung

5.4.1 Klassifizierer und Klassendiagramme

Klassendiagramme bilden die zeitunabhängige Struktur des Modells ab und entsprechen damit in ihrer Funktion den NIAM-Diagrammen.

UML definiert **Klassifizierer** als Oberbegriff zu allen Modellkonstrukten, die eine Zusammenfassung von Exemplaren nach Gemeinsamkeiten ausdrücken sollen. Der UML-Klassifiziererbegriff umfasst den *Klassenbegriff* des Kapitel 3. Für die Festlegungen dieses Abschnittes wird der Begriff *Klasse* auch als Synonym für Klassifizierer verwendet. Für die Klassendiagramme des OKSTRA werden aus UML folgende Klassifizierer benötigt:

Typen

Datentypen

Diese werden wie in Kapitel 3 angegeben interpretiert. Ihre benötigten Eigenschaften sind (siehe ebenfalls Kapitel 3):

Attribute

Assoziationen und Assoziationsrollen

Operationen

Generalisierungen als Beziehungen von Klassen zu Basisklassen.

5.4.1.1 Name

Jede Klasse erhält einen für das Modell eindeutigen Namen.

5.4.1.2 Instanzierbarkeit

Die Instanzierbarkeit eines Typs (siehe 3.3, 3.11) wird durch den Schriftstil des Namens angezeigt:

- Instanzierbare Klassen haben einen Namen in normaler Schrift.
- Nicht instanzierbare Klassen haben einen Namen in *kursiver* Schrift.

Datentypen sind immer instanzierbar.

5.4.1.3 Abteilungen

Klassen werden grafisch durch Rechtecke dargestellt. Jede Klasse enthält eine oder mehrere Abteilungen (rechteckige Boxen). Die erste Abteilung ist für den Namen, ein eventuelles Stereotyp und die Instanzierbarkeit reserviert.

Die folgenden beiden Abteilungen sind für Attribute bzw. Operationen vorgesehen. Weitere Abteilungen können angefügt werden, sie müssen mit einem Stereotyp versehen sein, das angibt, welche Art von Modellinformation sie enthalten. Existieren keine Attribute, kann die entsprechende Abteilung leer bleiben

5.4.1.4 Stereotype

Für die Klasse muss eines der folgenden Stereotype verwendet werden:

<<type>>

Ein ***Typ*** ist nach UML-Standard ein Klassifizierer, der

- nur abstrakte Operationen definiert,
- Attribute und Assoziationen ausweisen kann
- nur von Typ-Klassifizieren durch Spezialisierung abgeleitet sein darf

Laut Kapitel 3 ist das die Standard-Interpretation für Klassen im OKSTRA.

<<datatype>>

kennzeichnet einen Datentyp im Sinne des Kapitel 3.7. Es dürfen nur abstrakte Operationen angegeben sein.

<<enumeration>>

Eine ***Aufzählung*** ist ein Datentyp, dessen Wertebereich aus einer (im Regelfall kleinen) Menge namentlich identifizierter Elemente besteht. Die Liste dieser Aufzählungswerte wird in der zweiten Abteilung (der Attributabteilung) Zeile für Zeile notiert. Da eine Aufzählung eine besodere Art von Datentyp ist, dürfen Operationen angegeben werden.

<<metaclass>>

Die Klasse ist eine Metaklasse (s. Kap. 3.10), ihre Exemplare sind Klassen.

<<okstra>>

Ein Stereotyp speziell für die OKSTRA-Modellierung. Es ist eine Spezialisierung von <<type>> und bedeutet: Der Typ ist im bestehenden OKSTRA bereits definiert, seine Eigenschaften ergeben sich durch die Regeln der Uminterpretation des bestehenden OKSTRA-Modells in diesem Leitfaden.

<<okstra-daten>>

Eine Spezialisierung von <<datatype>> und bedeutet: Der Datentyp ist im bestehenden OKSTRA bereits definiert, seine Eigenschaften ergeben sich durch die Regeln der Uminterpretation des bestehenden OKSTRA-Modells.

Sind zusätzliche Stereotype erforderlich, so sind diese im Modell zu spezifizieren.

5.4.1.5 Sichtbarkeiten und Gültigkeit

Sichtbarkeiten werden in UML folgenden Modellkonstrukten zugeteilt:

- Attributen
- Assoziationsrollen
- Operationen

In UML können die Sichtbarkeitsstufen

{public}, kurz **+**, überall sichtbar

{protected}, kurz #, in Objekten abgeleiteter Typen sichtbar

{private}, kurz **-**, in den Objekten des definierenden Typs allein sichtbar

verwendet werden.

Es wird als Default {public} festgestzt, so dass das + nicht geschrieben zu werden braucht.

Die Verwendung von {protected} und {private} bzw. # und **-** ist nur zulässig bei Modellelementen, die der *Verständlichkeit* halber notiert werden (s.u.).

Die Gültigkeit (*scope*) gibt an, ob ein Attribut oder eine Operation auf Klassenebene oder auf Objektebene gültig sein soll.

Klassenoperationen sind in Kapitel 3.6 definiert. Daraus ergibt sich automatisch die Definition für Klassenattribute als Attribute, deren Akzessoren Klassenoperationen sind.

Klassenoperationen und -attribute werden in UML durch Unterstreichung gekennzeichnet.

5.4.1.6 Attribute

Es wird folgende UML-Syntax für Attribute zugelassen:

[Sichtbarkeit] Name [Kardinalität] [:Typ] [= Initialwert] [{Zusatz}]

Default für Sichtbarkeit ist +, für Kardinalität ist 1..1, für Zusatz ***changeable***. Bei einer Kardinalität > 1 ist statt eines einzelnen Wertes eine Menge von Werten zulässig. Die Kardi-

nalität * bedeutet: *beliebig viele*, eine Angabe 0..1 *kein oder ein,* 1 *genau ein*. Andere Zahlen als 0 oder 1 sind zulässig, z.B. 2..5 *zwei bis fünf*, 2..* *wenigstens zwei.*

Weitere Zusätze: ***frozen*** für nur lesbare Attribute, ***addOnly*** für multiple Attribute, die nur ergänzt werden können (d.h. man darf zusätzliche Werte hinzufügen)

Attribute *müssen* spezifiziert werden, wenn die damit verbundenen Akzessoroperationen sichtbar sein sollen. Diese werden dann im Klassendiagramm als Operationen nicht besonders bezeichnet. Ist {frozen} spezifiziert, fehlt die Mutator-Operation. Die Sichtbarkeit solcher Attribute ist per Default + und darf explizit nicht anders angegeben werden.

Attribute *können* spezifiziert werden, wenn der Aufbau des inneren Zustandes der Objekte aus Gründen der *Verständlichkeit* dargestellt werden soll. Die Sichtbarkeit muss dann mit – spezifiziert sein. In diesem Fall darf außer Sichtbarkeit und Name nichts weiter angegeben werden. Eine Angabe dieser Art ist so zu interpretieren, dass Objekte der Klasse *konzeptionell* einen entsprechenden inneren Zustand besitzen. Semantische Beschreibungen von Operationen können sich auf diesen Zustand beziehen.

5.4.1.7 Assoziationen

Die Eigenschaften von Assoziationen betreffen die Assoziationsenden. Genau wie für Attribute sind dies Name, Sichtbarkeit, Kardinalität und die dort angegebenen Zusätze. Die Namen der Assoziationsenden heißen ***Rollennamen***. Der Rollenname identifiziert eine ***Assoziationsrolle*** im Sinne des Kap. 3.

Die Kardinalität am Assoziationsende gibt an, wie viele Objekte unter der Assoziationsrolle sichtbar sein können. *Im Gegensatz zu NIAM werden in UML Rollennamen am Ziel notiert. Darauf muss man beim Lesen der Diagramme achten.*

Es werden dieselben Vereinbarungen für die Spezifikation und für Defaults wie bei Attributen getroffen, d.h.:

Ist ein Rollenname mit der Sichtbarkeit + (explizit oder per Default) eingetragen, werden für diese Assoziationsrolle implizit Akzessoroperationen angenommen.

Assoziationen *können* wie Attribute der Klarheit halber notiert werden, die Rollennamen haben dann die Sichtbarkeit – oder #. Der Rollenname dient oft zur Verbalisierung der Assoziation. Fehlt er, kann man eine Assoziationslinie als ***„gehört zu“*** lesen.

5.4.1.8 Aggregation und Komposition

Die *Aggregation* drückt die Beziehung ***„besteht aus“*** zwischen Objekten aus, d.h. die eine Klasse repräsentiert Bestandteile einer Ganzheit, die durch die andere Klasse beschrieben ist. Sie wird durch eine Assoziationslinie notiert, deren Ende an dem Klassifizierer für die Ganzheit eine ungefüllte Raute trägt. Dieses Ende hat implizit die Kardinalität 1.

Beispiel: Eine Straße besteht aus Abschnitten_oder_Ästen.

Die *Komposition* ist eine besondere Form der Aggregation: die Teile dürfen nur einem einzigen direkten „Besitzer“ gehören, werden mit diesem zusammen erzeugt und vergehen mit ihm. Ist die Ausweisung dieser Semantik nicht von vornherein möglich oder erwünscht, muss die Aggregationsbeziehung verwendet werden. Kompositionslinien haben statt der ungefüllten eine gefüllte Raute. Als Sprechweise: ***„besteht innerlich aus“***.

5.4.1.9 Objekte und Instanz-Dependenzen

Manchmal ist es vorteilhaft, Instanzen von Objekten in UML-Diagrammen zu notieren:

Ein *Objekt* wird ähnlich wie eine Klasse notiert. Der *Name* besteht aus einer optionalen Objekt-Identifikation, gefolgt von einem Doppelpunkt, gefolgt von dem Namen der Klasse, zu der das Objekt gehört. Attributwerte werden als *Attributname=Wert* notiert, die Assoziationslinie zwischen zwei Objekten bedeutet, dass diese unter einer Assoziation verknüpft sind.

Stehen zwei Modellelemente im Verhältnis von Typ zu Instanz, wird dies durch einen Dependenzpfeil vom Instanz- zum Typkonstrukt mit dem Stereotyp **<<instance>>** dargestellt.

Beispiele finden sich bei Sequenzdiagrammen und in den Rahmenmodellen (s.u.).

5.4.1.10 Operationen

Es wird folgende UML-Syntax für Operationen zugelassen:

[Sichtbarkeit] Name ([Parameterliste]) [:Typ] [{Zusatz}]

Default für Sichtbarkeit ist +, sowie keine Zusatzangaben. Für die Verwendung anderer Sichtbarkeiten gilt das bei Attributen gesagte sinngemäß.

Zusätze: ***isQuery**,* falls die Operation keine Zustände ändert.

Die *Parameterliste* ist eine komma-getrennte Liste mit Elementen der Form

[Richtung] Name : Typ

Richtung ist *in*, *out* oder *inout*, mit *in* als Default. *in* bedeutet, dass der Parameter von der Operation als Eingangsinformation benötigt wird, *out*, dass er keine Eingangsinformation enthält, sondern ein Ergebnis mitteilt, *inout*, dass er Information enthält, die von der Operation verändert werden kann.

5.4.1.11 Muster und Beispiel

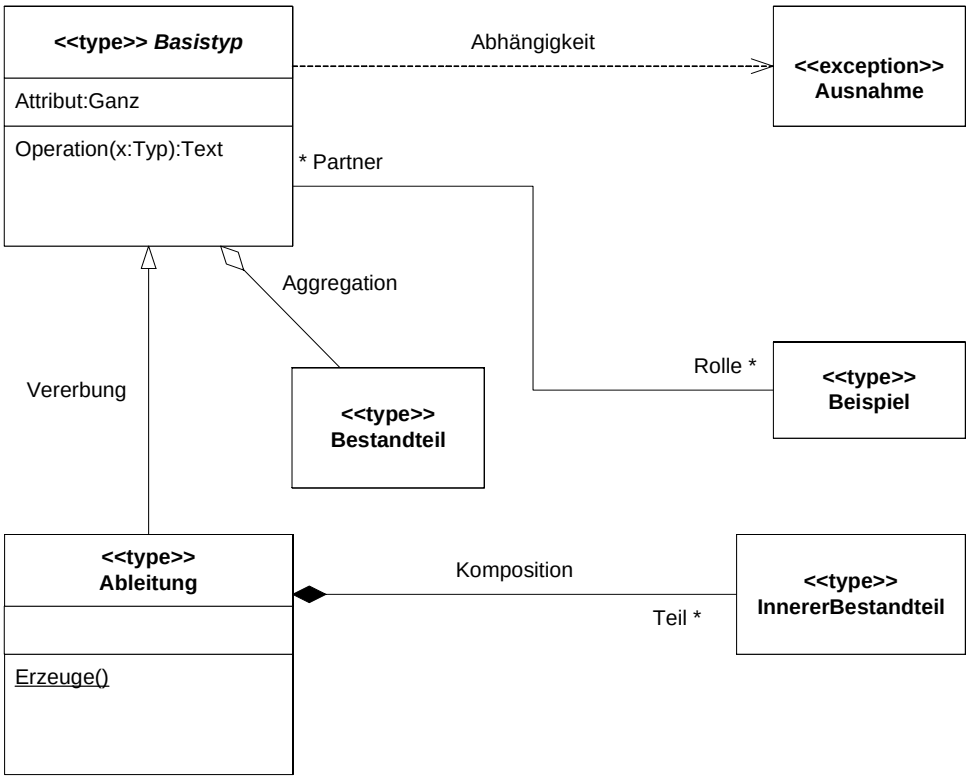


Abbildung 7. Muster Klassendiagramm.

Das folgende Beispiel stammt aus Anhang A. Hierzu folgende Erläuterungen:

- *Spezialisierung*: Eine ***AKS_Strecke ist*** ebenso wie ein AKS_Knoten ***ein AKS_Teil.***
- *Komposition:* Ein ***KBK besteht innerlich aus KBK_Position***-en***.***
- *Assoziation:* Ein ***AKS_Manager verwaltet beliebig viele AKS_Berechnungen,*** jede ***AKS_Berechnung gehört zu genau einem AKS_Manager***
- *Operation:* Die Operation ***NeueBerechnung*** von ***AKS_Manager*** verlangt als Eingabe einen Parameter ***Proto*** vom Typ ***Text.*** Sie teilt als Ergebnis eine ***AKS_Berechnung*** mit***.***
- *Dependenz:* Die Klasse ***KBK nutzt*** die Definition des Datentyps ***KBKid,*** wird diese geändert***,*** muss mglw***.*** die ***Klasse KBK*** angepasst werden***.***

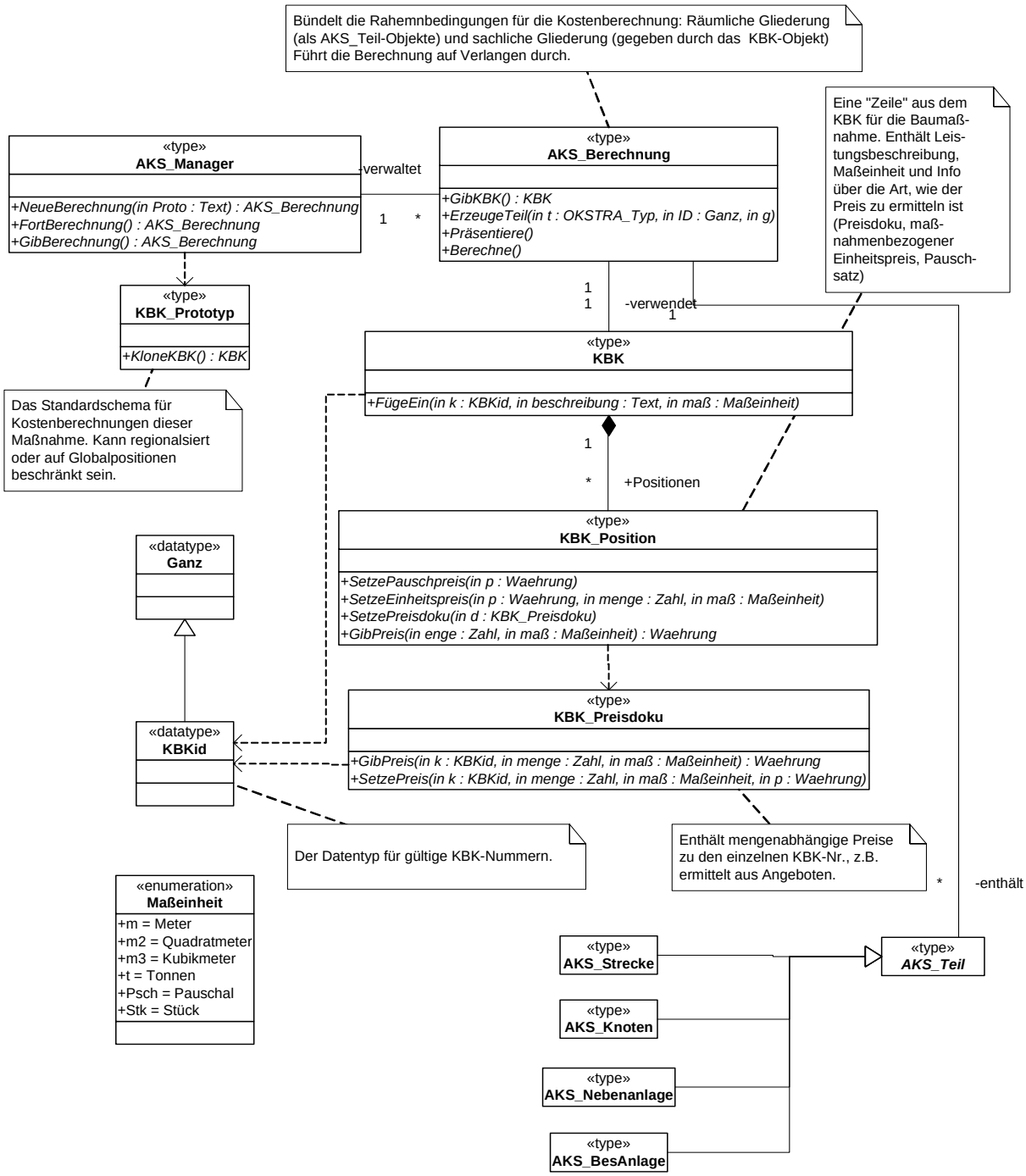


Abbildung 8. Beispiel Klassendiagramm.

5.4.2 Paketdiagramme

Pakete sind Hilfsmittel bei der Modellorganisation, um große Modelle in Teilmodelle zu zerlegen und dies Zerlegung auch grafisch darzustellen. Sie sind in erster Linie ein Hilfsmittel für die Spezifikationsmodellierung.

Ein Paket

- hat einen eindeutigen ***Namen***
- besteht aus *Modellelementen* und/oder ***Unterpaketen***
- legt einen ***Namensraum*** fest: Jedes Modellelement einer bestimmten Art muss einen eindeutigen Namen *innerhalb des Paketes* haben
- schirmt ***Sichtbarkeiten*** ab: ohne weiteres Zutun sind Modellelemente nur innerhalb ihres Paketes sichtbar.
- kann andere Pakete ***importieren***: die veröffentlichten Elemente eines importierten Pakets werden denen des Importeurs hinzugefügt (Vorsicht: dies kann Namenskonflikte erzeugen)

Ein Paketdiagramm enthält

Paketsymbole

Importpfeile

Generalisierungspfeile.

Ein Paketsymbol hat das Aussehen eines Ordners mit einem *Namensschild*. In das Paketsymbol trägt man ein:

den ***Namen*** in das Namensschild

die vom Paket ***veröffentlichten Klassifizierer*** in das Ordnerrechteck.

Die Enthaltenseinsbeziehung betrifft die Aufteilung eines Pakets in Unterpakete. Sie wird dargestellt, in dem die Unterpaketsymbole in das Ordnerrechteck eingetragen werden.

Die Generalisierungsbeziehung funktioniert so wie bei Einzelklassifizierern: Das spezialisierte Paket kann überall wie das Basispaket verwendet werden, es kann Elemente des Basispakets neu definieren bzw. Elemente hinzudefinieren.

Importpfeile sind Abhängigkeitspfeile mit dem Stereotyp **<<import>>.** Das Zielpaket wird vom Quellpaket importiert, die veröffentlichten Elemente des Zielpakets stehen dem Quellpaket zur Verfügung, nicht jedoch umgekehrt. Die Importbeziehung ist *nicht transitiv*: In einem Paket A, dass ein Paket B importiert, sind die von B seinerseits importierten Elemente nicht automatisch sichtbar.

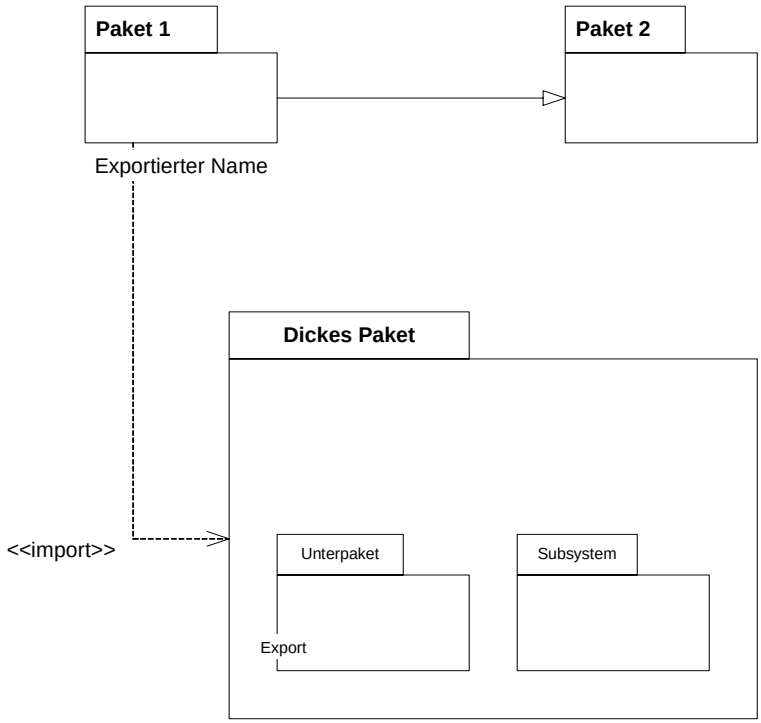


Abbildung 9. Muster Paketdiagramm.

5.4.3 Sequenzdiagramme

Sequenzdiagramme zeigen den Nachrichtenaustausch zwischen Objekten, die zusammenarbeiten, um einen Anwendungsfall abzuwickeln, und zwar in zeitlicher Reihenfolge.

Sequenzdiagramme werden ohne weitere Einschränkungen verwendet. Sie bestehen aus

Objekten, die Nachrichten austauschen. Dies dürfen auch Akteure sein, dann bezieht das Sequenzdiagramm die äußeren Nutzer mit ein.

Lebenslinien der Objekte. Die Linie beginnt am Objekt. Das Ende der Lebensdauer, wird, wenn es denn eintritt, durch ein großes **X** angezeigt.

Fokusanzeige: Ein Rechteck, das anzeigt, dass das Objekt etwas tut. Bei konzeptionellen Modellen lässt man den Fokus oft weg, manchmal erleichtert seine Darstellung jedoch den Überblick.

Nachrichtenpfeilen (durchgezogen), die das Senden einer Nachricht von einem Objekt zum anderen anzeigen. Hier sind die Stereotype **<<create>>** und **<<destroy>>** möglich, die das Erzeugen bzw. Vernichten eines Objektes bezeichnen.

Rückkehrpfeilen (gestrichelt), die insbesondere bei konzeptionellen Modellen weggelassen werden können. Sie zeigen auf das Objekt, das die Antwort zu einer Nachricht auffängt.

Die senkrechte Dimension entspricht der Zeit, und zwar üblicherweise von oben nach unten fortschreitend.

Beispiele finden sich im Anhang.

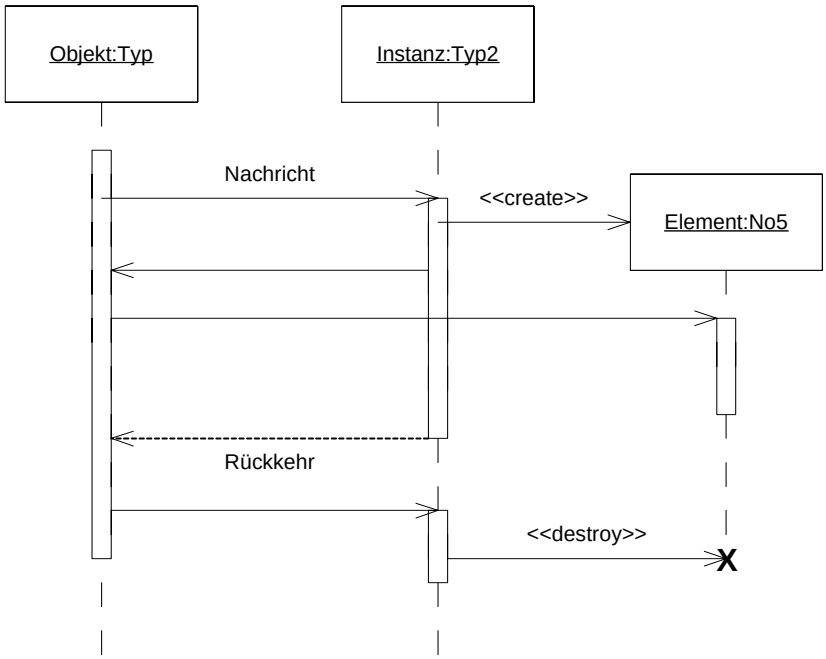


Abbildung 10. Muster Sequenzdiagramm.

6 Formalisierung und Dokumentation der OKSTRA-Modelle

Dieses Kapitel ist zur Ergänzung und Präzisierung der in 2.9 beschriebenen Aufgabe gedacht.

6.1 Modellarten

Fowler/Scott [Fow 00] unterscheiden drei Arten von Modellen

- Konzeptionsmodelle
- Spezifikationsmodelle
- Implementations- oder Realisierungsmodelle

Konzeptionsmodelle bilden die Konzepte der realen Welt und ihre Beziehungen im Geschäftsbereich ab (Ontologie). Die Modellierung des bestehenden OKSTRA in NIAM (ORM) gehört zu dieser Art von Modell.

6.2 Inhalt der Modelle

Zur Konstruktion von standardgerechter Software reichen *Objektmodelle* aus. Soll der objektorientierte OKSTRA jedoch auch zur Qualitätsprüfung und zu einer eventuellen Zertifizierung von Software herangezogen werden, muss jedoch nachgewiesen werden, dass die Software auch die gewünschte Prozessunterstützung leistet. In diesem Falle muss der objektorientierte OKSTRA auch die *Prozessmodelle* enthalten.

6.2.1 Prozessmodelle

Die Prozessmodelle bestehen aus

- Beschreibungen der Prozesse durch Aktivitäten,
- Beschreibungen der Anwendungsfälle.

Sie werden als UML-Aktivitäts- und UML-Anwendungsfalldiagramme notiert, zu denen zusätzlich eine freitextliche Beschreibung der Geschäftsprozesse, Aktivitäten, Ereignisse, Akteure und Anwendungsfälle mitgegeben wird.

Alle Beschreibungen enthalten:

- einen wohlbekannten und eingeführten *Namen* (z.B. *Bestandsfortführung, Kostenermittlung*)
- *Quellenangaben*, die auf die Regelwerke verweisen, aus deren Analyse sich das Modellelement ergeben hat. (z.B*. AKS 85*)
- falls gewünscht, eine kurze *fachliche Beschreibung* (notwendig, wenn die Regelwerke allein nicht genügend aussagekräftig sind)

Eine *Geschäftsprozessbeschreibung* enthält:

- die *Geschäftsziele* (z.B. *Vorschau auf die und laufende Kontrolle der Kosten des Projektes*)
- die *Rollen* der Teilnehmer am Prozess (z.B. *Planer, Bauabrechner*), d.h. die beteiligten Akteure

- Verweis auf das zugehörige *Aktivitätsdiagramm*

Eine *Aktivitätsbeschreibung* enthält:

- Angabe der benötigten *Eingangsinformationen*
- Angabe der erzeugten *Ergebnisse* (eventuell unter verschiedenen Bedingungen: Normalfall, Fehlerfall...)

Eine *Anwendungsfallbeschreibung* enthält:

- Angabe der *auslösenden* Geschäftsvorfälle
- Angaben zu notwendigen *Berechtigungen* der beteiligten Akteure
- Angabe der benötigten *Eingangsinformationen*
- Angabe der erzeugten *Ergebnisse* (eventuell unter verschiedenen Bedingungen: Normalfall, Fehlerfall...)
- *Ablauf* (als Szenario oder Verweis auf Aktivitätsdiagramm)

Eine *Akteurs*-Beschreibung enthält:

- *Art*: Rolle, System, Geschäftsbereich, Ereignis
- Beschreibung der fachlichen *Aufgaben* und *Zuständigkeiten* des Akteurs

Eine Beschreibung für *Ereignisse* im eigenen Geschäftsbereich enthält:

- *Auslösende* Bedingungen (z.B. Beginn oder Ende einer Aktivität, Fristablauf, Eintreten einer auf Messwerten basierten Bedingung)
- Liste der *fremden Geschäftsbereiche*, die eine Nachricht über das Ereignis benötigen
- Information, die den Empfängern bei Eintreten zur Verfügung gestellt werden soll

Eine Beschreibung für *Ereignisse* anderer Geschäftsbereiche, die den eigenen beeinflussen, enthält

- Information, die beim Empfang der Nachricht über den Eintritt benötigt wird

Die Beschreibungen können mit UML-Mitteln in den Diagrammen selbst notiert werden. Wo sie jedoch zu umfangreich werden, ist eine separate Beschreibung vorzuziehen und im UML-Diagramm ein Verweis (z.B. Abschnittsnummer eines begleitenden Dokumentes oder Hyperlink) anzubringen.

Weitere Informationen zu einer Beschreibung der genannten Modellelemente finden sich in [Oes 03].

6.2.2 Objektmodelle

Ein konzeptionelles Objektmodell für den objektorientierten OKSTRA berücksichtigt primär die *semantischen* Aspekte der Operationen und hat folgende Eigenschaften:

- Es enthält Klassen für alle zu berücksichtigenden Objektarten des Geschäftsbereichs.
- Es enthält je eine Operation pro identifizierter funktionaler Spezifikation (siehe 3.2).

  Beispiel: **GibMenge**. Berechnet im Objekt enthaltene Menge in verlangter Maßeinheit.

- Die Operationen werden allen Klassen zugeordnet, die die modellierte Funktionalität unterstützen müssen (Polymorphie *über* Klassen *hinweg*).

  GibMenge für: Tragschicht, Rohrleitung, Bodenaushub

- Polymorphe Varianten *innerhalb* einer Klasse (Operationen mit unterschiedlichen Parametrisierungen) brauchen noch nicht berücksichtigt zu werden.
- Die genaue Parametrisierung der Operationen braucht nicht angegeben zu sein.
- Operationen können auch als Attribute oder Assoziationsrollen gegeben sein.
- Kardinalitäten von Assoziationen sind nur anzugeben, wo bekannt
- Die funktionalen Spezifikationen für die Operationen sind vollständig dokumentiert.

Konzeptionsmodelle eignen sich zur Kommunikation des Software-Architekten mit der Fachwelt, aber nicht als Bauplan für Software. Sie müssen allerdings alles zur Erstellung eines solchen Plans nötige Wissen aus dem modellierten Fachbereich wiedergeben.

Spezifikationsmodelle enthalten zusätzlich die exakten *formalen* Spezifikationen für die Klassen. Die bestehende EXPRESS-, XML-und SQL-Modellierung ist von dieser Art. Um normgerechte Software herzustellen und zu verwenden, braucht man ein Spezifikationsmodell, es richtet sich nicht an die Fachwelt, sondern an die Software-Hersteller.

Für die OKSTRA-Modellierung wird von einem Spezifikationsmodell gefordert:

- Alle Klassen sind Typen oder Datentypen. Alle zu berücksichtigenden Objektarten des Geschäftsbereichs sind repräsentiert
- Jede benötigte funktionale Spezifikation ist durch wenigstens eine Operation berücksichtigt. Gibt es mehrere, so tragen sie den gleichen Namen (polymorphe Varianten).
- Gruppen von Operationen, die mehreren Klassen gemeinsam sind, sind in Basistypen zusammengefasst.
- Die genaue Parametrisierung wird zu jeder Operation angegeben, d.h Zahl und Typ der Parameter und des Ergebnisses
- Attribute ohne Mutator-Operation (unveränderliche Attribute) werden als solche gekennzeichnet.
- Alle Kardinalitäten von Assoziationen sind angegeben.
- Fachlich zusammenarbeitende Klassen sind zu Diensten zusammengefasst.
- Jeder Dienst definiert ein oder mehrere Schnittstellen, die fachlich kooperierende Operationen bündeln.
- Jeder Schnittstelle enthält Fabrikfunktionen für die Exemplare jener Klassen, die für die Nutzer des Dienstes instanzierbar sein sollen.

Implementationsmodelle beschreiben die Realisierung von Spezifikationsmodellen durch Klassen mit Methoden. Da ein Spezifikationsmodell auf unterschiedliche Weise realisiert werden kann, gibt es keine 1:1 Abbildung von Spezifikationsmodellen in Implementationsmodelle.

Die Modellierung des objektorientierten OKSTRA soll Operationen für die OKSTRA-Objekte formal und funktional festlegen, d.h. Spezifikationsmodelle definieren. Implementationsmodelle werden nicht benötigt.

6.3 Semantische Dokumentation

Die semantische Dokumentation, d.h. die Beschreibungen der Prozesselemente und die funktionalen Spezifikationen der Operationen, ist ein sehr wichtiger Teil des Modells. Es gibt folgende Alternativen:

- Kommentare in UML bzw. *tagged values* in { }. Dies ist der Vorschlag des UML-Standards. Das Problem dabei ist die Unübersichtlichkeit. Jede Operation muss ja selbst beschrieben werden, und alle ihren Parameter dazu.
- Kommentare in UML, die Verweise (z.B. Hyperlinks) auf ein oder mehrere separate Dokumente enthalten. Die UML-Dokumente müssen eine Navigation auf die Beschreibungsdokumente zulassen und umgekehrt.
- Eine Variante der vorigen Alternative ist die separate Führung der Beschreibungen in einer Datenbank. Damit ein plattformunabhängiger Modelltransport gewährleistet ist, braucht man entsprechend eine plattformunabhängige Datenbank (z.B. mySQL). Damit wiederum zwischen dem grafischen UML-Modell und der Datenbank kein Medienbruch entsteht, muss eine Navigation aus dem UML-Dokument in die Datenbank möglich sein (z.B. über URLs, wenn die Datenbank über einen Web-Applikationsserver erreichbar ist).
- Statt der vorigen „Zu-Fuß“-Ansätze leistet die Verwendung eines UML-fähigen, plattformunabhängigen CASE-Tools, also eines Software-Werkzeugs zur Erstellung von Bauplänen für Software, das Gewünschte.

  Eine erschöpfende und ständig aktualisierte Liste aller CASE-Tools für die Modellierung in UML findet sich unter

  http://www.objectsbydesign.com/tools/umltools_byCompany.html

  Sie zeigt, dass kommerzielle Produkte mit dem Anspruch auf vollständige Abdeckung des UML-Standards teuer sind und daher für den Austausch mit den Fachleuten kaum in Frage kommen. Diese Produkte sind zumeist für die Realisierung von einsatzfähiger Software konzipiert und enthalten viele Funktionen, die für die Modellierung des OKSTRA gar nicht benötigt werden.

  Die Liste zeigt jedoch auch, dass mittlerweile eine größere Zahl von frei verfügbaren Produkten existiert, wie z.B. ArgoUML oder Poseidon.

6.4 Erstellung, Pflege und Verteilung der Modelle

Das Modell des bestehenden OKSTRA wird dargestellt durch:

- NIAM-Diagramme als graphische Darstellung
- EXPRESS-Schemata als textbasierte Darstellung
- Objektbeschreibungen in Form von Text-Tabellen

Bei der Erweiterung des Modells müssen alle drei Darstellungsformen fortgeschrieben werden. Eine integrierte Darstellung des Modells wäre wünschenswert, weil damit erstens die Pflege erleichtert würde und zweitens der Transport des Modells, d.h. die Übergabe in IT-gerechter Form, vereinfacht würde.

Für den objektorientierten OKSTRA ist von vornherein eine Darstellungsform anzustreben, die den Medienbruch, der durch die nicht formal korrelierten Darstellungen des bestehenden, statischen OKSTRA entsteht, vermeidet.

Der objektorientierte OKSTRA wird zunächst durch UML-Diagramme beschrieben.

Im Gegensatz zur UML-Version 1.4 enthält der kommende UML-2.0-Standard nicht nur Regeln zum Transport der inhaltlichen Information der Modelle nach dem XMI-Standard (XML Metadata Interchange), sondern er spezifiziert in Ergänzung hierzu auch einen Weg, um die Layout-Information für die Grafik, die mit dem Modell verbunden wurde, zu übertragen (UML Diagram Interchange) [UML-DI 03].

Aus diesem Grund empfiehlt dieser Leitfaden zur Modellierung die Verwendung vom UML 2.0 sowie eines plattformunabhängigen CASE-Tools, das UML 2.0 und diesen Mechanismus unterstützt. Das Werkzeug sollte auch in der Lage sein, aus dem Spezifikationsmodell zusätzliche Sprachbindungen (C++, C#, Java, IDL...) herzustellen.

Dazu muss zunächst eine vergleichende Beurteilung der auf dem Markt verfügbaren Werkzeuge duchgeführt werden. Kriterien hierfür sind:

- Unabhängigkeit von der Betriebssystemplattform
- Unterstützung von UML 2.0 und UML Diagram Interchange
- Export in offene Grafikformate (PDF, SVG)
- Generierung von Sprachbindungen (insbesondere auch an XML-basierte. z.B. für Web Services)
- Skalierung nach benötigtem Funktionsumfang (preiswerte Version für die konzeptionelle Modellierung im Team mit Anwendungs-Fachleuten, Version mit hohem Funktionsumfang für die Erstellung, Wartung und Publikation der Spezifikations-Modelle)

7 Rahmenmodelle

Hier werden einige allgemeine Systemstrukturen beschrieben, die für den Aufbau des objektorientierten OKSTRA benötigt werden. Die angegebenen Modelle sind nur als Lösungs*ansätze* zu verstehen. Für viele der hier beschriebenen Rahmenmodelle existieren bereits Vorbilder in Form von existenten oder in Entstehung begriffener Standards, wobei manchmal sogar mehrere Alternativen zur Auswahl stehen. Der Leitfaden weist hier ausdrücklich darauf hin, dass wo immer möglich und vertretbar Standards genutzt werden sollten, jedoch obliegt eine Entscheidung darüber im Einzelfall den Gremien, die die Weiterentwicklung des OKSTRA offiziell betreuen. Folglich ist dieses Kapitel nicht als Spezifikation, sondern als Aufgabenliste zu verstehen.

7.1 Basisdatentypen

7.1.1 Einfache Datentypen

Die Definition der einfachen Datentypen lehnt sich an die existierende Modellierung in EXPRESS an. Einschränkungen bzgl. Wertebereich, Genauigkeit usw. können durch Spezialisierung erhalten werden.

Ganz	die ganzen Zahlen der Mathematik
Reell	die reellen Zahlen der Mathematik, (üblicherweise Gleitkommaformat)
Logisch	3-wertige Logik: Wahr, Falsch, Unbekannt
Bool	2-werige Logik: Wahr, Falsch
Text	Zeichenkette beliebiger Länge
Blob	Binärdaten ohne Interpretation, beliebige Länge

7.1.2 OKSTRA-Datentypen

Die existierenden OKSTRA-EXPRESS-Schemata für Typen von Werten (Zahlen usw.) (*Allgemeine_Objekte* sowie einige weitere in anderen Teilmodellen) können vom objektorientierten Modell aus wie folgt referiert werden: Es wird ein Klassifizierer mit dem Stereotyp <<okstra-daten>> und dem Namen des Typs wie in EXPRESS angeschrieben. Von diesem können dann nach Bedarf Ableitungen gebildet werden, und Attribut- und Operationsbeschreibungen können ihn dann verwenden.

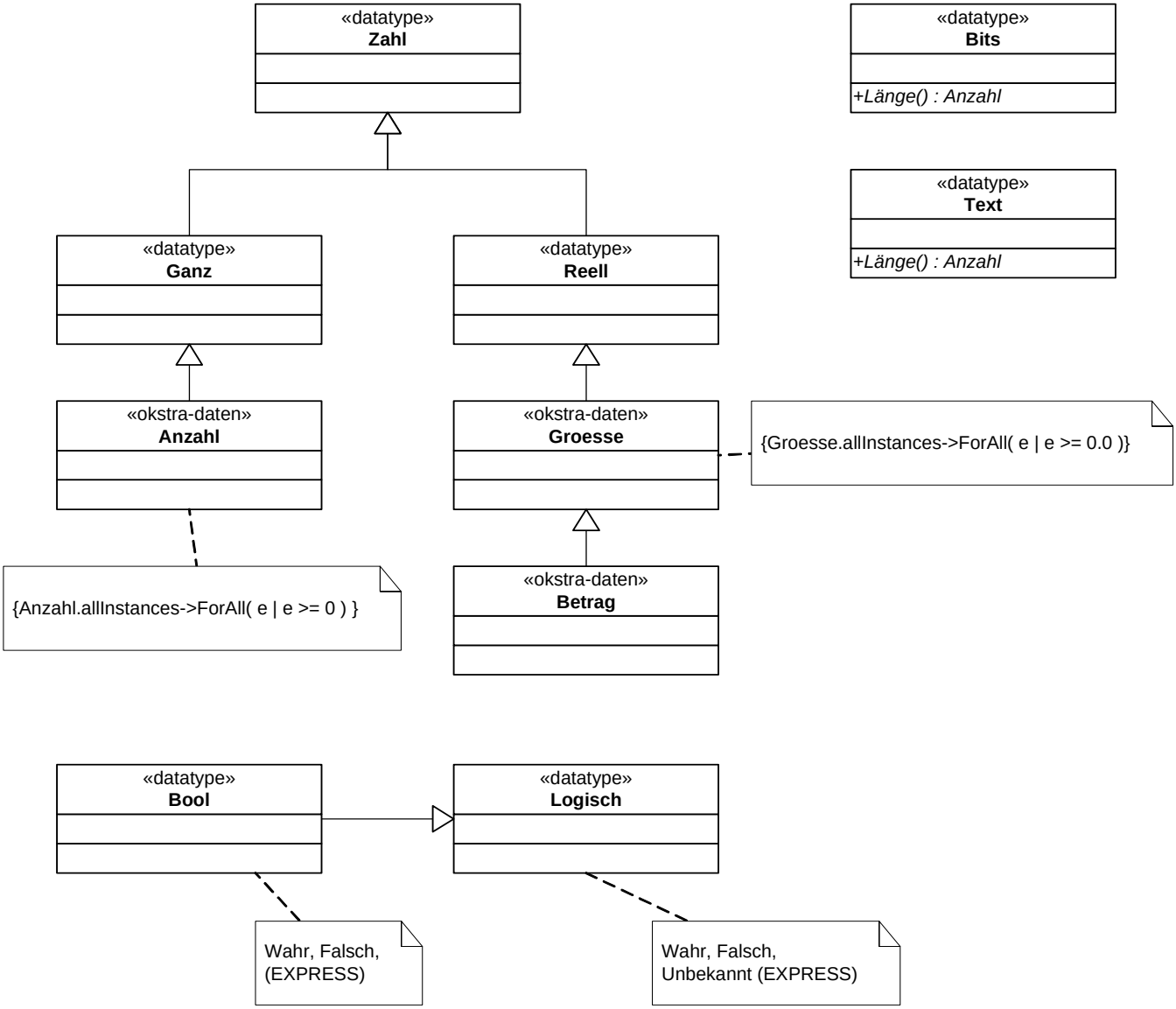
7.1.3 Aufzählungen

Aufzählungen werden durch einen Klassifizierer mit dem Stereotyp **<<enumeration>>** angegeben. Die Liste der Aufzählungswerte erscheint in der Attributabteilung des Klassifizierers.

7.1.4 UML-Diagramm Datentypen

Das folgende Beispiel zeigt einige im OKSTRA und im Beispiel Kostenberechnung benötigte Datentypen. Bedingungen für die Wertebereiche der Datentypen *Anzahl* und *Groes-*

se sind beispielhaft in *Object Constraint Language OCL* angeschrieben, einer zu UML definierten formalen Sprache für Bedingungen. Die Syntax und Semantik der OCL-Ausdrücke ist dem UML-Standard [UML-S 04] bzw. einschlägiger Literatur zu entnehmen.



7.2 Das OKSTRA-Objekt

Alle OKSTRA-Klassen werden letzlich aus einer Klasse ***OKSTRA_Objekt*** abgeleitet. Diese nimmt alle Operationen auf, die jedes Objekt des OKSTRA zur Verfügung stellen muss:

Typ()

Die Operation gibt als Ergebnis ein Objekt vom Typ *OKSTRA_Typ*, das den Typ des befragten Objektes repräsentiert. Näheres hierzu siehe unter Typsystem.

HatTyp(t)

Teilt mit, ob das Objekt vom Typ t oder von einem Subtypen davon ist

IstIdentisch(o)

Teilt mit, ob das Objekt mit o identisch ist.

IstGleich(o)

Teilt mit, ob das Objekt den gleichen Zustand wie o hat.

7.3 Objekt-Identifikatoren und Namensräume

Der statische OKSTRA definiert keine Objektidentität. Dies ist auch nicht notwendig, solange der OKSTRA primär als Schema zum Datenaustausch gesehen wird, da dann der Lebenslaufgedanke, der für die Objektorientierung typisch ist, unerheblich ist.

Bereits bei Architekturen, wo der OKSTRA als Schema zur dauerhaften Speicherung von Objekten verwendet wird, ist dies nicht mehr so. In Informationsarchitekturen, bei denen die Objekte sich gegenseitig über Nachrichten verständigen, muss jedes Objekt während seiner Lebensdauer eindeutig über einen Namen identifizierbar sein, unabhängig, ob eine dauerhafte Speicherung vorgesehen ist oder nicht.

Lose gekoppelte Systemarchitekturen wie Web Services behandeln den Transport von Objektidentitäten nicht vollständig transparent. Um mit solchen Infrastrukturen arbeiten zu können, wird eine Klasse ***OKSTRA_ID*** eingeführt, die Objektidentitäten kapselt. OKSTRA_ID wird als Datentyp konstruiert, z.B. als Ableitung aus Text.

> URIs (also die Adressen zur Identifizierung von Ressourcen im World Wide Web) können z.B. als Implementation von OKSTRA_IDs verwendet werden.

Weitere Operationen der Klasse OKSTRA_Objekt:

GibID( )

Gibt die OKSTRA_ID des Objektes.

Operationen der Klasse OKSTRA_ID:

GibObjekt( )

Gibt das Objekt zur ID

Zweckmäßig ist es oft, wenn man der ID schon den Typ des identifizierten Objektes ansieht, ohne dieses erst selbst befragen zu müssen:

Typ()

Die Operation gibt als Ergebnis ein Objekt vom Typ *OKSTRA_Typ*, das den Typ des befragten Objektes repräsentiert. Näheres hierzu siehe unter Typsystem.

Objekt-Identifikatoren werden üblicherweise von einer entsprechenden Fabrik (s. 7.7) vergeben. Da es vollkommen unpraktikabel ist, eine einzige zentrale „Id-Fabrik“ für alle nur denkbaren OKSTRA-Objekte zu unterhalten, baut man die Identifikatoren zweckmäßig aus Teilen zusammen, deren Fabriken miteinander in Verbindung stehen. Eine solche Verteilung der Verantwortlichkeit für die Vergabe der Identifikatoren zerlegt deren Gesamtmenge in geschachtelte *Namensräume*.

> So können z.B. die Identifikatoren für die Objekte einer Baumaßnahme aus der Maßnahmen-Id (die von einem zentralen Server bundesweit vergeben wird (BVWP!)) und einer maßnahmen-internen Id (die von einem Server z.B. des zuständigen SVA vergeben wird) zusammengebaut werden.

7.4 Ansammlungen

An etlichen Stellen werden bei der OKSTRA-Modellierung Ansammlungen (engl. *collections*) benötigt. Dies ist bereits bei der bestehenden Modellierung in EXPRESS der Fall. *Eine* wesentliche Anwendung von Ansammlungen ist die Ausdrückbarkeit von Assoziationen durch Operationen. Ein Observator, der alle mit seinem Objekt über eine 1:n-Assoziation verknüpften Partnerobjekte mitteilen soll, muss diese Gesamtheit in Form eines Objektes verpacken können.

Akzessoren für 1:n oder m:n –Assoziationen liefern als Antworrt bzw. erwarten als Parameter Ansammlungen.

Die Typen für Ansammlungen können entweder dazu benutzt werden, das entsprechende Interface an andere OKSTRA-Typen zu vererben, oder sie können durch eigene Klassen implementiert werden oder beides.

Wie üblich sind die folgenden Ausführen als Anregungen zu verstehen. Weitere Vorbilder liefern die Klassenbibliotheken der gängigen Programmiersprachen, z.B. Java oder C#.

Eine ***Ansammlung*** kann leer sein oder beliebig viele *OKSTRA_Objekte* enthalten. Jede Ansammlung muss die folgenden Operationen unterstützen:

Erzeuge(t)

Erzeugt eine neue leere, Ansammlung, die Elemente des Typs t aufnehmen kann. Es können nur Elemente eingefügt werden, die von diesem Typ sind (und natürlich allen Subtypen dazu)

ElementTyp()

Gibt den Typ der Elemente der Ansammlung.

IstIn(o)

Überprüft, ob das Objekt o Element der Ansammlung ist. Antwortet mit Wahr, wenn ja.

GibAnzahl()

Teilt die Anzahl der Elemente der Ansammlung mit.

Die folgenden Operationen haben in den gleich zu besprechenden Ableitungen von *Ansammlung* unterschiedliche Parametrisierung.

FürAlle(c)

Die wichtigste Möglichkeit, die eine Ansammlung erlaubt, ist, alle ihre Elemente nacheinander derselben Behandlung zu unterziehen. In objektorientierten Architekturen sind hierfür zwei konkurrierende Operationsmuster anzutreffen. Beim ersten wird zu einer Ansammlung ein sogenanntes *Iterator*-Objekt erzeugt. Dieses verfügt über eine Operation, die bei jeder Ausführung ein neues Element der Ansammlung abliefert. Das zweite Muster fordert für Ansammlungen eine Operation, die von selbst intern alle Elemente abarbeitet und auf jedes davon eine von außen als Parameter gegebene Operation, einen *Callback*, anwendet.

Beim Iterator muss die nutzende Umgebung über einen Mechanismus verfügen, der den wiederholten Aufruf des Iterators gestattet. Solche Mechanismen sind in Programmiersprachen i.A. in Form von *Programmschleifenkonstrukten* vorhanden.

Das zweite Muster kommt ohne die implizite Voraussetzung von Schleifen aus. Für den OKSTRA ist dies zweckmäßiger.

Die Operation *FürAlle* gestattet, nacheinander auf alle Elemente Operationen anzuwenden, die von den Elementen der Ansammlung unterstützt werden; die Reihenfolge, in der die Elemente angesprochen werden, ist i.A. willkürlich, kann aber für besondere Arten von Ansammlungen besonders geregelt sein. Die durchzuführenden Operationen werden in einem *Callback*-Objekt c gekapselt.

FügeEin(o, ...)

Fügt das Objekt o als Element in die Ansammlung ein.

Entferne(o, ...) : o

Entfernt ein Objekt o aus der Ansammlung

7.4.1 Mengen

Mengen sind ungeordnete Ansammlungen. Ein Objekt kommt in einer Menge höchstens einmal vor. Folgende Parametrisierungen gelten für das Einfügen und Entfernen

FügeEin(o)

Entferne(o) : o

7.4.2 Folgen

Folgen sind geordnete Ansammlungen. Ein Objekt ist innerhalb der Folge durch eine Position (Ordnungszahl > 0) ausgezeichnet und kann darüber angesprochen werden. *FügeEin* und *Entferne* müssen die Position als Parameter angeben. Die Iterationsfunktion arbeitet die Elemente nach aufsteigender Position ab.

FügeEin(o, p)

Fügt das Element o vor der Position p ein. Nach der Ausführung steht daher o an der Position p.

Entferne(p) : o

Entfernt das Objekt an der Position p. Das Objekt wird als Antwort mitgeteilt.

GibElement(p)

Gibt das Element an der Position p der Folge als Antwort.

FürAlle(c)

Die Folge wird nach aufsteigender Position abgearbeitet. Der Callback wird mit der Position des Elementes parametrisiert.

7.4.3 Tabellen

Tabellen sind ungeordnete Ansammlungen. Ein Objekt ist innerhalb der Tabelle durch einen eindeutigen Schlüssel ausgezeichnet und kann darüber angesprochen werden. *FügeEin* und *Entferne* müssen den Schlüssel angeben. Der Schlüssel kann ein beliebiges *OKSTRA_Objekt* sein. Die Eindeutigkeit des Schlüssel wird über *IstGleich*() überprüft. Die Iterationsfunktion arbeitet die Elemente willkürlich ab.

FügeEin(o, s)

Fügt das Element o mit dem Schlüssel s ein.

Entferne(s) : o

Entfernt das Objekt mit dem Schlüssel s. Das Objekt wird als Antwort mitgeteilt.

GibElement(s)

Gibt das Element zum Schlüssel s in der Tabelle als Antwort.

FürAlle(c)

Die Folge willkürlich abgearbeitet. Der Callback wird mit dem Schlüssel des Elementes parametrisiert.

7.4.4 Callbacks

Für die Iterationsfunktionen sind Objekte sogenannter *Callback*-Klassen erforderlich. Diese kapseln die wiederholt auszuführende Operation. Das Modell spezifiziert je einen Callback-Typ für jeden der Ansammlungstypen. Das Callback-Interface fordert nur die Operation

Execute(o, ...)

Bei Ausführung der *FürAlle*-Operation wird für jedes Element einer Ansammlung *Execute* ausgeführt. Die Realisierung der Callback-Objekte enthält als Zustand alles, was zur Kommunikation sowohl mit dem Nutzer von *FürAlle* als auch mit dem Ansammlungselement nötig ist.

7.4.5 Darstellung von EXPRESS-Aggregaten

Um den Anschluss an die existierende Modellierung zu behalten, hier die Regeln für die Abbildung der EXPRESS-Aggregate an.

EXPRESS-Sets bilden sich auf Mengen ab.

EXPRESS-Listen bilden sich ab auf Folgen mit den positiven ganzen Zahlen als Indexmenge und u.U. variabler Kardinalität.

EXRESS-Bags und –Arrays werden zz. in der OKSTRA-Modellierung nicht verwendet. Sollte ihre Nutzung notwendig werden, sollte ein entsprechender Ansammlungstyp hinzumodelliert werden. Arrays können z.B. auf Folgen abgebildet werden mit Indexmengen aufeinanderfolgender ganzer Zahlen und fester Kardinalität. Implementationen für den Typ Folge in einer Umgebung, die Arrays unterstützt, können Operationen anbieten, die die Folge-Objekte in native Arrays umwandelt (und zurück).

7.4.6 Akzessoren für multiple Attribute und :n Assoziationen

Akzessoren für multiple Attribute gestalten sich wie folgt::

Der Observator liefert alle Attribute als Ansammlung ab, d.h. als Menge, Folge oder Tabelle. Ein einzelner Wert kann über die Operationen für den verwendeten Ansammlungstyp entnommen werden.

Der Mutator nimmt eine Ansammlung als Parameter und weist ihre Elemente dem multiplen Attribut zu.

Analog wird für die Akzessoren von :n-Assoziationen verfahren.

Welcher Ansammlungstyp gewählt wird, kann in der Spezifikation vorgegeben werden oder zunächst offengelassen sein.

Als Beispiel die Operationsfolge für das Einfügen einer neuen Verknüpfung unter einer Assoziation:

- Observator liefert Ansammlung der Assoziationspartner.
- *FügeEin* stellt neues Objekt in die Ansammlung.
- Mutator ersetzt bisherige Assoziationspartner durch die der geänderten Ansammlung.

Dies ist ein konzeptionelles Muster, das sicherstellt, dass die Assoziation modifizierbar ist. Häufig möchte man die Ansammlungen jedoch nicht sichtbar machen. Dann muss man die Ansammlungsoperationen hinter solchen der betroffenen Klasse verstecken.

Beispiel: Multiples Attribut *Monatswerte* (irgendwelche!)

MaxMonatswerte()

Die Maximalzahl der Werte. Hier 12.

AnzahlMonatswerte()

Die aktuelle Zahl der Werte.

GibMonatswert(i)

Gibt den i-ten Monatswert zurück

NeuerMonatswert(w)

Belegt den nächsten freien Monatswert.

ÄndereMonatswert(i,w)

Ändert den i-ten Monatswert

Man vereinbart in diesem Leitfaden am besten ein Standardmuster für solche Schnittstellen, einschließlich Namensgebung.

Es folgt eine beispielhafte Modellierung in UML für die bisher beschriebenen Klassen.

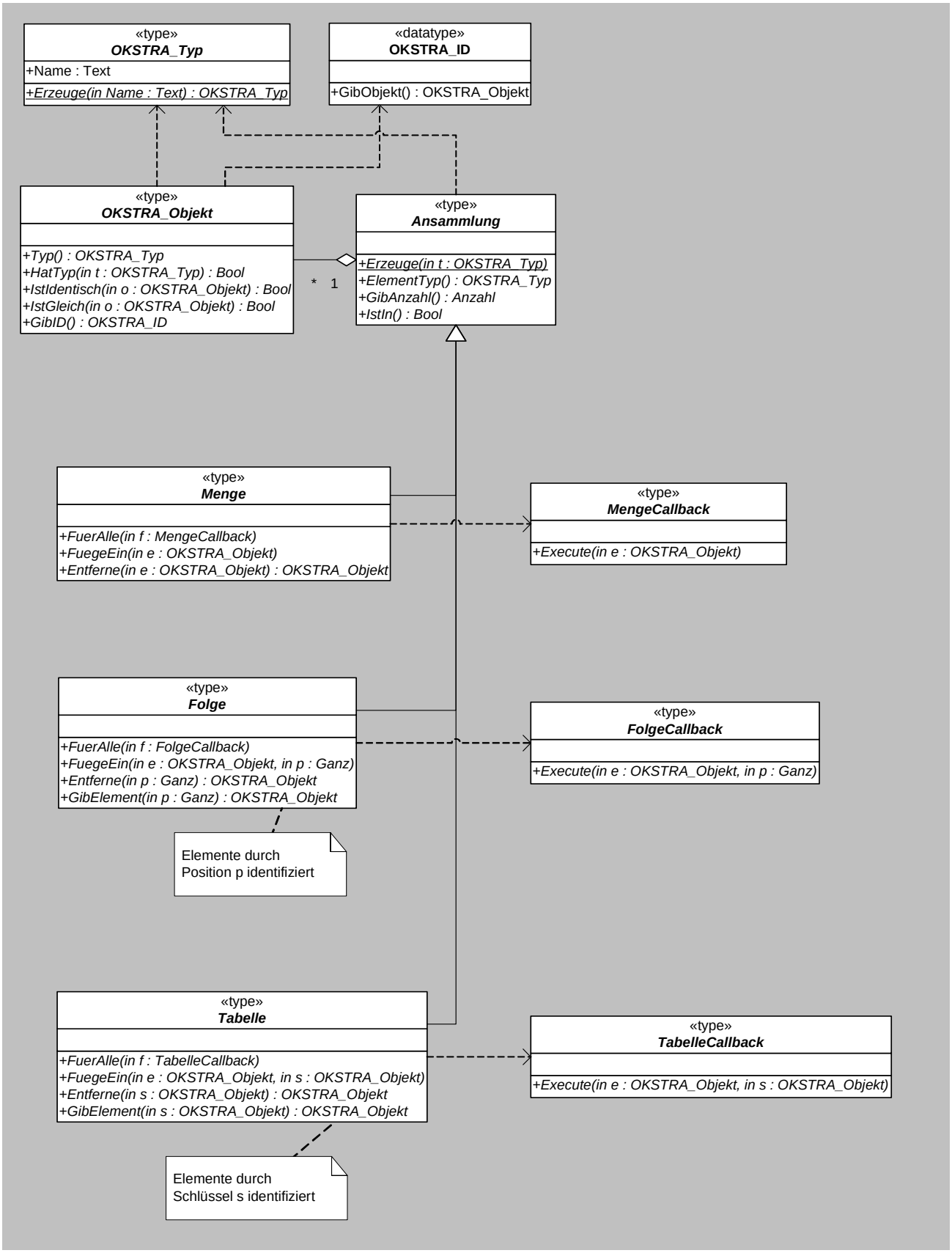


Abbildung 11. Ansammlungen, OKSTRA-Typ, OKSTRA-Objekt.

7.5 Verzeichnisse, Kataloge

Für viele Aufgabe werden dienstebasierte Systeme Verzeichnisse und Kataloge bemühen, um festzustellen, wo sich bestimmte Objekte oder Dienste befinden. Dies hat zu einer Reihe von Standards geführt, die Grundlage für eine OKSTRA-Modellierung von Verzeichnis-Objekten sein sollten:

- The Internet Engineering Task Force (IETF)

  Lightweight Directory Access Protocol (LDAP) RFCs

  http://www.bind9.net/rfc-ldap
- Organization for the Advancement of Structured Information Standards (OASIS)

  Directory Services Markup Language v2.0

  http://www.oasis-open.org/committees/dsml/docs/DSMLv2.doc
- Organization for the Advancement of Structured Information Standards (OASIS)

  Universal Description, Discovery & Integration (UDDI)

  http://www.oasis-open.org/committees/uddi-spec/tcspecs.shtml#uddiv2

7.6 Ereignisse

Aktionen, die bestimmte Objekte betreffen, haben sehr oft auch Auswirkungen auf andere Objekte.

Dies ist z.B. bei Kostenermittlungen evident: Wird eine Trasse geändert, ändern sich die Baukosten und es sind andere Grundstücke tangiert. Vorhandene Kostenberechnungen (auch Grunderwerbspläne, LBP usw.) werden damit ungültig. Es ist daher wünschenswert, dass die Kostenberechnungssoftware (GE-Software usw.) von der Änderung automatisch in Kenntnis gesetzt wird.

Ereignisse bieten ein formales Rahmenwerk für solche Kommunikationen. (Siehe auch [Gam 95], Observer Pattern).

Ein wichtiges Anliegen ist, dass sich Objekte darüber informieren müssen, wenn sich etwas ergeben hat, das für andere Objekte von Belang ist. Hierzu gibt es ein Standardmodell, die *Ereignisbehandlung*. (Ereignisbehandlung findet sich auch in einigen modernen Programmiersprachen wieder. Das OKSTRA-Modell kann sich jedoch nicht darauf verlassen, sondern muss eine unabhängige Modellierung vorgeben.)

Objekte, die daran teilnehmen wollen, definieren zunächst die *Arten von Ereignissen*, die wichtig für sie sind, z.B. dass sich eine Trasse geändert hat. Das ergibt für jede Ereignisart einen eigenen Typ. Diese Ereignisklassen werden aus einem Basistyp *Ereignis* abgeleitet. Die Instanzen einer Ereignisklasse sind die tatsächlich ausgelösten Fälle, z.B. „Trasse OU Neuhaus geändert am 19.04.2000“, „Trasse A 888 geändert am 06.12.1997“.

Ereignisverarbeitung ist ein zentrales Anliegen bei der Automatisierung von Geschäftsprozessen, so dass sich mittlerweile einige Standardisierungsvorschläge herausgebildet haben:

- IBM, Akamai, HP, SAP, Sonic Software, The Globus Alliance, TIBCO

  Web Services Notification (WS-Notification), derzeitige Version 1.0

http://www-106.ibm.com/developerworks/library/ws-resource/ws-notification.pdf

- Hewlett-Packard

  Web Services Events (WS-Events) Version 2.0

  http://devresource.hp.com/drc/specifications/wsmf/WS-Events.pdf

- BEA, Microsoft, TIBCO

  Web Services Eventing (WS-Eventing)

  http://xml.coverpages.org/WS-Eventing200401.pdf

Eine Ereignismodell-Spezifikation sollte sich in Richtung eines (hoffentlich) konvergierenden Standards bewegen.

Um zu demonstrieren, was eine Ereignis-Signalübermittlung leisten soll, folgt hier ein Beispiel, dass die Grundzüge eines jeden solchen Systems enthält. Der folgende Ablauf für die Ereignisbehandlung dient als Ausgangspunkt für die entsprechende Modellierung.

> Für jede Ereignisklasse wird festgestellt, wo entsprechende Ereignisse ausgelöst werden können. „Trasse geändert"-Ereignisse werden typischerweise vom Entwurfssystem ausgelöst, generell sind aber alle Programme, die Geschäftsprozesse unterstützen, wo die Entwurfsgeometrie geändert wird, Kandidaten (Kennzeichnung EG im GP-Katalog).
>
> Ein Geschäftsbereich meldet die Art der Ereignisse, über die er informiert werden will, in einem Ereignisregister der Baumaßnahme an. Dazu muss er die Ereignis*klasse,* z.B. die Klasse TrasseGeändert, als Objekt übergeben. Um dies modellieren zu können, müssen wir die Ereignisklassen selbst als *Instanzen* einer Klasse auffassen können. Solche Klassen heißen *Metaklassen.* In unserem Fall ergibt das die Metaklasse *Ereignisklasse.*
>
> Wie werden nun die Ereignisse verarbeitet? Will z.B. das Entwurfssystem mitteilen, dass es die Trasse geändert hat, so erzeugt es ein TrasseGeändert-Objekt und schickt es der Baumaßnahme. Die Baumaßnahme schaut im Ereignisregister nach, welche Geschäftsbereiche davon benachrichtigt werden wollen. Den betroffenen Geschäftsbereichen vermittelt sie ihrerseits die Nachricht über das Ereignis weiter. Die Weiterverarbeitung ist dann Sache des angesprochenen Geschäftsbereichs. Für eine Kostenberechnung würde das z.B. bedeuten, dass die aktuell gültige Berechnung ungültig werden muss. Diese wird entsprechend gekennzeichnet. Bei der nächsten Anforderung einer Kostenberechnung würde diese dann zuerst neu berechnet werden. (Eine sofortige Neuberechnung ist unzweckmäßig, weil im Zuge der Änderung der Trasse noch weitere Folgeänderungen ausgelöst werden können.)

Im Endeffekt besteht das zugehörige Rahmenmodell aus folgenden Typen:

Ereignisklasse ist eine Metaklasse für die Typen, die Ereignisarten modellieren. Diese Typen werden von einem gemeinsamen Basistyp ***Ereignis*** abgeleitet. Die einzige Operation ist:

GibQuelle()

Antwortet mit dem Objekt, das das Ereignis ausgelöst hat.

Der Typ ***EreignisVerarbeiter*** fordert eine Operation

LöseAus(e)

Nimmt ein Signal an und führt die hierfür notwendigen Operationen in dem annehmenden Objekt aus.

EreignisVermittler sind spezielle ***EreignisVerarbeiter***.

RegistriereEreignis(e, v)

wird von einem *EreignisVerarbeiter* v aufgerufen, der über Ereignisse der Ereignisklasse e informiert werden möchte.

LöseAus sucht für jedes angenommene Ereignis-Signal die Ereignisverarbeiter heraus, die davon informiert werden möchten und übermittelt ihnen das Signal über ihre eigene *LöseAus*-Operation.

Das folgende Diagramm zeigt das beschriebene Ereignismodell. Man beachte, dass die Assoziation zwischen *EreignisVermittlern* und *EreignisVerarbeitern* dargestellt ist, obwohl sie nicht sichtbar ist, um den Ablauf verständlicher zu machen.

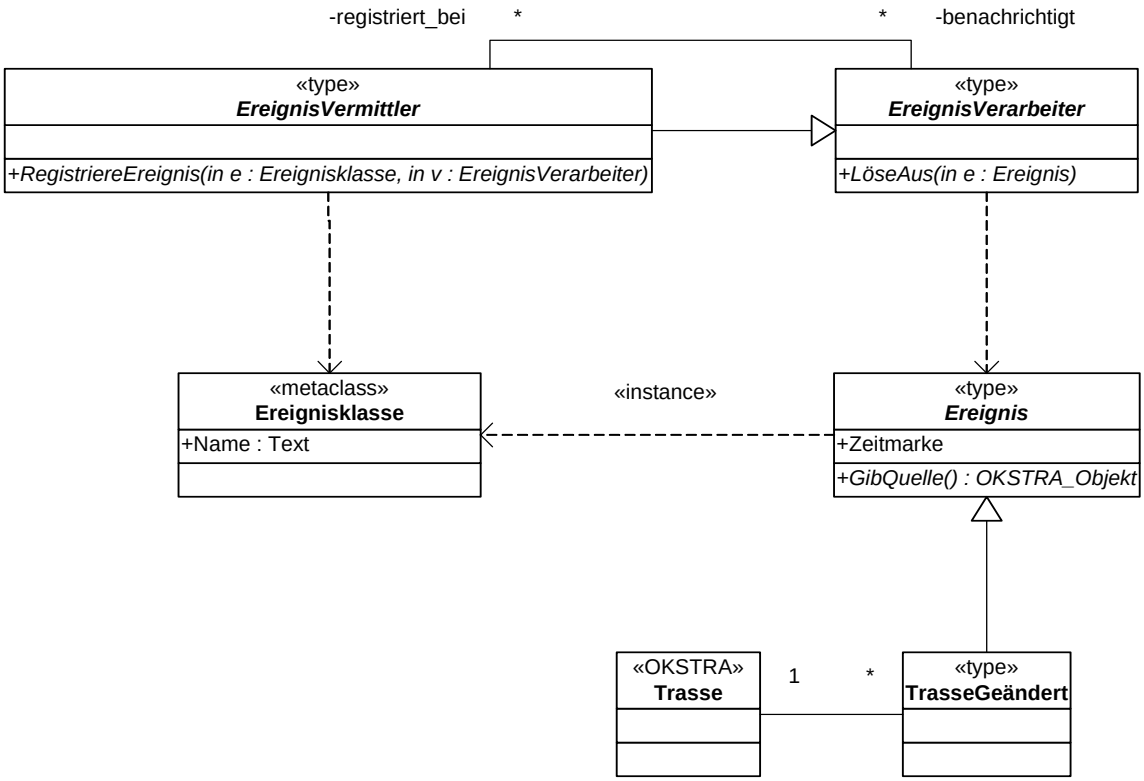


Abbildung 12. Klassendiagramm Ereignisse.

Um den Ablauf zu demonstrieren, folgt ein Sequenzdiagramm, dass Ereignisse im Modell der Kostenberechnung zeigt.

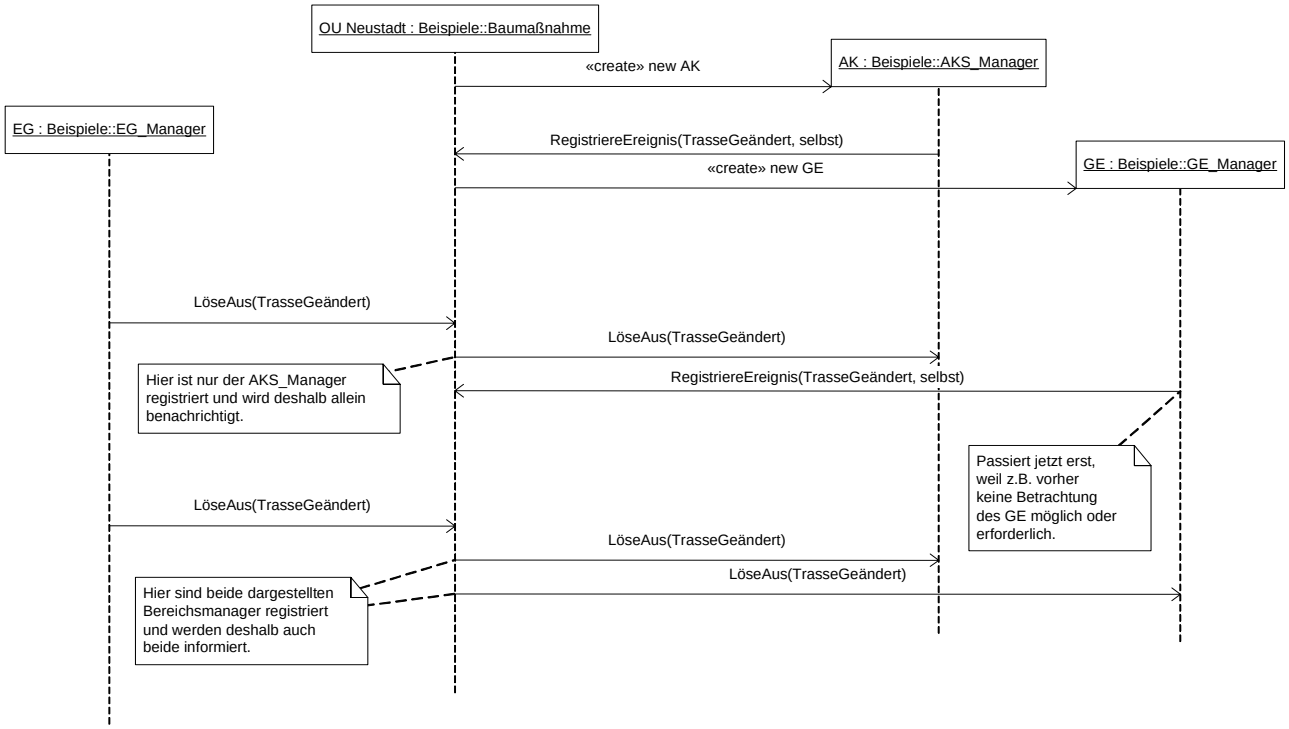


Abbildung 13. Übermittlung von Signalen für Ereignisse.

7.7 Fabriken

Objektorientierte Sprachen und andere Systeme bieten sehr unterschiedliche Mechanismen an, um Objekte zu erzeugen. Um hiervon zu abstrahieren, wird in der objektorientierten Modellierung das Entwurfsmuster *Fabrik* benutzt. Eine Fabrik ist ein Objekt, das andere Objekte erzeugen kann. Für das OKSTRA-Modell wird dieses Muster konsequent angewendet : Zu jedem instanzierbaren Typ gibt es genau eine zugehörige Fabrik-Klasse.

Ein Dienst, der Objekte einer Klasse A an seiner Schnittstelle als Ergebnisse anbietet, muss solche Objekte auch erzeugen können. Entweder ist die Fabrikfunktionalität daher im Inneren des Dienstes unsichtbar abgeschottet, oder es werden Fabrikoperationen auch an der Schnittstelle des Dienstes angeboten.

> Ein Beispiel: Wir entwerfen ein Objekt *Höhenlinie_Fabrik*, das eine Operation *Erzeuge* definiert, die eine Höhe als Zahlenwert und ein Digitales-Geländemodell-Objekt als Parameter annimmt. Es konstruiert aus diesen Parametern die Geometrie der Höhenlinie der verlangten Höhe im DGM und verpackt diese in ein Höhenlinien-Objekt.

Die Fabriken selbst müssen natürlich auch erzeugt werden. Um nicht in logische Widersprüche zu geraten, muss man daher zulassen, das einige dieser Fabrik-Objekte ständig zur Verfügung stehen, ohne dass andere Objekte sie erzeugen.

Typ ***OKSTRA_Fabrik***

Alle Typen, deren Objekte *OKSTRA_Objekte* erzeugen können, leiten aus diesem Typ ab

Erzeuge( ) : o

Erzeugt ein OKSTRA_Objekt. Falls zugelassen wird, dass eine Fabrik mehrere Sorten von Objekten erzeugen darf, kann *Erzeuge* parametrisiert werden.

Fabriken können dynamisch mit den Typen assoziert werden, zu denen sie Objekte erzeugen. Dafür dient eine Registrierungs-Operation.

Typ ***OKSTRA_Typ***

RegistriereTyp(name, f)

Registriert name als neuen instanzierbaren Typ und f als Fabrik-Objekt dafür.

GibErzeuger() : f

Gibt ein *OKSTRA_Fabrik*-Objekt für den Typ zurück.

7.8 Serialisierung

Nicht alle OKSTRA-konformen Software-Systeme werden als Komponenten verteilter Systeme verfügbar sein oder werden. Um mit ihnen weiterarbeiten zu können, ist daher der datei-orientierte Austauschweg beizubehalten. Für diese Anforderungen ist ein Mechanismus notwendig, der die Objekte in die bekannte austauschbare Form (z.B. CTE) bringt und entsprechende Dokumente wieder zurück in Objekte wandelt.

Typ ***Serialisierung***

Dies sind die Objekte, die die Externdarstellung repräsentieren. Die Ableitung aus *OKSTRA_Fabrik* fordert die *Erzeuge*-Operation. Diese stellt aus dem aktuellen Objekt das gewünschte *OKSTRA_Objekt* her.

Typ ***Serialisierbar***

Alle Typen, die serialisierbare Objekte beschreiben, leiten hieraus ab.

Serialisiere() : s

Stellt die Serialisierung s des Objektes her.

Typ ***DeserialisiererFabrik***

Alle Fabrik-Typen, deren Objekte eine Serialisierung s in das zugehörige *OKSTRA_Objekt* o verwandeln können, leiten aus diesem Typ ab.

Deserialisiere(s) : o

Deserialisiert s zu einem *OKSTRA_Objekt* o.

Der Grund für diese kompliziert aussehende Aufspaltung ist folgender: Beim inkrementellen Ausbau des OKSTRA kommen serialisierbare Klassen hinzu. Würde die Deserialisierungsoperation nun an das Serialisierungsobjekt gehängt, müsste man entweder eine Klassenhierarchie von Ableitungen daraus aufbauen, wobei jeder *OKSTRA_Objekt*-Ableitung eine entsprechende basierend auf *Serialisierung* entsprechen würde, oder die Methode zur Deserialisierungsoperation müsste bei jeder Modelländerung mitgeändert werden.

Bei dem vorgestellten Vorschlag wird die Deserialisierung in der zugehörigen Fabrik angesiedelt. Der ganze Vorgang beruht nun darauf, dass bei der Erzeugung des Serialisierung-Objektes diesem intern und unsichtbar der Typ des Ausgangsobjektes mitgegeben wird. Der Verständlichkeit des Modells halber ist dies als unsichtbares Attribut eingetragen. Bei *Erzeuge* wird dieser Typ benutzt, um über *OKSTRA_Typ* und *GibErzeuger* an ein Objekt zu kommen, das die *Deserialisiere*-Operation kann. Deren Ergebnis wird dann von *Erzeuge* abgeliefert.

7.9 Typsystem

Es ist zweckmäßig, jeden instanzierbaren Typen des objektorientierten OKSTRA durch je ein Objekt einer Klasse *OKSTRA_Typ* zu beschreiben. Dieses Vorgehen hat Vorbilder in relationalen Datenbanken, wo die Datenbankschemata selbst in Tabellen hinterlegt sind, sowie in modernen objektorientierten Programmiersprachen wie C++ mit dem Run Time Type Interface RTTI, sowie die Reflection-Klassen in Java und C#.

Ein voll ausgebautes Typsystem gestattet es, ein Objekt nach seiner Typzugehörigkeit zu befragen und daraus zu ermitteln, welche Operationen für es zur Verfügung stehen. Dies ist für die OKSTRA-Modellierung deshalb wünschenswert, weil bei einer inkrementellen Modellierung innerhalb der Klassenhierachie zu einer Basisklasse regulär neue Ableitungen hinzukommen werden. Dynamische Typverwaltung macht es den Software-Entwicklern leicht, vorausschauend auf solche Entwicklungen zu reagieren und unnötige Eingriffe in getestete und laufende Systeme zu vermeiden.

Beispiele für die Möglichkeiten des Typsystems fanden sich in den vorigen Abschnitten.

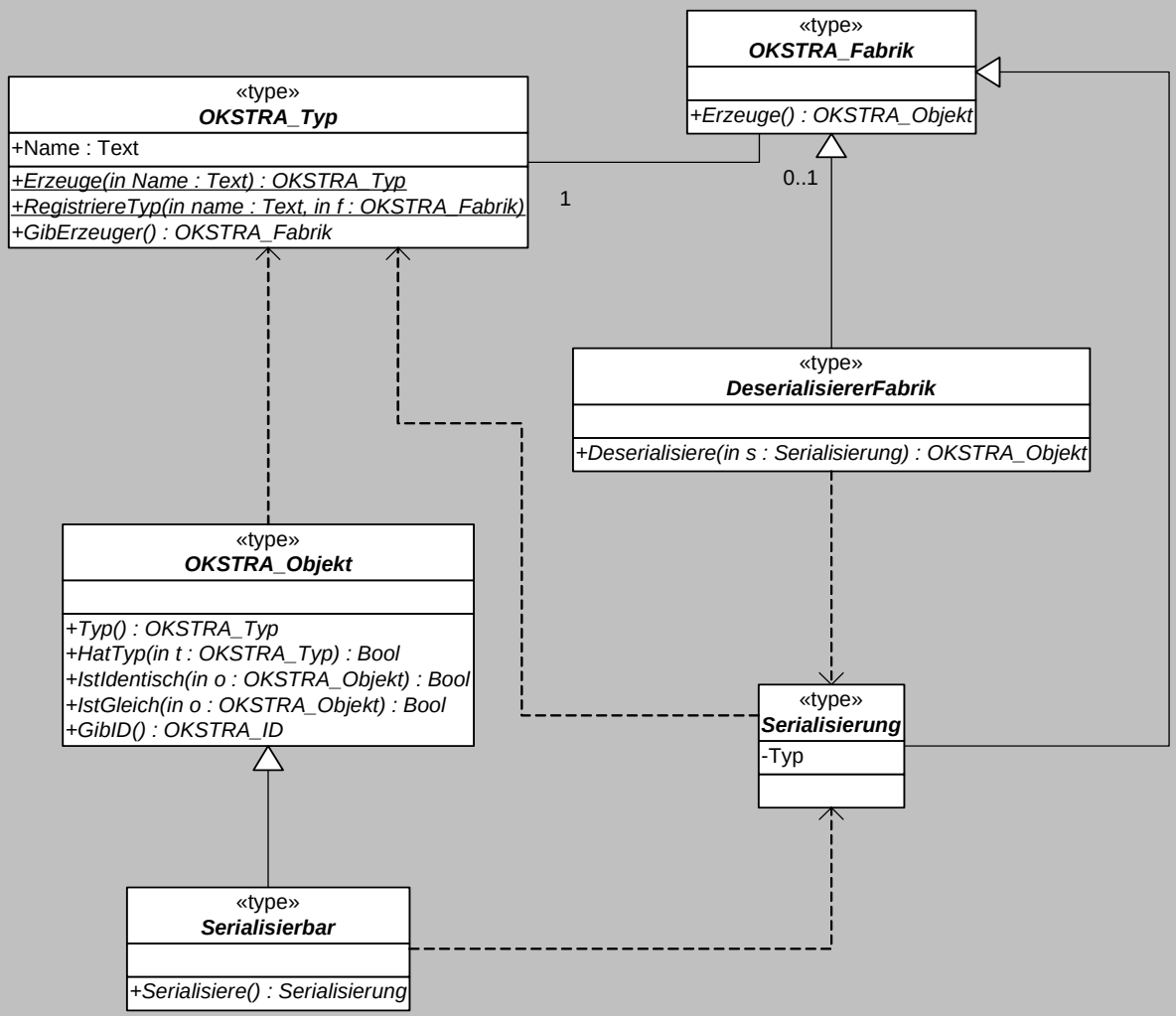


Abbildung 14. Fabriken, Serialisierung.

7.10 Sicherheit

Informationssysteme sind immer stärker Bedrohungen durch Viren, DDoS-Agenten (zur gezielten Lahmlegung von Servern und Netzen), Spyware, Fernsteuerungswerkzeuge usw. ausgesetzt. Daher sind ausgefeilte Sicherheitsarchitekturen mittlerweile unverzichtbar. Für verteilte Systeme auf OKSTRA-Basis gilt dies selbstredend auch. Wenn Komponenten wie z.B. Web Services innerhalb eines solchen Systems sicher kooperieren sollen, muss der OKSTRA eine entsprechende Sicherheitsarchitektur definieren. Diese muss sowohl Authentisierung – Prüfung der *Zugangs*berechtigung – als auch Autorisierung – Prüfung der *Zugriffs*berechtigung – umfassen. Auch wenn einige Dienste ausschließlich in Intranets betrieben werden, muss der Zugriff auf kritische Informationen reguliert und abgesichert werden. Um so mehr gilt dies bei Szenarien, die eine Informationsübermittlung zwischen den Instanzen für den Straßenbau und –betrieb und externen Dienstleistern vorsehen.

Das Gebiet ist technisch wie juristisch sehr umfangreich und erfordert Spezialkenntnisse zur Verschlüsselungstechnik. Daher kann das Thema nur angerissen werden. Außerdem kann eine Sicherheitsarchitektur nur im Konsens *aller* Anwender geschaffen werden kann, damit sich alle über Kosten, Restrisiken und Auswirkungen auf das Arbeitsumfeld im Klaren sind. Dazu müssen sich Fachanwender mit ihren fachlichen Sicherheitsbedürfnissen, Sicherheitsadministratoren mit den Sicherheitanforderungen ihrer IT-Abteilungen und Sicherheitsingenieure als Berater mit Technologiekompetenz zusammenfinden.

Hier folgt als Beispiel nur eine grobe Skizze für ein simples Authentisierungs-System.

> Wir benötigen eine Anmelde-Operation, die den Zugang zu einer Baumaßnahme erlauben oder verbieten kann. Dazu übergibt ein Client, z.B. eine bestimmte Entwurfssysteminstallation, der Maßnahme eine ClientID. Die Operation liefert bei erlaubten Zugriff ein Ticket-Objekt zurück, in dem verzeichnet ist, was dieser Client alles darf. Das Ticket muss gesichert sein gegen Verfälschung auf den Transportwegen (das können wir einer sowieso erforderlichen Verschlüsselungtechnologie überlassen), und außerdem müssen wir dafür sorgen, dass keine gefälschten Tickets erfolgreich verwendet werden können.
>
> Das Ticket wird allen Operationen, die eine Zugangskontrolle ausüben sollen, „vorgezeigt“, z.B. den Operationen, die Zugang zu einem Geschäftsbereich (d.h. den Objekten, die er verwaltet) erlauben.
>
> Eine Abmelde-Operation beendet eine Serie von Aktionen auf der Maßnahme. Es ist Sorge dafür zu tragen, dass bei einer unterbliebenen Abmeldung (z.B. durch Netzausfall) eine Abmelde-Automatik in Kraft tritt.
>
> Es wäre nun ziemlich lästig, bei jeder Fachdaten-Operation im Geschäftsbereich das Ticket prüfen zu müssen, andererseits muss man verhindern, dass ein Client sich ein tieferliegendes Fachobjekt merkt, um später unter Umgehung der Sicherheitsmechanismen darauf zuzugreifen. Der übliche Weg, damit umzugehen, ist die Einrichtung von Sessions (Sitzungen), in deren Kontext die Operationen durchgeführt werden.

Ein guter technischer Überblick ist [Smi 02].

Zur Modellierung einer Sicherheitsarchitektur hier nur wenige Anmerkungen.

Es gibt bereits eine ganze Anzahl erprobter und eingesetzter Verfahren zur *Authentisierung*. Der OKSTRA sollte die vorhandenen Architekturen nutzen können, ohne selbst das „Rad neu zu erfinden“. Hilfreich ist die Anlehnung an die Standards:

- Organization for the Advancement of Structured Information Standards (OASIS)

  SAML (Security Assertion Markup Language), derzeitige Version 1.1

  http://www.oasis-open.org)

- IBM, Microsoft, VeriSign

  Web Services Security (WS-Security), derzeitige Version 1.0

  http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

- Richtlinien des Bundesamtes für Sicherheit in der Informationstechnik (BSI)

  http://www.bsi.de

Im Gegensatz zu Authentisierung ist die Architektur zur Autorisierung (also *was* ein vorher authentisierter menschliche oder maschinelle Benutzer *womit* tun darf) eine vom Fachgebiet *nicht* unabhängige Angelegenheit. Typische, auch für objektorientierte Anwendungsszenarien nutzbare Modelle enthalten folgende Elemente:

- Rollen, die die Benutzer an Hand ihrer Befugnisse im Geschäftsbetrieb einteilen
- Rechte für den Zugriff auf folgenden Ebenen:
  - Teilbestände von Exemplaren, die mit Hilfe räumlicher, zeitlicher und fachlicher Kriterien definiert sind. Hierunter fallen auch die Grenzfälle Alle und Einzeln per Objekt-ID.
  - Operationen
- Die Zuordnung von Rollen und Rechten zueinander.

7.11 Persistenz

Persistenzoperationen werden von Diensten bereitgestellt, die Objekt-Inhalte dauerhaft speichern. Das klassische Muster dafür ist das CRUD-Muster (Create, Read, Update, Delete), wobei der Persistenzdienst die in Klammern genannten Operationen für einzelne Objekte oder Ansammlungen davon bereitstellt. Für die speicher-fortschreibenden Operationen (CUD) kann man sich die Frage stellen, ob der OKSTRA überhaupt ein Rahmenmodell vorgeben soll, denn vor der Speicherung steht die Prüfung der zu speichernden Inhalte, und das ist eine fachliche Aufgabe. Die CUD-Operationen werden daher nur *innerhalb* von fachspezifischen Fortführungsoperationen benötigt, sind deren Privatsache und müssen daher im OKSTRA keine Berücksichtigung finden.

Anders ist es mit dem Lesen. Anwendungen, die Auswertungen vornehmen, Abfragen an Abfragedienste richten können, die Objekte verschiedenster Art über Verknüpfungen und Bedingungen auswählen. Die Abfragen werden in Abfragesprachen ausgedrückt. Beispiele dafür sind:

- Open GIS Consortium (OGC)

  Filter Encoding Implementation Specification , derzeitige Version 1.0.0

  http://www.opengis.org/docs/02-059.pdf

- World Wide Web Consortium

  XML Query

  http://www.w3.org/XML/Query#specs

Hier sollte der OKSTRA die Struktur der Abfrageoperationen vorgeben.

7.12 Transaktionen

Es besteht ein nicht-triviales Problem, das nicht übersehen werden darf: Während der Durchführung von komplexeren Änderungen, die mehrere Objekte betreffen, hat man einen unsauberen, d.h. fachlich nicht unbedingt konsistenten Zwischenzustand in den Objekten: Müssen beispielsweise zwei Objekte in Abhängigkeit voneinander geändert werden, so ist ein konsistenter Zustand erst dann wieder erreicht, wenn beide Objekte modifiziert sind. Werden die Objekte nun während der Änderung befragt, kann es sein, dass die Information aus dem einen schon den neuen, die aus dem anderen aber noch den alten Stand wiedergibt.

Das Mittel zur Vermeidung solcher Probleme sind *Transaktionen* (eine Folge von Objektänderungen, die als atomare Änderung angesehen werden; d.h. vorher und nachher herrscht ein konsistenter Zustand und andere Leseoperationen vor Abschluss der Transaktion „sehen“ den Zustand vor Beginn der Transaktion).

Neben Transaktionen gibt es debn Begriff der *Business Activity*. Während bei Transaktionen die kombinierten Zustandsänderungen entweder alle komplett oder alle gar nicht in Kraft treten, können bei Business Activities auch Teilergebnisse gerettet und verwertet werden, oder gescheiterte Aktionen durch Umgehungsaktionen oder Stornierungen kompensiert werden.

Standardisierungsvorschläge für Koordination von Aktivitäten in Geschäftsprozessen gibt es mittlerweile diverse konkurrierende:

- BEA, IBM, Microsoft

  Web Services Security (WS-Security), derzeitige Version 1.0

  ftp://www6.software.ibm.com/software/developer/library/ws-coordination.pdf

  Web Services Atomic Transaction (WS-AtomicTransaction)

  ftp://www6.software.ibm.com/software/developer/library/ws-atomictransaction.pdf

  Web Services Business Activity Framework (WS-BusinessActivity)

  ftp://www6.software.ibm.com/software/developer/library/ws-busact.pdf

- Arjuna, Fujitsu, IONA, Oracle, Sun

  Web Services Transaction Management (WS-TXM) Ver1.0
  http://developers.sun.com/techtopics/webservices/wscaf/wstxm.pdf

Der Anhang B zeigt an einer praktischen Fragestellung die Wichtigkeit des Problems. Mangels eines manifesten Standards wurde dort ein eigenes Transaktionsmodell entwickelt.

7.13 Workflow

Workflows sind Schemata für automatisierte Prozessabläufe. Die Klassen eines Modells für die Abwicklung von Prozessen abstrahieren die für alle Arten von Prozessen gemeinsamen Eigenschaften, z.B.:

- Prozesszustand: In welcher Phase steht der Prozess gerade? Ist die laufende Aktivität beendet?

- Prozessfortschaltung: Welches Ereignis startet welche Folgeaktivität?
- Prozessressource: Wer oder was ist mit dem Prozess gerade beschäftigt?

Für Workflows gibt es mittlerweile eine ganze Anzahl von Standardisierungsansätzen. Einige Referenzen :

- Object Management Group (OMG): http://www.omg.org/docs/formal/00-05-02.pdf
- Workflow Management Coalition (WfMC): http://www.wfmc.org/standards/docs.htm
- Organization for the Advancement of Structured Information Standards (OASIS):

  http://www.oasis-open.org/committees/documents.php?wg_abbrev=business-transaction
- Electronic Business PML (ebPML): http://www.ebpml.org
- Business Process Management Initiative (BPMI): http://www.bpmi.org/

Außerdem haben BEA, Hewlett-Packard, IBM, Microsoft, und Sun entweder mit- oder nebeneinander Standardisierungsvorschläge publiziert:

- BPEL4WS: http://www-106.ibm.com/developerworks/library/ws-bpel/
- WSCL: http://www.w3.org/TR/2002/NOTE-wscl10-20020314/
- WSCI: http://www.w3.org/TR/wsci/

Eine kontinuierliche Beobachtung dieser Ansätze ist empfehlenswert. Angesichts der Vielfalt der existierenden Bemühungen ist ein eigener Weg des OKSTRA in diesem Bereich wenig erfolgversprechend. Der Autor geht davon aus, dass im Laufe der Zeit sich aus der Vielfalt Konvergenzen zu einem regulären Standard geben wird.

7.14 Raumbezug

Der Raumbezug wird im bestehenden OKSTRA durch ein konzeptionelles Geometrie-Schema beschrieben, das in der Realisierung als XML-Schema durch das Objektmodell der GML 3.0 (Geography Markup Language) abgedeckt wird.

Standardisierte Schnittstellen für Dienste für raumbezogene Objekte wurden und werden vom Open Gis Consortium (OGC), http://www.opengis.org , definiert.

7.15 Historisierung

Im bestehenden OKSTRA gibt es ein Modell für bestimmte historisierbare Objekte.

Für ein *allgemeines* objektorientiertes Historisierungsmodell führt man eine Klasse ***OKSTRA_Hist_Objekt*** ein mit folgenden Operationen:

IstHistorisch()

Die Operation liefert den Wert WAHR für ein historisches Objekt, FALSCH für ein aktuelles Objekt.

ZustandsDatum():d

Liefert das Datum d , zu dem der Zustand das letzte Mal aktuell war.

GibHistorischenZustand(d:Datum):o

Die Operation liefert ein historisches Objekt im Zustand zum Datum d. Es können auf diesem Objekt erfolgreich nur Operationen durchgeführt werden, die den Objektzustand nicht ändern.

> Auch wenn das aktuelle Datum eingesetzt wird, kommt ein nicht veränderbares Objekt zurück! Der Nutzer der Operation muss sich darauf verlassen können, dass die Eigenschaft der Nichtveränderbarkeit unabhängig vom Datum ist.

Man kann zusätzlich eine Ableitung ***OKSTRA_Tau_Objekt*** einführen, die eine weitere Operation erlaubt:

TaueAufHistorischenZustand(d:Datum):o

Überträgt die historischen Objektzustände in das zugehörige aktuelle Objekt.

Die konkreten Klassen, die aus *OKSTRA_Tau_Objekt* abgeleitet werden, müssen eine Semantik besitzen, die solch eine Operation auch zulässt. Problematisch ist bei der Semantik dieser Operation, wie weit der Auftauprozess über verknüpfte Objekte fortzusetzen ist. Dies kann möglicherweise nur auf der Ebene der konkret aus dieser Klasse abgeleiteten Klassen entschieden werden.

Anmerkungen:

Historische Objekte sind identisch, wenn sie Vorgänger desselben aktuellen Objektes zum selben Zeitpunkt sind.

Eine „negative Historisierung“, also Objekte mit prognostizierten Zuständen, ist ebenfalls denkbar.

> Die Ermittlung der Eigenschaften einer solchen Klasse sei als kleine Übung empfohlen. Hinweis: Was passiert mit einem Prognose-Exemplar, wenn das aktuelle Datum das Prognosedatum des Exemplars erreicht?

7.16 Dokumente

Hier soll nur die Frage aufgeworfen werden, in wie weit der OKSTRA ein Objektmodell für Metadaten analoger und digitaler Dokumente spezifizieren sollte.

> Zu Metadaten gehören z.B. Angaben über die Erstellung (Person, Ort, Zeit), Format, Verbleib, Zweck, Identifikation.

8 Roadmap zur Modellierung des objektorientierten OKSTRA

Dieses Kapitel zählt die Schritte auf, die für die Modellierung des objektorientierten OKSTRA erforderlich sind.

8.1 Pflege des Leitfadens

Zunächst ist dieser Leitfaden zu diskutieren, zu ergänzen und verbindlich zu verabschieden. Dazu und zur späteren Pflege und Weiterentwicklung des Leitfadens sollte eine Arbeitsgruppe gebildet werden.

An Ergänzungen sind zunächst festzulegen:

8.1.1 Benennungsregeln für Objekte und Operationen

Verwendung von Groß- und Kleinschreibung: Sind Groß- und Kleinbuchstaben namensunterscheidend oder nicht? (Falls nicht, werden z.B. die Schreibungen NULLPUNKT, Nullpunkt und NullPunkt als identische Namen angesehen.)

Verwendung von Zahlen, Umlauten und Sonderzeichen: Welche sind erlaubt und wo dürfen sie in Namen vorkommen?

Wie sind die Bezeichner des existierenden OKSTRA auf die neuen Namensregeln abzubilden? (Wie werden aus Attributen und Relationen die Namen der zugehörigen Akzessoren gebildet?)

8.1.2 Festlegung der UML-Version

Vorgeschlagen wird die Version 2.0, deren Erhebung zum OMG-Standard noch 2004 erfolgen soll.

8.1.3 Festlegung der Dateiformate für die Publikation der Modelle

Vorgeschlagen werden PDF für druckfähige Dokumente, XMI für den Austausch zwischen CASE-Systemen und SVG für web-fähige Präsentation.

8.1.4 Festlegung der unterstützten Sprachbindungen

Näheres siehe 6.4. Die Festlegung sollte einvernehmlich mit Vertretern der Software-Industrie abgestimmt sein.

8.1.5 Evaluierung und Empfehlung von UML-Modellierungs-Werkzeugen

Siehe hierzu die Ausführungen in 6.4. Werden unterschiedliche Werkzeuge für die konzeptionelle Arbeit mit den Expertengruppen einerseits und die Detailmodellierung zur Publikation andererseits empfohlen, so ist mindestens die XMI-Schnittstelle zum Datenaustausch zu fordern.

8.1.6 Regelungen für den Review-Prozess

Das Review der Modelle bekommt für den objektorientierten OKSTRA eine größere Bedeutung, weil ja die Modelle zur Konstruktion ausführbarer Software, Programmen, eingesetzt werden. Daher ist besonderer Wert auf Regelkonformität, Konsistenz und technische

Umsetzbarkeit zu legen. Folglich sollte der Leitfaden ein praktikables, formales Vorgehensmodell für die Planung und Durchführung der Qualitätssicherung der Modelle beinhalten. Praktikabilität bezieht sich dabei auf die notwendigen Ressourcen (Personal, Mittel für die Erstellung von Prototypen usw., Zeitdauer). Einige Anhaltspunkte sind in 2.8 zu finden.

8.2 Entwicklung der Rahmenmodelle

Die in Kapitel 7 angegebenen Modelle sind über den dort vorgegebenen exemplarischen zu Rahmen hinaus zu entwickeln. Hierbei sollte, wo immer möglich, auf internationale offene Standards zurückgegriffen werden. Das Kapitel gibt auch Hinweise zur gegenwärtigen Standardisierungslage.

8.3 Fachliche Modellierung

In Anbetracht der Vielzahl der im Straßen- und Verkehrswesen anfallenden Tätigkeiten ist vor der fachlichen Modellierung eine Bestandsaufnahme der Geschäftsbereiche sinnvoll, mit einer Beurteilung, welchen Nutzen eine Automatisierung für die Akteure des Bereichs haben würde. Dieser Leitfaden kann keine erschöpfende Anleitung für eine solche Analyse geben, sondern nur Stichworte:

Höhere Aktualität der Informationen?

> Z.B. durch kürzere Pflegezyklen

Höhere Zuverlässigkeit der Informationen?

> Z.B. durch Benachrichtigung über Änderungen der Datengrundlage

Schnellere Verfügbarkeit der Informationen?

> Z.B. durch entfallende Arbeitsschritte bei der Datenaufbereitung

Geringere System-Infrastruktur-Aufwände?

> Z.B. durch vereinfachte oder entfallende Client-Installationen

Geringere Datenpflege-Aufwände?

> Z.B. durch zentrale Datenhaltung

Integrationsfähigkeit mit bestehenden IT-Lösungen?

> Z.B. durch Nutzung von vorhandenen und offengelegten Komponenten-Schnittstellen

Vereinfachung von Tätigkeiten?

> Z.B. durch Automatisierung schematischer und routinemäßiger Abläufe

Analysen verschiedener öffentlicher und privatwirtschaftlicher Organisationen des Straßen- und Verkehrswesens werden zu unterschiedlichen Ergebnissen kommen und müssen daher zunächst harmonisiert werden. Der Aufbau des objektorientierten OKSTRA sollte aber zuerst den Bereichen zu Gute kommen, die den größten Nutzen davon haben.

9 Literaturverzeichnis

[Boo 99] Booch,G., Rumbaugh,J., Jacobson,I., The Unified Modeling Language User Guide. Addison Wesley. 1999.

[Fow 00] Fowler,M., Scott,K. UML Distilled, Second Edition. Addison Wesley. 2000

[Gam 95] Gamma, E., Helm,R., Johnson,R., Vlissides,J., Design Patterns. Addison Wesley. 1995

[GP-Neu 02] Geschäftsprozesskatalog zum Prozess Neubaumaßnahme. 2002. http://www.okstra.de/ookstra.html

[Oes 98] Oestereich, B., Objektorientierte Softwareentwicklung. Oldenbourg. 4. Aufl. 1998

[Oes 03] Oestereich, B., Objektorientierte Geschäftsprozessmodellierung. dpunkt.verlag. 2003

[Oes 04] Oestereich, B., Objektorientierte Softwareentwicklung. Oldenbourg. 6. vollst. überarb. Aufl. 2004

[Smi 02] Smith,R.E., Authentication. Addison Wesley. 2002

[UML 03] OMG Unified Modeling Language Specification Version 1.5. 2003

[UML-I 04] OMG UML 2.0 Infrastructure Specification. 2004

[UML-S 04] OMG UML 2.0 Infrastructure Specification. 2004

[UML-DI 03] OMG UML 2.0 Diagram Interchange Specification. 2003

[UML-O 03] OMG UML 2.0 Object Constraint Language Specification. 2003

Anhang A
Modellierung der Kostenermittlung

Verantwortlich für den Inhalt:

Dipl.-Phys. Bernd Weidner
interactive instruments GmbH, Bonn

Redaktion: Bernd Weidner (interactive instruments)

A.1 Einleitung

Dieses Dokument beschreibt das objektorientierte Modell für den Geschäftsprozess Kostenermittlung und, aufbauend darauf, einen als verteiltes System realisierten Prototypen dafür. Die methodischen Erkenntnisse, die sich aus der Modellierung des Prozesses ergaben, bilden die Grundlage des Leitfadens zur objektorientierten Modellierung des OKSTRA.

A.1.1 Vorbemerkungen

Im Folgenden ist das Originaldokument zur Modellierung unverändert wiedergegeben. Die Referenzen auf Standards und Literatur wurden nicht aktualisiert, um den ursprünglichen Quellenstand getreu wiederzugeben.

Zum Verständnis des Dokumentes sind erforderlich:

- Wissen um Bedeutung und Einsatz des AKS-Prozesses innerhalb des Planungsprozesses. Eine Beschreibung hierzu findet sich im *Geschäftsprozesskatalog* (GP-Neu 02).
- Kenntnis der (AKS 95) und ihrer Ergänzung (ARS 13/90)
- Kenntnis des *Leitfaden zur objektorientierten Modellierung des OKSTRA (LF Modell 02)*.

Für das Verständnis der UML-Komponentendiagramme, wie sie in der Beschreibung des Prototypen verwendet werden, wird auf die einschlägige UML-Literatur, z.B. (OesOOSE 98, Boo 99) verwiesen.

A.1.2 Motivation

Nach Aufstellung des Geschäftsprozesskataloges (GP-Neu 02) stellte sich die Frage, welcher Prozess sich besonders für eine weitergehende Analyse eignen würde. Dass die Wahl schließlich auf die Kostenermittlung fiel, hat fachliche und modellierungstechnische Gründe. Die folgende Liste dieser Gründe gibt keine Prioritäten wiederge; es ist ihre Gesamtheit, die es attraktiv erscheinen lässt, gerade diesen Bereich weiter zu untersuchen:

- Der Prozess hat offensichtlich große wirtschaftliche Relevanz.
- Er ist prozedural gut durch Regelwerke erschlossen, so etwa durch die „Anweisung zur Kostenberechnung von Straßenbaumaßnahmen (AKS 85)“, die HOAI und den Standardleistungskatalog STLK.
- Die Durchführung einer Kostenermittlung ist trotz Software-Hilfsmitteln langwierig und schematisch. Aktualisierungen werden daher nur bei zwingender Notwendigkeit durchgeführt. Die existierende Software erlaubt zwar durchaus eine recht komfortable Verwaltung der Projekte, Pflege von Preisdokumentationen, regionalisierte Leistungskataloge mit Freitexten usw. Die wirklich zeitaufwändige Aktivität, nämlich das Auffinden und Berechnen der benötigten Informationen aus den Planunterlagen bleibt weitgehend händische Arbeit. Es fehlen Datenbrücken z.B. zu den Entwurfssystemen. Eine Automatisierung des Prozesses könnte zu verbesserter Aktualität der Kostendaten und Einsatz gewonnener Arbeitskapazität für kreative und organisatorische Aufgaben führen.

- Das eigentliche Berechnungsverfahren bei einer Kostenberechnung ist täuschend einfach: Es wird eine Summe von Positionen „Menge x Preis“ gebildet. Die Modellierung des Prozesses kann sich daher statt auf algorithmische Verfahren auf Fragen der Organisation der Fachinformation und des Informationstransports konzentrieren. Gerade in diesen Bereichen können objektorientierte Verfahren ihre Wirksamkeit besonders gut entfalten.

  Anmerkung: Das Problem der Mengenbestimmung selbst ist sicher nicht gering einzuschätzen. Die hier angestrebte Modellierung kann jedoch davon abstrahieren, wenn sie bei den kostenrelevanten OKSTRA-Objekten die Existenz einer Operation zur Mengenermittlung *voraussetzt.* Für die tatsächliche Realisierung ist diese Operation natürlich auszufüllen, d.h. als Methode(n) zu implementieren, was durchaus kompliziert sein kann.

- Die Kostenermittlung bezieht sich zwangsläufig auf viele Nachbarprozesse innerhalb des Gesamtablaufs einer Maßnahme, da sie von überall die preislich zu bewertenden Fachobjekte benötigt. Damit wird sie zu einem geeigneten Kandidaten für die Automatisierung durch ein verteiltes System. Ein Kostenberechnungssystem würde z.B. Entwurfssysteme, Planungssoftware für die Landschaftspflege, die Software für den Grunderwerb sowie andere Kostenberechnungs-Systeme, z.B. für Bauwerke, „anzapfen“, die irgendwo anders und auf ganz anderen Computern installiert sind.

- Bereits der bestehende OKSTRA® behandelt die Kostenberechnung nach AKS (OKSTRA 00, Teilbericht B, 3.5.1.1, S. 107). Dort wird allerdings nur ein Verweis auf die elektronischen Dokumente modelliert: „Auf die explizite Angabe von Formaten wird verzichtet, da diese im Verlaufe weniger Jahre erheblichen Änderungen unterworfen sein können.“ Diese Begründung trifft für eine statische Modellierung ganz sicher zu. Die Möglichkeiten eines objektorientierten Modells bestehen aber gerade darin, die Abhängigkeit von Formaten komplett zu eliminieren: Die in den Objekten vorliegende Information kann ja durch *Operationen* in beliebige Form gebracht werden.

- Im Gegensatz zu der etwas summarischen Behandlung der Kostenermittlung (AKS- und AVA-Prozesse) im bestehenden OKSTRA® liefert eine vertiefte Analyse eine neue Sicht auf die Welt der Fachobjekte. Bestehende Objektarten müssen vermutlich „angereichert“ werden, z.B. durch zusätzliche Attribute; der übergreifende, vieles berücksichtigende Charakter einer Kostenermittlung wird fehlende Fachobjektarten aufdecken helfen.

- Da Kostenermittlungen einerseits unmittelbar etwas mit der Herstellung realer Objekte, z.B. Straßendecken oder Rohrleitungen zu tun haben, andererseits vor deren Existenz auf Planunterlagen zurückgreifen müssen, kann ihre tiefere Analyse auch etwas zur Überbrückung der z.T. differierenden Sichten der Bereiche Neue Daten und Bestand des OKSTRA® beitragen.

A.1.3 Überblick über den Inhalt

Die Untersuchung beginnt mit einer Analyse der Anwendungsfälle, die im Geschäftsprozess Kostenermittlung gefunden wurden. Zur Analyse wurden verwendet:

- die grundlegenden Regelwerke (HOAI, HVA F-StB, AKS, VOB, STLK, RAB-BRÜ)
- ergänzende Literatur hierzu
- Beispiele für Kostenberechnungen und Ausschreibungen

- Beschreibungen, Testversionen und Demonstrationen einiger Programme zum Thema (KOSTRA®, AKSWIN, ASTRA, ARCHITEXT)
- Last not least: Befragung von Fachleuten, insbesondere für das Vorgehen beim Grunderwerb

Danach wird ein Objektmodell für ein System zur Kostenberechnung nach AKS vorgestellt und durch Klassen- und Sequenzdiagramme erläutert.

Es folgt die Beschreibung des Prototypen

- in Form von Szenarien für einzelne Anwendungsfälle
- in Form eines Komponentendiagramms
- in Form eines Verteilungsplans

Ein zusammenfassender Ausblick stellt am Ende die aus Modellierung und prototypischer Realisierung gewonnenen Einsichten dar.

A.2 Die Anwendungsfälle der Kostenermittlung

A.2.1 Definition des zentralen Anwendungsfalles *Kostenermittlung*

Wie andere Geschäftsprozesse auch, tritt der Prozess *Kostenermittlung* an mehreren Punkten des Gesamtablaufs einer Baumaßnahme auf. Er wird hier als vom Anfang bis zum Ende der Maßnahme durchlaufender Prozess betrachtet, der in bestimmten Phasen aktiviert wird, dort zu einem Ergebnis kommt und dann bis zur nächsten Aktivierung ruht. Der Vorteil dieser Betrachtungsweise ist, dass es dabei unerheblich ist, wie oft und wann der Prozess wieder aufgenommen wird.

Der zentrale Anwendungsfall des Prozesses, nämlich die mehr oder weniger detaillierte Angabe der Kosten, die die Baumaßnahme verursacht, wird ebenfalls mit dem Namen ***Kostenermittlung*** belegt. Je nach vorhandener Datengrundlage kommen dabei unterschiedliche Verfahren und Regelwerke zum Tragen, so dass Kostenermittlung ein abstrakter Anwendungsfall ist, der durch unterschiedliche konkrete Anwendungsfälle verwirklicht wird.

Um diese zu fassen, bietet sich die folgende Definition HVA F-StB 2.1 (2) vorgenommen (HVA F-StB 99) an:

„(2) Die HOAI sieht für die Berechnung der anrechenbaren Kosten unterschiedliche Kostenermittlungsarten (Kostenschätzung, Kostenberechnung, Kostenanschlag, Kostenfeststellung) vor, die sich im wesentlichen durch den dem jeweiligen Planungsstand entsprechenden Genauigkeitsgrad unterscheiden.

Gemäß § 17 AVB-ING werden folgende Kostenbegriffe unterschieden:

Vorläufige Kostenannahme	= grob überschlägige Ermittlung der Gesamtkosten anhand entsprechender Erfahrungswerte oder typisierender Kennwerte
Kostenschätzung	= überschlägige Ermittlung der Gesamtkosten auf Grund von Erfahrungswerten (i.d.R. Ergebnis der Leistungsphase 2)
Kostenberechnung	= Ermittlung der angenäherten Gesamtkosten im Zuge der Entwurfsbearbeitung auf Grund der im Einzelnen ermittelten Mengen und der zugehörigen Einzelkosten (i.d.R. Ergebnis der Leistungsphase 3)
Kostenanschlag	= Ermittlung der tatsächlich zu erwartenden Gesamtkosten durch die Zusammenstellung von Auftragnehmerangeboten, Eigenberechnungen sowie anderen für die Baumaßnahme bereits entstandenen Kosten (i.d.R. Ergebnis der Leistungsphase 7)
Kostenfeststellung	= Nachweis der tatsächlich entstandenen Kosten des Bauvorhabens auf Grund von Abrechnungsbelegen (i.d.R. Ergebnis der Leistungsphase 8*)*.“

Von diesen werden in das Modell die letzten vier als Anwendungsfälle aufgenommen. Für die *Kostenannahme* gibt es nach Aussagen der befragten Fachleute keine bindenden Regeln oder Verfahren, daher ist eine formalisierte Umsetzung in ein IT-gerechtes Modell als

automatisierbare Aufgabe nicht möglich. Das schließt nicht aus, dass Kostenannahmen als Attribute bei der Definition einer Maßnahme z.B. im BVWP dokumentiert werden, ein entsprechender Anwendungsfall gehört dann aber in den Bereich Bedarfsplanung.

A.2.2 Die konkreten Anwendungsfälle zur Kostenermittlung

Für die verschiedenen Kostenermittlungsarten sind zz. zwei unterschiedliche und nicht harmonisierte Sätze von Regelwerken zuständig:

Während der Planung ist nach den Regularien der (AKS 85) samt erläuterndem Rundschreiben (ARS 13/90) zu verfahren. Dies gilt besonders für Kostenschätzungen während der Vorplanung und die für den Vorentwurf nach RE aufzustellende Kostenberechnung. Auch während der Bauvorbereitung und Baudurchführung werden noch Berechnungen auf dieser Grundlage durchgeführt, namentlich im Falle von Kostenfortschreibungen sowie bei Änderungen der Kostenteilungsvereinbarungen zwischen den Baulastträgern.

Kostenschätzungen während der Vorplanung werden üblicherweise in vereinfachter Form vorgenommen (Kosten je km, je m^2, je Knotenpunkt usw.(ARS 13/90)).

Die ***Kostenberechnung*** für den Vorentwurf hat sich auf den Kostenberechnungskatalog (KBK) der (AKS 85) zu stützen.

Kostenanschläge und die endgültige ***Kostenfeststellung*** gehen dagegen von einer ganz anderen Datengrundlage aus, nämlich Leistungsbeschreibungen, die im Rahmen der Ausschreibungs- und Vergabeprozesse angefertigt werden. Die hierfür grundlegenden Regelwerke sind die VOB sowie die Standardleistungskataloge STLK.

Die durch diese Zweigleisigkeit entstehende Problematik wird sehr transparent in (KBK-STLK 00) erläutert. Naturgemäß spiegelt sie sich auch in den Anwendungsfalldiagrammen wieder.

Für Überprüfungen, d.h. Vergleiche der Kosten einer Maßnahme mit den dafür vorgesehenen Ansätzen, sind Kosten auf aktualisierter Datengrundlage zu ermitteln. Rechentechnisch fallen hier keine neuen Anwendungsfälle an. Als Ergebnis einer Überprüfung der Kosten ist u.U. eine *Kostenfortschreibung* notwendig, d.h. die aktualisierten Kosten führen zu einer Aktualisierung der Haushaltsansätze. Diese Aktivität wurde nicht in das Anwendungsfalldiagramm einbezogen. Sie ist wesentlich haushaltstechnischer bzw. haushaltsrechtlicher Natur und gehört in Geschäftsbereiche, die nicht nur mit einer einzelnen Maßnahme zu tun haben (Kosten und Leistungsrechnung, Controlling).

Zum Projektcontrolling für die Einzelmaßnahme gehört jedoch die Verfolgung der Kosten während des Projektablaufs. Der hierfür wichtigste Anwendungsfall ist der ***Kostenvergleich***. Die Vergleichsmöglichkeit wird explizit von (AKS 85) verlangt. Da der Vergleich die bei den Kostenermittlungen berechneten Daten benötigt, wurde er als Anwendungsfall in das Kostenermittlungsmodell aufgenommen.

A.2.3 Anwendungsfälle zur Maßnahmenkonfiguration

Sowohl die (AKS 85) als auch die §4 VOB/A fordern (nicht unbedingt zwingend) eine Aufteilung der Gesamtmaßnahme. Die AKS fordert eine räumliche Gliederung, die VOB eine Aufteilung in räumliche Teile und fachliche Einheiten (*Teillose* und *Fachlose*). Beide Aufteilungsmuster sind konzeptionell vollkommen unabhängig voneinander. Sie werden durch zwei spezialisierte Anwendungsfälle der Maßnahmenkonfiguration, ***Lose bilden*** und ***Räumlich gliedern gem. AKS***, abgedeckt. Diese erzeugen die *Konfigurationseinheiten.*

Dieser Begriff ist als Oberbegriff, und zwar nur für die Zwecke des hier beschriebenen Modells, für die AKS-Teile bzw. die Lose zu verstehen. Eine Konfigurationseinheit fasst die fachlichen Informationsträger (z.B. Trassenobjekte) für die Zwecke einer bestimmten Kostenermittlungsart zusammen. Die Bildung dieser Zusammenfassung ist als gemeinsamer Kern beider genannten Anwendungsfälle mit ***Maßnahme konfigurieren*** bezeichnet.

Konkret hat man sich das etwa so vorzustellen, dass z.B. im Anwendungsfall *Räumlich gliedern...* räumliche Bereiche für die verschiedenen AKS-Teile definiert werden. Diese Bereiche werden dann in *Maßnahme konfigurieren* als räumliches Auswahlkriterium benutzt, um die darin liegenden Fachobjekte (z.B. Rohrleitungen) in Objektmengen zu sammeln, die der wesentliche Bestandteil der AKS-Teil-Objekte sind. Bei der Kostenberechnung selbst werden dann die AKS-Teile später separat und nacheinander abgearbeitet.

Die dargestellten Anwendungsfälle formalisieren die nach Aussagen der Fachleute zz. gängige Arbeitsweise. Wie gesagt ist, sind die Begriffe *Maßnahmenkonfiguration* und *Konfigurationseinheit* nicht durch irgendwelche Regelwerke abgedeckt, sondern nur innerhalb dieses Modells als Hilfsbegriffe eingeführt.

A.2.4 Die Kostengliederung

Zunächst ergibt sich eine Gliederung der Gesamtkosten für das Projekt in ***Grunderwerbskosten*** und ***Baukosten*** (AKS 85). Diese Kostenanteile sind von dem oder den Baulastträger(n) aufzubringen. Daneben stehen die ***Honorare*** für ***Ingenieurleistungen***, die aus den Verwaltungsetats der Straßenbauverwaltungen bzw. –betriebe finanziert werden. Die Ermittlung der Honorare ist daher ein eigenständiger Anwendungsfall, der jedoch wegen der Regelungen der HOAI (HOAI 96) und des (HVA F-StB 01) unmittelbar mit dem der Baukostenermittlung verknüpft ist. Diese Verknüpfung geschieht über den Anwendungsfall ***Anrechenbare Kosten ermitteln***. In der Praxis sieht das z.B. so aus, dass über die genannten Regelungen aus den Baukosten die anrechenbaren Kosten als Bemessungsgrundlage berechnet werden.

Der Anwendungsfall *Honorare Ingenieurleistungen* bestimmt dann aus den anrechenbaren Kosten und der Honorarzone die Honorare.

Sowohl der KBK der (AKS 85) als auch der STLK ist in *Leistungsbereiche* strukturiert. Die Baukostenermittlung kann daher in die Ermittlung der Kosten für jeden der Leistungsbereiche zerlegt werden.

Im Anwendungsfalldiagramm (Abb. 16) ist das dadurch ausgedrückt, dass die ***Leistungsbereichskosten*** als Anwendungsfall in eine Reihe von Spezialfällen aufgegliedert ist, bei denen u.U. auch weitere Regelwerke zum Tragen kommen. (Namentlich ist das bei den Bauwerkskosten der Fall, wo die (RAB-BRÜ 92) zu berücksichtigen ist.)

Die im Anwendungsfalldiagramm dargestellten Spezialfälle der Leistungsbereiche, also z.B. ***Erdbau, Entwässerung, Landschaftsbau*** usw. sind exemplarisch zu verstehen. Die in Abb. 16 gezeigte Aufgliederung wurde von den Mitgliedern des AK 9.7.1 im Rahmen des UML-XML-Workshops im August 2000 angeregt. Die endgültige Festlegung der Aufteilung in Leistungsbereiche sollte neben der Strukturierung insbesondere des STLK die horizontale Gliederung der Geschäftsprozesse in Geschäftsbereiche, die noch vorgenommen werden muss, berücksichtigen.

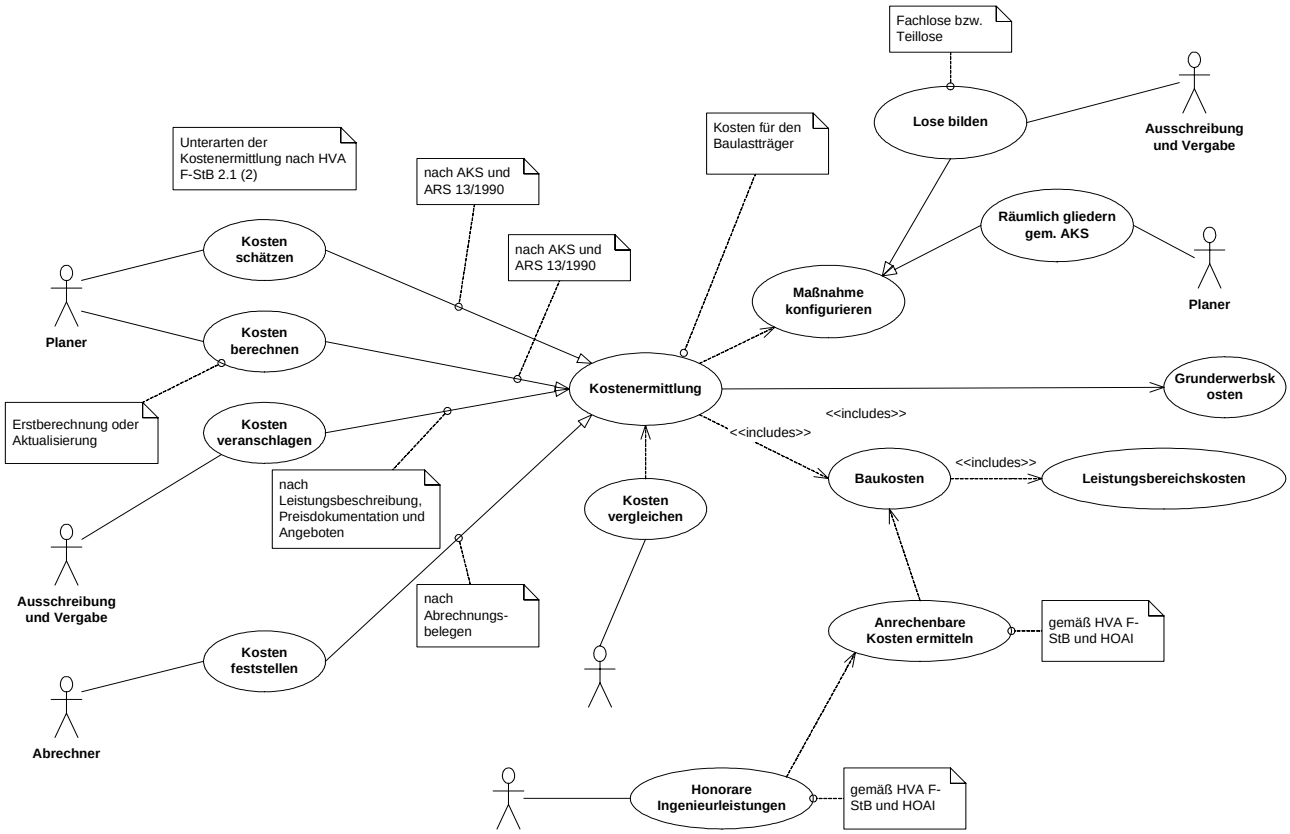


Abbildung 15. Anwendungsfälle der Kostenermittlung.

Leistungsbereichskosten ist ein abstrakter Anwendungsfall, der im Diagramm nur dazu dient, die den genannten Spezialfällen gemeinsamen Aufgaben übersichtlich darzustellen.

A.2.5 Mengen- und Preisbestimmung

Um Kosten zu ermitteln, werden *Mengen* und *Preise* benötigt. Sowohl die Baukostenermittlung als auch die Grunderwerbskostenermittlung *müssen* diese beiden Teilaufgaben lösen, die in den Anwendungsfalldiagrammen (Abb. 16 und 17) als ***Ermittlung der Preise, Ermittlung der Mengen*** und ***Ermittlung der Flächen*** eingetragen und über einen <<include>>-Pfeil als obligatorische Aufgaben der Kostenermittlung gekennzeichnet sind.

Anmerkung: Ein <<include>>-Pfeil drückt aus, dass der Anwendungsfall am Ziel des Pfeiles ein fester und immer notwendigen Bestandteil desjenigen an der Quelle ist.

Zur ***Baukostenermittlung***:

Die benötigten Preise werden entweder manuell erfasst oder aus maschinenlesbaren Dokumenten bezogen. Als solche kommen *Preisdokumentationen* in Frage oder *Angebotsdaten*. Der Anwendungsfall ***Preisdokumentation pflegen*** beinhaltet das Eintragen von Preisen samt ihrer Mengenabhängigkeit in einen entsprechenden Datenbestand. Die Pflege kann manuell oder automatisch aus Angeboten erfolgen. Existierende Programme zur Kostenermittlung (z.B. KOSTRA®, AKSWIN, PCASTRA, ARCHITEXT) bieten die Pflege einer Preisdokumentation an.

Preise manuell erfassen beinhaltet eine interaktive Eingabe von Preisen als Berechnungsgrundlage. Dies betrifft z.B. viele Pauschalpreis-Positionen der Leistungskataloge.

Angebotsdaten erfassen führt die tatsächlich für das Projekt angebotenen Preise in die Berechnungsgrundlage ein. Die Daten werden häufig schon maschinenlesbar geliefert.

Analog werden ***Mengen*** entweder ***manuell erfasst*** oder ***berechnet***. Die automatische Mengenberechnung verläuft nach Algorithmen, wie sie in (REB 97), (GAEB 99) dokumentiert sind oder nach Vereinbarung.

Anmerkung: Zur Durchführung von Mengenberechnungen bieten die Objekte, die die Mengen nachweisen müssen, Operationen an. Deren Implementation könnte die erwähnten Algorithmen nutzen oder auch eigene verwenden. In jedem Fall muss eine Implementation gewährleisten, dass ihre Ergebnisse mit der von Prüfberechnungen übereinstimmen.

Grundlage der Berechnung sind Entwurfsunterlagen oder Aufmaße. Auch bei der endgültigen ***Kostenfeststellung*** kann im Prinzip nach Plan abgerechnet werden, wenn die Entwurfsunterlagen während der Bauausführung entsprechend aktualisiert wurden. Die Zusammenhänge finden sich im folgende Anwendungsfalldiagramm wieder:

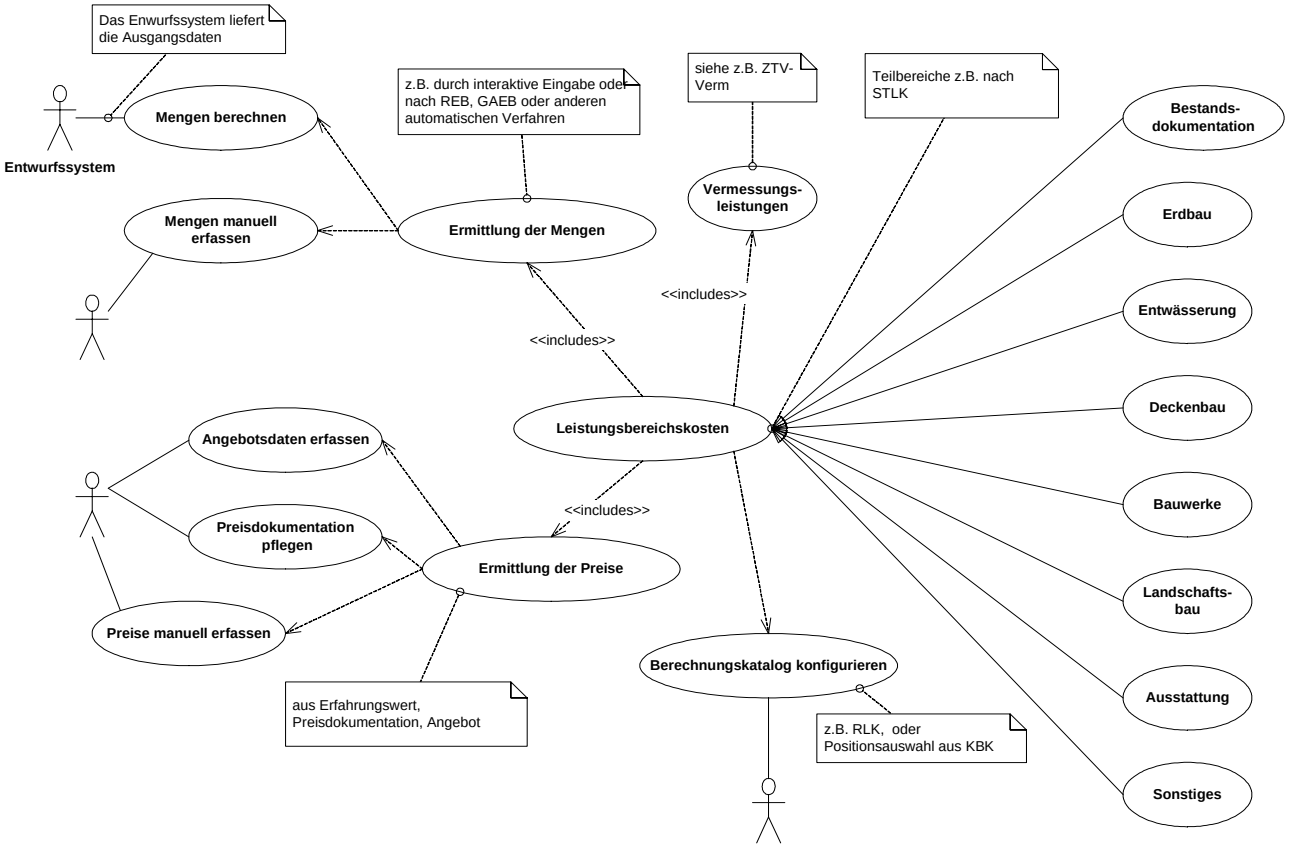


Abbildung 16. Baukosten.

Zur ***Grunderwerbskostenermittlung***:

Das Analogon zur Preisdokumentation bei der Baukostenermittlung ist im Teilbereich Grunderwerb(GE) die *Bodenrichtwertkarte.* Die Pflege dieser Karten ist nicht Aufgabe der Straßenbauverwaltung, es gibt also keinen entsprechenden Anwendungsfall. Dagegen ist die Tätigkeit ***Richtwert zuordnen*** zu einer Erwerbsfläche sicherlich ein Anwendungsfall. Neuerdings werden vielerorts Bodenrichtwertkarten digital z.T. flächendeckend zum Abruf

über einen Web Map Server zur Verfügung gestellt. (Siehe z.B. für das Land Niedersachsen http://www.gutachterausschuesse-ni.de/. Eine Suche nach „Digitale Bodenrichtwertkarte“ im Internet mit der Suchmaschine Google ergab Anfang Juni 2002 71 Treffer.). Das Zuordnen der Richtwerte kann prinzipiell also auch automatisch erfolgen.

Nach Abschluss der GE-Verhandlungen liegen die aktuellen Bodenpreise vor und müssen dann den Erwerbsflächen zugeordnet werden. Dies ist als Anwendungsfall ***Preis eintragen*** im Diagramm für die Grunderwerbskostenermittlung eingetragen. Beide Anwendungsfälle sind Möglichkeiten der ***Ermittlung der Preise*** für Erwerbsflächen.

Die zu betrachtenden Mengen sind für den GE die Größen der Erwerbsflächen. Diese werden je nach Projektphase auf unterschiedliche Art gewonnen: Während der Vorplanung wird die gesamte zu erwerbende Fläche z.B. durch einen Ansatz (Länge der Linie x Breite) geschätzt. Für den Vorentwurf können exakte Erwerbsflächen durch Schnitt der Entwurfsfläche mit den betroffenen Flurstücken gewonnen werden. Dies wird oft als Dienst des verwendeten Entwurfssystems zur Produktion von Grunderwerbsverzeichnissen und –plänen angeboten (***Flächen berechnen***). Nach der Schlussvermessung sind dann die neuen Flurstücke, die durch die Maßnahme erzeugt wurden, samt ihrem Flächeninhalt genau bekannt.

Es folgt ein UML-Anwendungsfalldiagramm zu den Grunderwerbskosten.

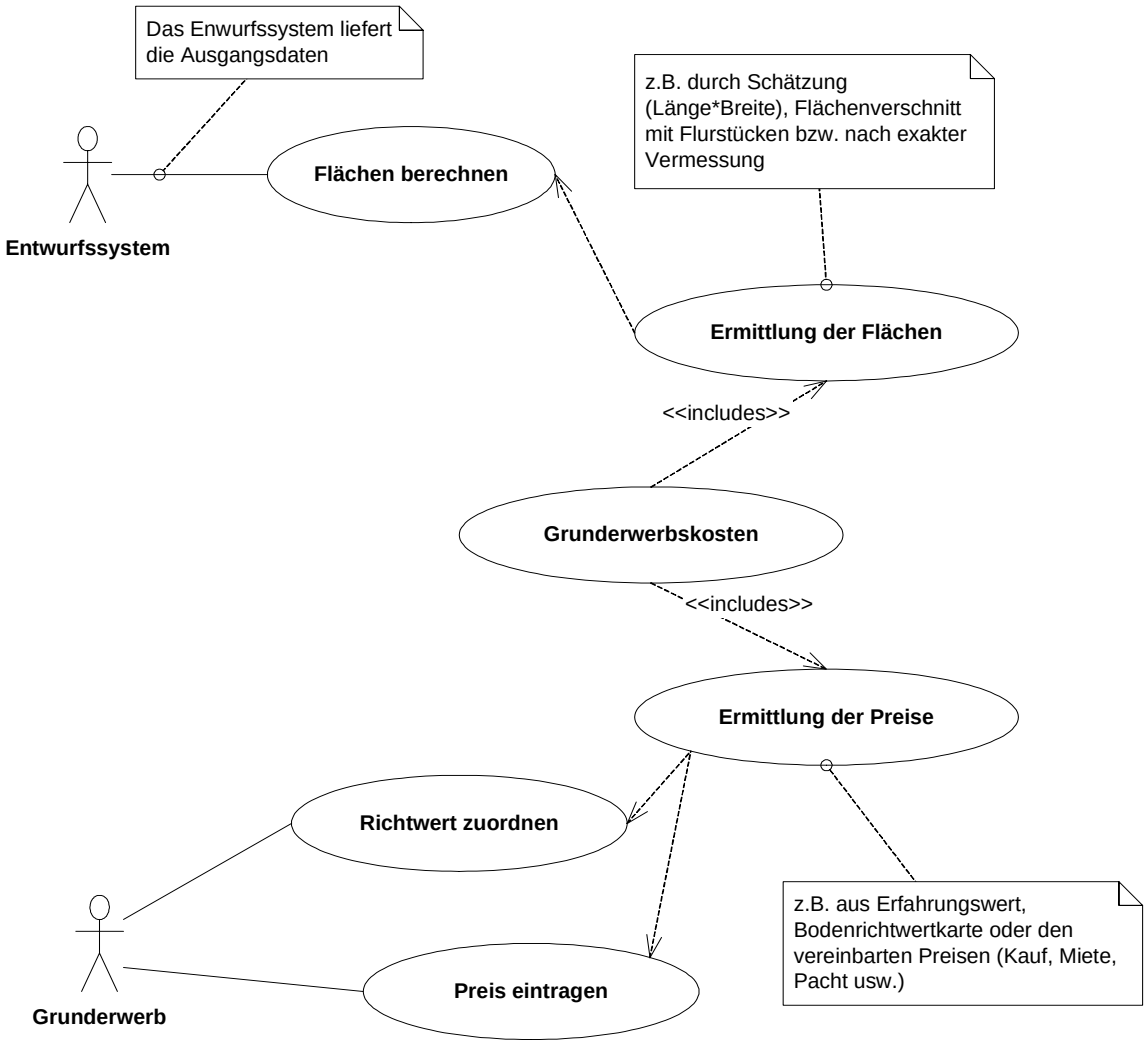


Abbildung 17. Grunderwerbskosten.

A.3 Das Objektmodell

A.3.1 Vorbemerkungen

In diesem Kapitel wird das aus der Anwendungsfallanalyse hergeleitete Objektmodell präsentiert, also die aufgefunden Klassen und ihre Operationen. Die hier präsentierten Modelle richten sich nach den Regularien aus dem *Leitfaden zur objektorientierten Modellierung des OKSTRA (LF Modell 02).* Insbesondere sei auf die Regelungen bezüglich Assoziationen und Akzessoren hingewiesen: Angeschriebene Rollennamen an den Assoziationen mit Sichtbarkeit *public* (Kennzeichen +) sind operational zu verstehen, d.h. fordern auf der Quellseite der Assoziation entsprechende Akzessoren, auch ohne dass diese explizit als Operationen angeführt werden.

A.3.2 Maßnahmen

Die Anwendungsfälle der Kostenberechnung sind immer im Kontext der Maßnahme, also des konkreten Bauprojektes, zu sehen, in dem die Kosten entszehen. Alle für eine Kostenberechnung (und für andere Geschäftsprozesse) benötigten Objekte werden daher auf irgendeine Weise einen Bezug zu der Maßnahme aufweisen müssen, in der sie auftauchen. Es ist daher sinnvoll, sich zunächst mit der Modellierung der Maßnahmen selbst zu beschäftigen.

A.3.2.1 Baumaßnahme

Die Einführung von *Baumaßnahmen* als Objekte einer entsprechenden Klasse ergibt sich zwanglos aus ihrem Charakter als (z.B. durch die PROJIS-Nr.) identifizierbaren und instanzierbaren Konstrukten mit Lebenslauf von der Bedarfsfeststellung bis Abbruch oder Fertigstellung. Das gezeigte Modell ist insofern vereinfacht, als die Unterscheidung zwischen Neubau- und Ausbau-Maßnahmen fehlt. Man würde dies berücksichtigen, indem Ableitungen *Neubaumaßnahme* und *Ausbaumaßnahme* von *Baumaßnahme* eingeführt würden und Baumaßnahme zu einer abstrakten Klasse würde. Der wesentliche Unterschied zwischen beiden Arten von Baumaßnahmen – nämlich dass die Straße, an der gebaut wird, bereits existiert oder nicht – ist in diesem exemplarischen Modell ohne Belang.

Baumaßnahmen haben Zustände, die z.B. nach den Regelungen für den PROJIS-Datenaustausch modelliert sein könnten. Beispiel: *Bearbeitungsstand* – etwa *UVE*: *UVS bzw. Variantenuntersuchung abgeschlossen, PA: Planfeststellungsverfahren beantragt* usw. Das vorliegende Modell nimmt hier keine genaue Modellierung vor (und damit vorweg). Diese muss durch ein entsprechendes Expertenteam erarbeitet werden, das die umzusetzenden Regelwerke benennt und evtl. Ergänzungen dazu einbringt. Die erforderlichen Attribute sind verkürzt mit ***PROJIS-Info*** bzw. ***Zustand*** gekennzeichnet.

Die Baumaßnahme-Objekte übernehmen folgende Verantwortlichkeiten:

- Portalfunktion für den Zugang zu allen Objekten, die zur Baumaßnahme gehören. Damit ist ein Authentisierungsmechanismus verbunden, der hier angedeutet wird, ohne ein Modell für die noch zu schaffende, endgültige OKSTRA®-Sicherheitsarchitektur zu präjudizieren. Die Portalfunktion wird realisiert durch die Ableitung aus dem Typ ***Pforte***, sowie eine Operation ***GibManager()***, die den Zugang zu den Diensten der ver-

schiedenen Geschäftsbereiche eröffnet. Die Portalfunktion wird hier angesiedelt, um zu erzwingen, dass ein Zugriff auf die Objekte der Maßnahme nur den Anwendern möglich ist, die für die Maßnahme auch zuständig sind.

- Namensraumfunktion, um eindeutige Identifikatoren (Kennzeichen) für die OKSTRA®-Objekte dauerhaft zur Verfügung zu stellen, so dass Objekte, z.B. Querschnitte, unterschiedlicher Maßnahmen voneinander unterscheidbar werden. Da Baumaßnahmen selbst eindeutig identifizierbar sind, bietet es sich an, dort eine entsprechende Operation einzuführen, die von den Factories der Objekte (s.u.) verwendet werden kann. Diese Operation bildet aus dem Identifikator der Baumaßnahme (z.B. der PROJIS-Nr.) und einem generierten Schlüssel ein eindeutiges Kennzeichen. Hierzu dient die Ableitung vom Typ ***ID_Server***, der diese Operation unter dem Namen ***GibID()*** bereitstellt.
- Es ist Ereignisvermittler für Ereignisse, die während des Ablaufes der Maßnahme auftreten. Siehe hierzu den *Leitfaden (LF Modell 02)*.

Es folgt das UML-Klassendiagramm.

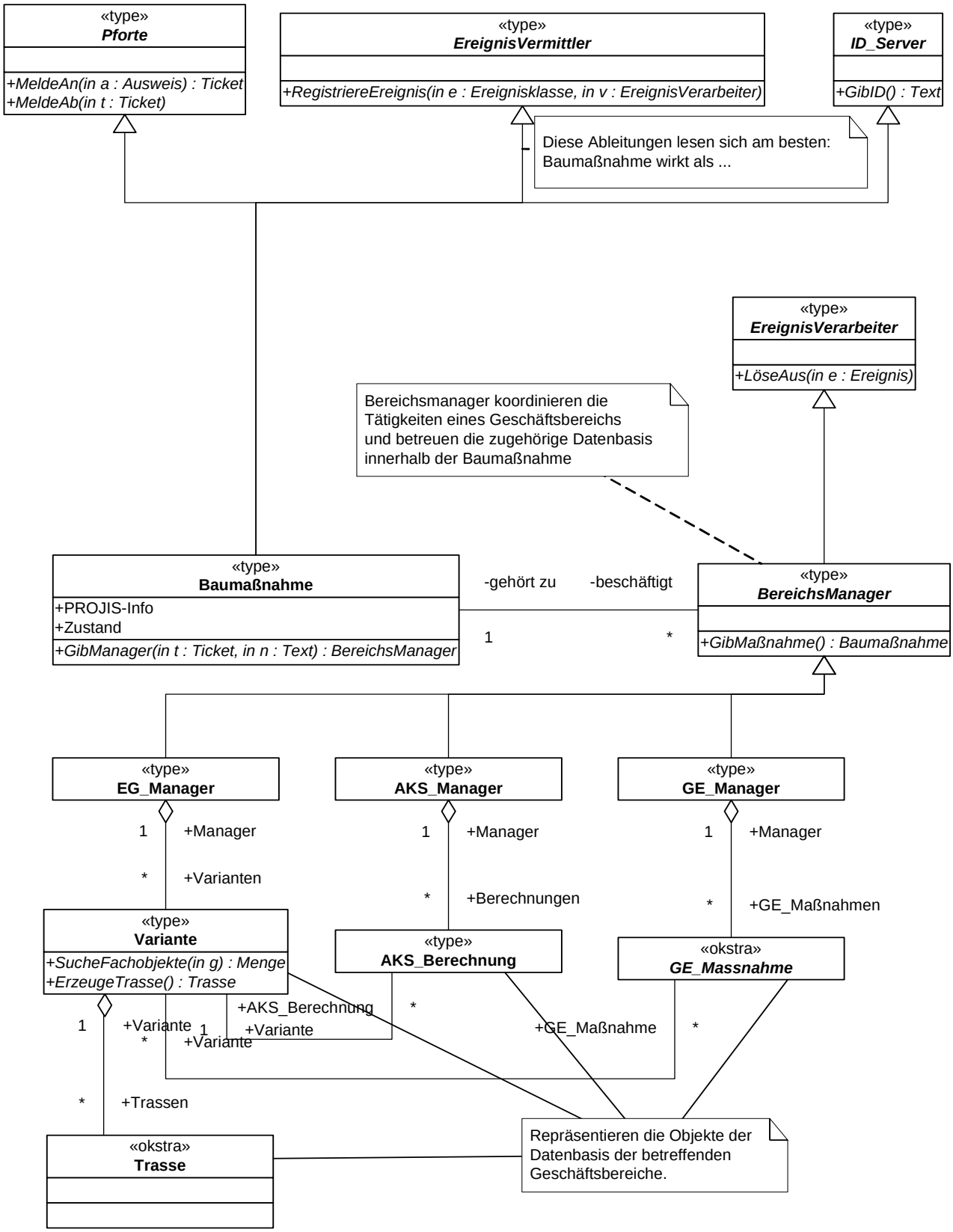


Abbildung 18. Baumaßnahme und Geschäftsbereiche.

Für das rudimentäre Sicherheitsinterface folgt ebenfalls ein UML-Diagramm.

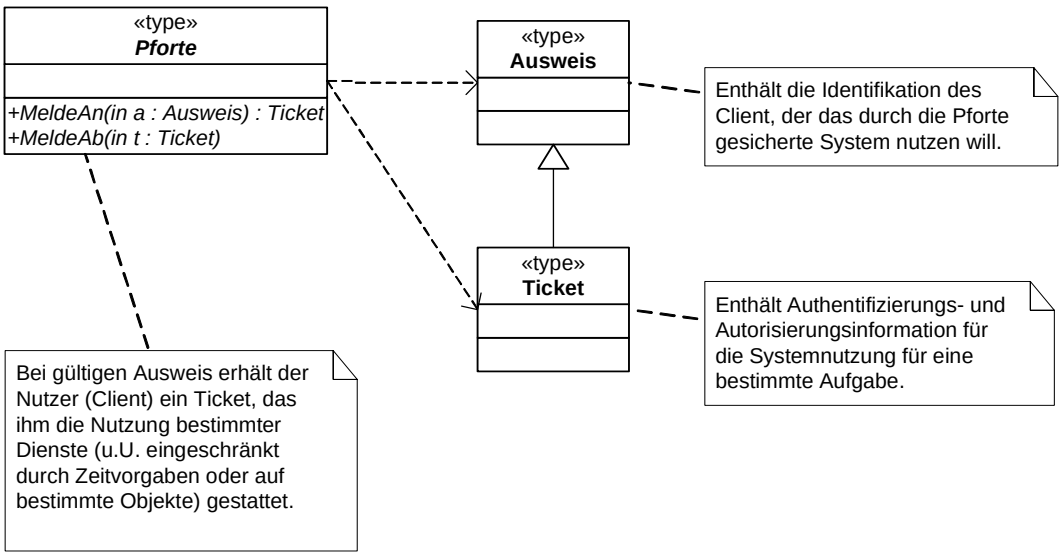


Abbildung 19. Prototyp Sicherheitsmodell.

Die Operation ***GibManager(ticket, bereich)*** gibt bei gültigem Ticket das Bereichsmanager-Objekt (s.u.) zurück, das den Zugang zum Bereich *bereich* kontrolliert.

A.3.2.2 Baumaßnahmenverzeichnis

Wie findet man eine Baumaßnahme? Dazu benötigt man offensichtlich ein *Verzeichnis aller Maßnahmen*. Die genaue Struktur dieses Verzeichnisses ist offen gelassen, es bietet sich jedoch eine *hierarchische*, *verteilte* Lösung an. Hierarchische Verzeichnisse sind jedem PC-Benutzer als *Ordner* bekannt. Bei einem verteilten Verzeichnis befinden sich die Ordner auf verschiedenen Servern.

Werden z.B. auf einer Straßenbauortsinstanz Daten zu einer Baumaßnahme benötigt, so wird erst im eigenen Teil des Verzeichnisses gesucht. Findet sich dort nichts, wird die Suche an die Zentrale delegiert. Dort besteht das Verzeichnis aus Einträgen für die Unterverzeichnisse aller Ortsinstanzen, so dass nun nacheinander die Server aller anderen Ortsinstanzen befragt werden können, Führt dies zu nichts, könnte die Suche weiter „nach oben“ delegiert werden, z.B. zum zuständigen Bundesministerium usw. usf.

Anmerkung: Das ursprüngliche Vorbild für derartige Verzeichnisse ist das Domain Name System DNS, mit dem die Adressen im Internet verwaltet werden.

Auch die Verzeichnisstruktur für Universal Description, Discovery, and Integration (UDDI) (http://www.uddi.org/), die angestrebte Architektur für die Registrierung von kommerziellen Web Services, ist verteilt.

Das UML-Diagramm hierzu:

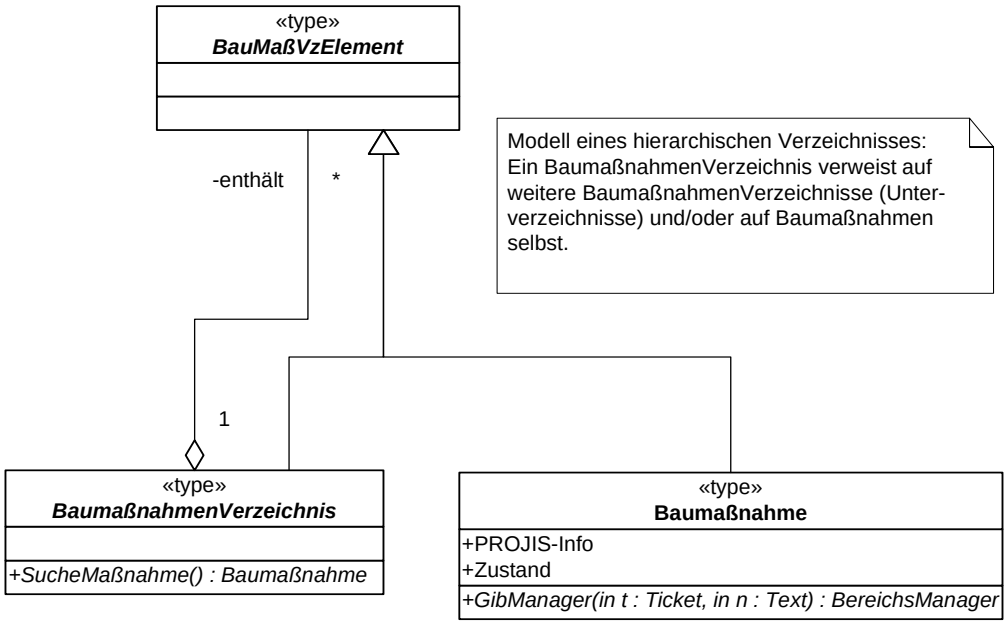


Abbildung 20. Baumaßnahmenverzeichnis.

Die Operation ***SucheMaßnahme()*** ist ohne Parametrisierung notiert (konzeptionelles Modell). Bei der Erstellung des Spezifikationsmodell ist die Parametrisierung nachzutragen, z.B. in Form eines *Suchkriterium*-Objektes, das die Suchkriterien zum Auffinden einer Baumaßnahme kapselt. Was die technischen Details der Beschreibung einer Maßnahme und einer durchsuchbaren Verzeichnisstruktur für Maßnahmen betrifft, können die Spezifikationen für UDDI und Resource Description Framework (RDF) Anregungen geben,

Neben dieser Operation muss das Verzeichnis sicher auch noch weitere Funktionen unterstützen, wie z.B. das Eintragen einer neuen Maßnahme, das Entfernen einer beendeten Maßnahme usw. Die entsprechenden Operationen sind hier weggelassen, weil unklar ist, welche Geschäftsprozesse bzw. Teilprozesse für die Pflege des Verzeichnisses verantwortlich sein sollen und wie diese Pflege darin ablaufen soll.

A.3.2.3 Bereichsmanager

Die GP-Analyse hatte ergeben, dass sich alle Prozesse bestimmten Geschäftsbereichen zuordnen lassen. Die für das Modell wichtigen Charakteristika eines Geschäftsbereichs sind:

- Pflege einer eigenständigen Datenbasis mit zunehmender Detaillierung und Genauigkeit
- Zugriff auf diese Datenbasis nur bei entsprechender Zuständigkeit
- Kommunikationsfähigkeit mit anderen Geschäftsbereichen nach Bedarf, und strikt reglementiert

Bei der Kostenermitlung sind exemplarisch drei Geschäftsbereiche vertreten, *Entwurfsgeometrie (EG)*, *AKS-Berechnung* und *Grunderwerb(GE)*. Dass die AKS als eigener Geschäftsbereich geführt wird und nicht Teil eines allgemeinen *Kostenmanagement*-Bereiches ist, gibt die derzeitige Situation wieder, in der praktisch kein Zusammenhang mit dem Bereich *AVA (Ausschreibung, Vergabe, Abrechnung*) sichtbar ist.

Jeder Geschäftsbereich wird durch einen *Bereichsmanager*-Typ repräsentiert. Der Basistyp selbst gestattet die Ermittlung der *Baumaßnahme*, die das Managerobjekt verwaltet.

Jeder Manager leistet wenigstens folgendes:

- Rückgriff auf das Maßnahme-Objekt
- Erzeugen einer neuen Instanz der Datenbasis (Factory-Funktion), also im Beispiel das Erzeugen einer neuen *Variante* oder *GE-Maßnahme*, oder einer neuen *AKS-Berechnung*
- Suchen einer Instanz der Datenbasis
- u.U. Vernichten einer Instanz der Datenbasis

Die Ausgestaltung dieser Operationen ist bereichsabhängig, sie muss durch die Bedürfnisse der nutzenden Geschäftsprozesse festgelegt werden. Dies wurde im Beispielmodell nur für den *AKS_Manag*er durchgeführt und wird im Folgeabschnitt beschrieben.

A.3.3 Die Modellierung zum Anwendungsfall Kostenberechnung

Der Anwendungsfall Kostenberechnung richtet sich nach der (AKS 85). Die Modellierung berücksichtigt neben den dort geforderten Konstrukten und Verfahrensweisen auch gängige Praxis, wie sie sich u.A. in einschlägiger Software (KOSTRA®, AKSWIN) wieder findet.

A.3.3.1 AKS_Manager

Der Typ, der den Bereichsmanager für die AKS formalisiert, heißt *AKS_Manager*. Seine Verbindung zu *Baumaßnahme* und den anderen Bereichsmanagern finden sich im Diagramm der Abb. 18, die zu den von ihm verwalteten Objekten in Abb. 21

NeueBerechnung() erzeugt eine neue Kostenberechnung nach AKS aus der Datenbasis des Entwurfs- und des Grunderwerbsbereichs. (Ein praxisnäheres Modell würde den Entwurfsbereich in mehrere Bereiche zerlegen, parallel zu den Leistungsbereichen des Anwendungsfalldiagramms.) Die Operation wird mit einem *KBK_Prototyp*-Objekt versorgt, welches die zu verwendende Katalogstruktur repräsentiert.

FortBerechnung() erzeugt eine *fortgeschriebene* Berechnung. Die aktuelle Berechnung wird zu Vergleichszwecken gesichert, es wird eine neue Berechnung durchgeführt unter Zugrundelegung derselben Katalogstruktur wie bei der noch aktuellen.

GibBerechnung() teilt das aktuelle *AKS_Berechnung*-Objekt mit. Weitere polymorphe Varianten zu dieser Operation können vorgesehen werde, z.B. um eine frühere Berechnung abzurufen.

A.3.3.2 AKS_Berechnung und ihre Kollaboration

Laut AKS kann der dort angegebene Kostenberechnungskatalog konfiguriert werden, und zwar

- durch Einführung regionalisierter Varianten
- durch Verwendung der sog. Globalpositionen (ARS 13/90)
- durch sog. Freitexte

Typ KBK_Prototyp

Dieser Situation trägt das Modell wie folgt Rechnung: Der Typ ***KBK_Prototyp*** repräsentiert Objekte, die die Struktur des im Allgemeinen bei einer Baumaßnahme zu verwenden-

den KBK beschreiben. Dies deckt die regionalisierten KBK-Varianten und die Nutzung der Globalpositionen ab. Der *AKS_Manager* weiß darüber Bescheid, wo sich der für *NeueBerechnung* angeforderte Katalogprototyp befindet.

Typ KBK

Zu Beginn einer *neuen* Kostenberechnung wird aus dem *KBK_Prototyp*en ein maßnahmenspezifischer, als *lokal* bezeichneter Katalog vom Typ ***KBK*** durch Kopie erzeugt. Dies leistet die Operation ***KloneKBK()*** in *KBK_Prototyp*. Dieser kann durch Freitext weiter ergänzt werden:

FügeEin(kbkid, beschreibung, maß) erzeugt im KBK eine neue Leistungsposition *kbkid* der Beschreibung *beschreibung* mit der Maßeinheit *maß. kbkid* ist ein Wert zum Datentyp ***KBKid***. Die Werte dieses Datentyps gehorchen den Regeln für KBK-Positionsnummern, wie sie in der AKS festgelegt sind. *maß* ist Wert aus einer Aufzählung gültiger ***Maßeinheit***en, wie *Tonnen*, *Quadratmeter* oder *Stück*.

Anmerkung: Kompliziertere Ansätze für den Umgang mit Maßeinheiten sind denkbar, um Einheitenumrechnungen zu erleichtern, ob sich das aber wirklich lohnt, ist zweifelhaft. Leider kann man das Einheitenproblem nicht ganz außer acht lassen, weil Mengen der selben Art z.B. manchmal in m^3, manchmal in t benötigt werden.

Typ KBK_Position

Die zu verwendenden Preise werden in den lokalen *KBK* der neu zu erstellenden Berechnung eingefügt. Das bedeutet, dass ein Zugriff auf die einzelnen Positionen eines *KBK*-Objektes möglich sein muss. Die Positionen finden sich daher als Objekt des Typ ***KBK_Position*** wieder. (Man beachte, das dies für die Klasse *KBK_Prototyp* nicht erforderlich ist. Diese kann irgendwie strukturiert sein und muss nur auf Anforderung einen lokalen *KBK* erzeugen können.)

Die *KBK_Position*-en sind vom *KBK*-Objekt aus über die als überall sichtbar gekennzeichnete Assoziation *Positionen* erreichbar. Getreu den Regeln des Leitfadens existieren für diese Assoziation implizit Akzessor-Operationen.

KBK_Position muss in der Lage sein, entweder einen Pauschpreis oder mengenabhängige Einheitspreise aufzunehmen, wobei diese unmittelbar oder über eine Preisdokumentation eingebracht werden können. Daher werden drei entsprechende Operationen benötigt:

SetzePauschpreis(preis) setzt einen Pauschalpreis *preis* ein.

SetzeEinheitspreis(preis, menge, maß) setzt einen Einheitspreis *preis* pro Einheit *maß* an, der bis für Mengen <= *menge* gültig ist.

SetzePreisdoku(preisdoku) legt fest, dass der Preis für die Position aus einem Preisdokumentationsobjekt *preisdoku* vom Typ ***KBK_Preisdoku*** bezogen werden soll.

Zur Berechnung der Kosten zu einer Position muss zu gegebener Menge der sich daraus ergebende Preis erfragt werden. Dies leistet ***GibPreis(menge, maß).***

Typ KBK_Preisdoku

Eine Preisdokumentation muss mengenabhängige Preise ausweisen können. Dies leistet sie durch die Operation ***GibPreis(menge, maß)***. Sie ist offenbar polymorph verwandt mit der gleichnamigen Operation von *KBK_Position*. In der Tat wird letztere ihre Aufgabe an die Preisdokumentation *delegieren*, wenn die fragliche Position durch *SetzePreisdoku()* entsprechend konfiguriert wurde.

Gepflegt wird die Preisdokumentation durch ***SetzePreis(menge, maß, preis)***, mit offensichtlicher Semantik.

Typ AKS_Berechnung

Objekte dieses Typs repräsentieren vollständige AKS-Berechnungen. Jede *AKS_Berechnung* trägt einen Verweis auf alle Objekte der parallelen Geschäftsbereiche, die die Fachobjekte bündeln (sozusagen die lokalen Datenbanken der Fachobjekte der Bereiche). Für den Entwurfsbereich wurde der entsprechende Typ ***Variante*** genannt, für den ebenfalls oben eingezeichneten Grunderwerbsbereich heißt der Typ ***GE_Massnahme.*** Dieser Typ ist mit <<*okstra*>> gekennzeichnet, um anzuzeigen, dass eine Repräsentierung bereits im bestehenden OKSTRA® existiert.

Zur Konfiguration des *KBK* einer Berechnung bietet *AKS_Berechnung* die Operation ***GibKBK()*** an, die den Zugriff auf den Katalog und seine Positionen gestattet.

Die Operation ***ErzeugeTeil()*** dient zur Bildung eines neuen Teiles im Sinne der AKS, siehe nächster Abschnitt.

Berechne() ist eine Operation, die vom *AKS_Manager* angestoßen wird, um eine Kostenberechnung durchzuführen. Nach Durchführung enthält das *AKS_Berechnung*-Objekt die aktuellen Kosten.

Präsentiere() ist die Operation, die die Ergebnisse der Kostenberechnung verfügbar macht. Das Modell spezifiziert keine nähere Art der Präsentation. Es können unterschiedliche Formate unterstützt werden, z.B. Druckausgabe, Druckvorschau, KOSTRA®-kompatible Datei). Ist dies gewünscht, kann die Operation entsprechend oft polymorph implementiert werden.

Das UML-Klassendiagramm findet sich auf der folgenden Seite.

A.3.3.3 AKS_Teil

Die für die AKS zu definierenden Teile der Maßnahme werden durch Objekte des Basistyps ***AKS_Teil*** dargestellt, dessen Ableitungen gerade die Objekte der Hauptteile nach der AKS repräsentieren. Die Hauptteilstruktur ist für den eigentlichen Berechnungsprozess irrelevant, sie spielt nur für die Strukturierung des Ergebnisses durch *Präsentiere* eine Rolle.

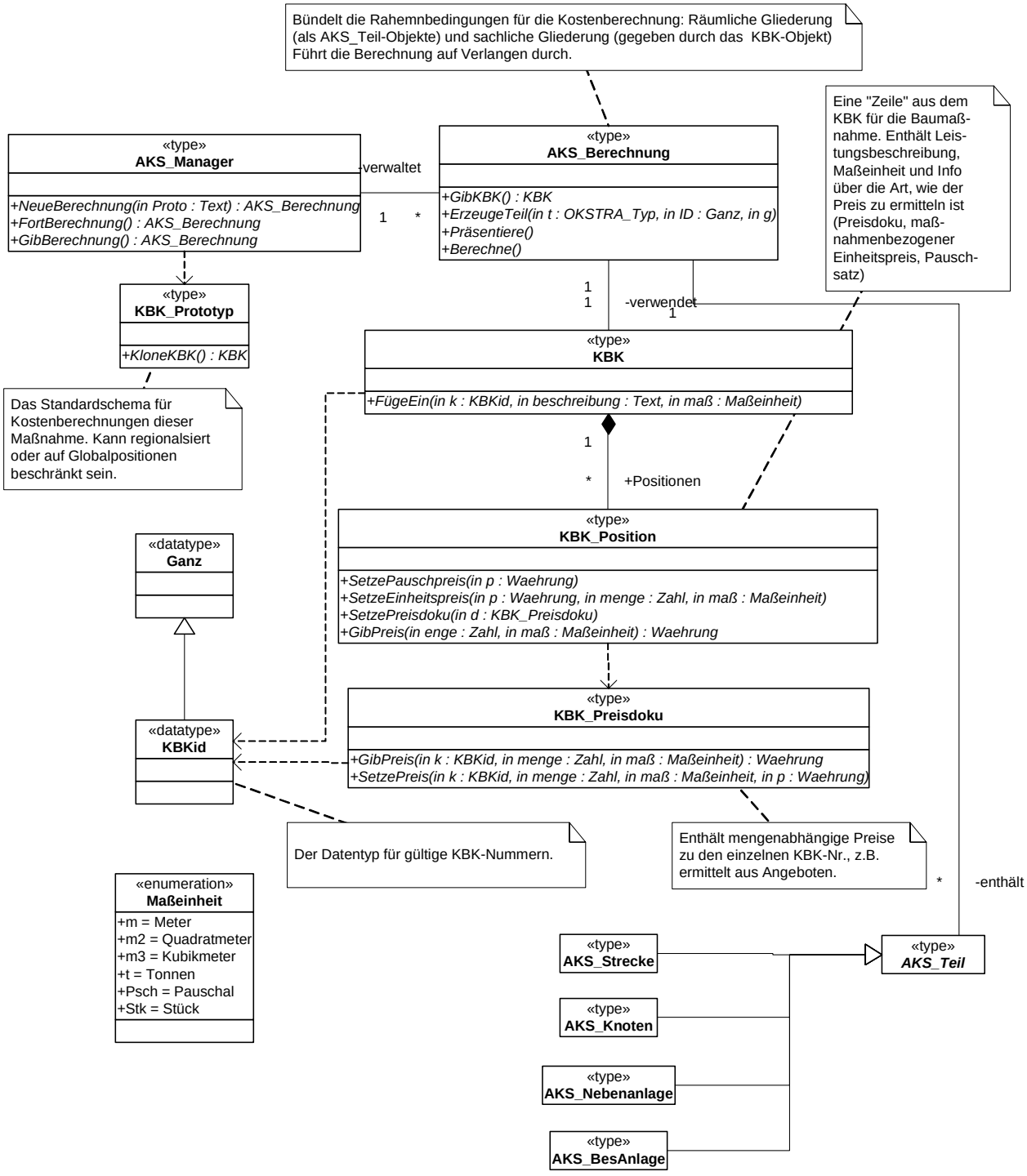


Abbildung 21. Geschäftsbereich AKS. Kostenberechnungskatalog.

Ein *AKS_Teil* wird durch die Operation ***ErzeugeTeil(typ, id, geo)*** hergestellt. Dabei ist *typ* der Typ des Teiles, z.B. ***AKS_Knoten***, *id* ein von außen vergebener Name und *geo* eine Geometrie, die die räumliche Ausdehnung des Teiles beschreibt.

AKS_Teil verfügt über eine Operation ***GibTeilgeometrie()***, die die bei Erzeugung des Teiles mitgegebene Geometrie nachweist.

Ein *AKS_Teil* muss die davon tangierten Fachobjekte nachweisen können, damit die Kostenberechnung nacheinander die Mengen ermitteln kann. Jedes Teil referiert daher die dazugehörenden Fachobjekte, die etwas zur Kostenberechnung beitragen. Solche Fachobjekte müssen über spezielle Operationen zur Unterstützung der Berechnung verfügen. Sie werden deshalb von einem Typ ***AKS_Fachobjekt*** abgeleitet.

Ein *AKS_Fachobjekt* muss zwei Dinge beherrschen: Es muss erstens angeben können, zu welchen Kostenberechnungskatalog-Positionen es einen Beitrag liefern kann. Dies können mehrere sein, z.B. eine Global-Position und eine Detail-Position. Die Entscheidung, welche davon tatsächlich genutzt wird, kann nicht dem Fachobjekt obliegen, sondern gehört in die Verantwortung des *AKS_Berechnung*-Objektes. Das dort verwendete *KBK*-Objekt muss so beschaffen sein, dass eine eindeutige Auswahl einer KBK-Position möglich ist. Die Operation ***FürKBKid()*** gibt eine Ansammlung von KBK-Nummern zurück, die die KBK-Positionen identifizieren, zu denen das *AKS_Fachobjekt* etwas aussagen kann.

Die zweite Aufgabe des *AKS_Fachobjektes* ist, die für die Kostenberechnung benötigte Mengenberechnung durchzuführen. Dies leistet die Operation ***GibMenge(KBKid, maß)***. Die *KBKid* ist die Nummer der KBK-Position, für die die Mengenberechnung durchzuführen ist, *maß* ist wie vorher die Bezeichnung der gewünschten Maßeinheit. Die gewählte Parametrisierung erlaubt bei Notwendigkeit unterschiedliche Mengenaussagen je nach verlangter KBK-Position.

A.3.3.4 Typen für die Integration der Fachobjekte

Die Verantwortlichkeit, die benötigten Mengenangaben zu machen, obliegt offensichtlich den Objekten, die die zu bauenden Gegenstände repräsentieren, also den *Fachobjekten* des OKSTRA®.

Exemplarisch ist die Integration dieser Objekte in die Kostenberechnung für ein hypothetisches ***Tragschicht***-Objekt in Abb. 22 dargestellt. Die angegebene Attributierung der Klasse durch *Dicke*, *Material* und *Bindemittel* ist wiederum exemplarisch und für die Praxis sicherlich zu überarbeiten.

Viele Fachobjekte haben einen Geometriebezug. So wird die *Tragschicht* durch eine Flächengeometrie beschrieben. Alle derartigen geometriebehafteten Fachobjekte werden aus einem Typ ***FachobjektMitGeometrie*** abgeleitet. Damit man daraus auch wirklich Mengen berechnen kann, verfügen alle solchen Objekte über eine Assoziation zu einem Geometrieobjekt, das sich zur Mengenberechnung eignen muss. Derartige Geometrieobjekte werden zu einem Typ ***MengenGeometrie*** abstrahiert.

Der Typ *MengenGeometrie* verlangt eine Operation ***BerechneMenge(maß)***, die bei Ausführung die Mengenberechnung durchführt. Als Beispiele für instanzierbare Ableitungen von *MengenGeometrie* bieten sich die aus dem OKSTRA® bekannten *Volumen*- und *Oberfläche-O*bjektklassen an. Weitere Klassen lassen sich aus den REB bzw. der GAEB-VB 23.004 Allgemeine Mengenberechnung gewinnen, z.B. Prismen, Rampen usw.

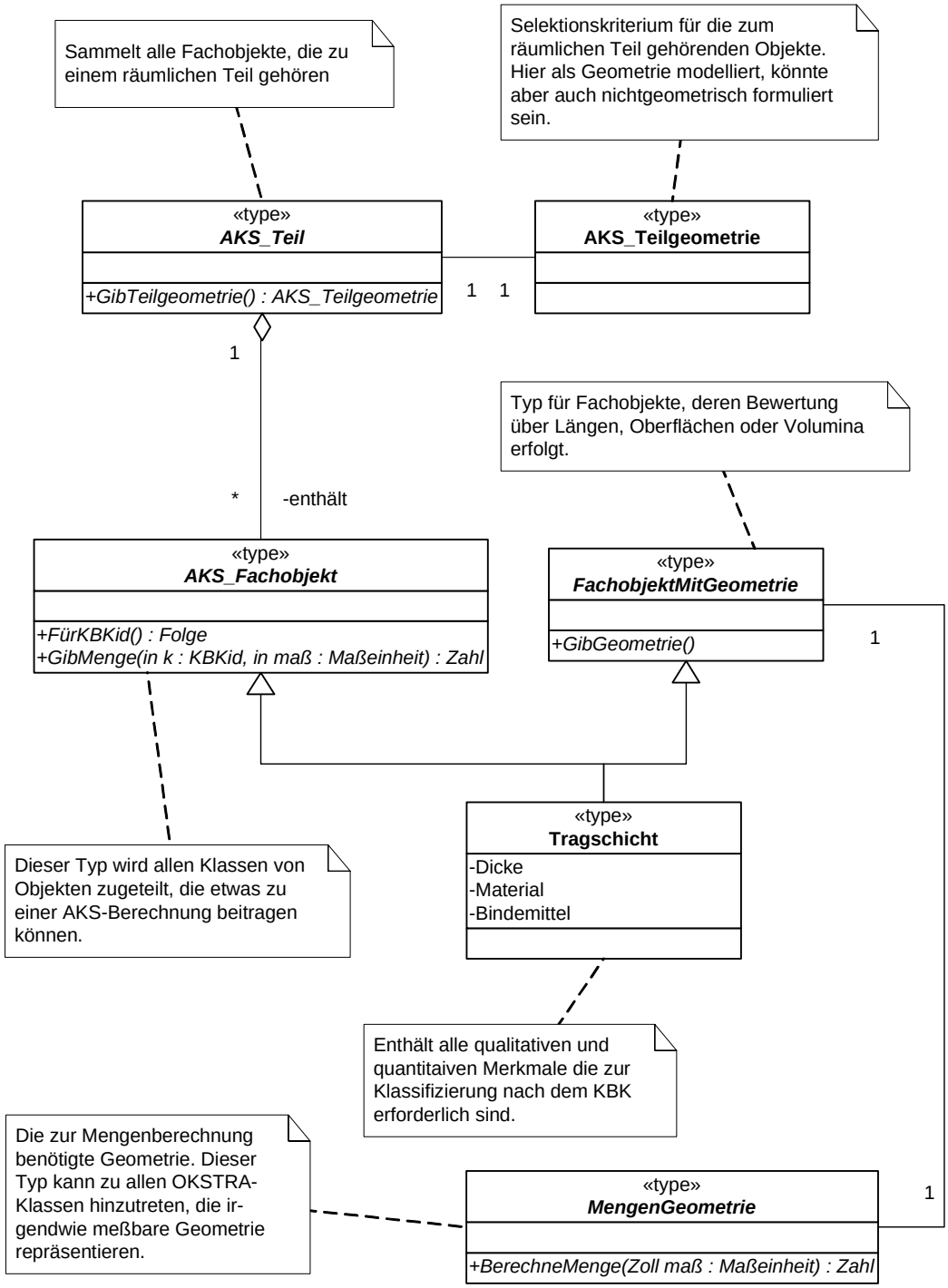


Abbildung 22. AKS- und Mengen-Interface für die Fachobjekte.

A.3.3.5 Ablauf der AKS-Berechnung

Die bisher gezeigten Klassendiagramme geben den Teil des Objektmodells wieder, der die strukturellen Beziehungen der Objekte umfasst:

- Die Klassensymbole zeigen, welche Operationen für die Objekte des Typen zur Verfügung stehen.

- Eine Assoziationslinie zeigt, dass sich Objekte der verbundenen Typen kennen: das eine kann Nachrichten an das andere senden (d.h. Operationen ausführen, die das andere Objekt betreffen).
- Generalisierungslinien zeigen an, wie sich Typen aus anderen Typen zusammensetzen. Jeder Typ repräsentiert mit seinen Operationen die Mittel zur Bewältigung einer bestimmte Aufgabe innerhalb des Gesamtprozesses.

Wie die Objekte zur Lösung der Gesamtaufgabe miteinander „reden" müssen, d.h. in welcher Reihenfolge wer welche Operationen ausführt, geht aus den bisherigen Diagrammen und Erläuterungen noch nicht hervor. Es ist Thema der folgenden Ausführungen. Die grafischen Darstellungen dafür sind die UML-Sequenzdiagramme der Abb. 23 und 24.

A.3.3.6 Konfiguration und Anstoß der Berechnung

Der Ablauf beginnt mit der Suche der *Baumaßnahme,* für die die Berechnung durchzuführen ist, im *Baumaßnahmenverzeichnis* durch *SucheMaßnahme()*. Die gefundene Maßnahme wird als nächstes nach dem Bereichsmanager für die AKS, d.h. dem zuständigen *AKS_Manager*-Objekt befragt – *GibManager()*.

Dieses Manager-Objekt wird als nächstes durch *NeueBerechnung()* damit beauftragt, eine neue Berechnung anzulegen.

Dazu ist zuerst ein initial leeres Objekt vom Typ *AKS_Berechnung* zu erzeugen, das später die Berechnungsdaten aufnehmen wird. Dies ist eine innere Funktionalität des *AKS_Managers* und wird daher im Sequenzdiagramm nicht namentlich durch eine Operation dargestellt, sondern durch eine <<*create*>> Nachricht angezeigt. Die Struktur der Berechnung wird durch eine Kopie des Kostenberechnungs-Kataloges vorgegeben. Die *KloneKBK()*-Operation leistet dies. Sie wird als Folge der <<create>>-Nachricht ausgelöst, die die neue Berechnung mit geklonten *KBK* zu verknüpfen.

Nicht gezeigt ist, woher der Erzeugungsmechanismus weiß, welches das zum Klonen des KBK anzusprechende *KBK_Prototyp*-Objekt ist. Das hat folgenden Grund: Bei der Implementation der Typen (z.B. in Form von Klassen einer objektorientierten Programmiersprache) ist auf den vom Implementationssystem vorgesehenen Konstruktionsprozess für Objekte einzugehen. Dabei kann es sich um polymorphe Konstruktoren, Factory-Objekte oder eine Kombination aus solchen Verfahren und zusätzlichen Operationen zum Initialisieren der Objekte handeln. Da das OKSTRA®-Modell unabhängig von Implementationsdetails bleiben muss, kann es hierzu keine andere Aussage machen als die im Sequenzdiagramm dargestellte: Erzeuge (wie auch immer) ein *AKS_Berechnung*-Objekt und teile diesem (wie auch immer) das zu verwendende KBK-Schema mit. Es handelt sich dabei sozusagen um eine interne Unterhaltung der an der AKS-Kollaboration beteiligten Objekte.

Anmerkung: Eine *Kollaboration* ist eine Gruppe von Typen, die eng miteinander zusammenarbeiten, so eng, dass man sie zweckmäßig als Einheit betrachten und auch implementieren sollte. Eine präzisere Definition ist vermutlich nicht möglich. Ein gutes Objektmodell zeichnet sich dadurch aus, dass die Typen sich in Gruppen (auf logischer Ebene *Pakete*, auf Implementationsebene *Komponenten*) zusammenfassen lassen, die mit einander so sparsam wie möglich verflochten sind.

Im Sequenzdiagramm drückt sich die bewusste Vernachlässigung solcher interner Mechanismen so aus, dass die meisten Operationsanforderungen von der Applikationsseite, also von Objekt ganz links ausgehen. In ihrer Gesamtheit bilden sie die Schnittstelle, die ein als Komponente realisierter Dienst (z.B. ein Web Service) offerieren müsste.

Andere Operationen, wie z.B. *KloneKBK(),* sind eingetragen, um anzuzeigen, dass hier wiederum eine Grenze zu einer anderen Komponente überschritten wird: die zur KBK-Schema-Verwaltung, die die Pflege des standardmäßig zu verwendenden KBK in Form des *KBK_Prototyp*-Objektes zur Aufgabe hat.

Schließlich sind Operationen wie *Berechne()* eingetragen. Diese gehören zwar zum Inneleben der AKS-Kollaboration, sie erfordern jedoch eine weitergehende Analyse – siehe folgender Unterabschnitt.

Nach diesem Exkurs nun die weitere Ablaufsbeschreibung.

Die durch *NeueBerechnung()* zur Verfügung gestellte *AKS_Berechnung* muss nun konfiguriert werden. Dies besteht zunächst in der Erstellung der *AKS_Teil*e durch *Erzeuge-Teil()*. Ein AKS-Teil wird primär durch den Raumbezug (Parameter *g,* siehe Klassendiagramm Abb. 22) gegeben. Zusätzlich ist der Typ des Teils (z.B. durchgehende Strecke) und die Identifikation (Teilnummer) anzugeben. Es ist nur eine solche Nachricht eingetragen, i.d.R. werden aber mehrere Teile zu konfigurieren sein, dann sind so viele Nachrichten abzusetzen, wie es Teile geben soll.

Außerdem muss eventuell der an der Berechnung hängende *KBK* konfiguriert werden, d.h. es müssen Freitext-Positionen ergänzt werden, und der *KBK* muss mit Preis-Informationen gefüttert werden. Aus Platzgründen ist hier nur die Ergänzung mit Freitext-Positionen dargestellt – *FügeEin()*.

Das folgende UML-Sequenzdiagramm zeigt den Anstoß für eine Erstberechnung nach AKS sowie für eine Fortschreibung.

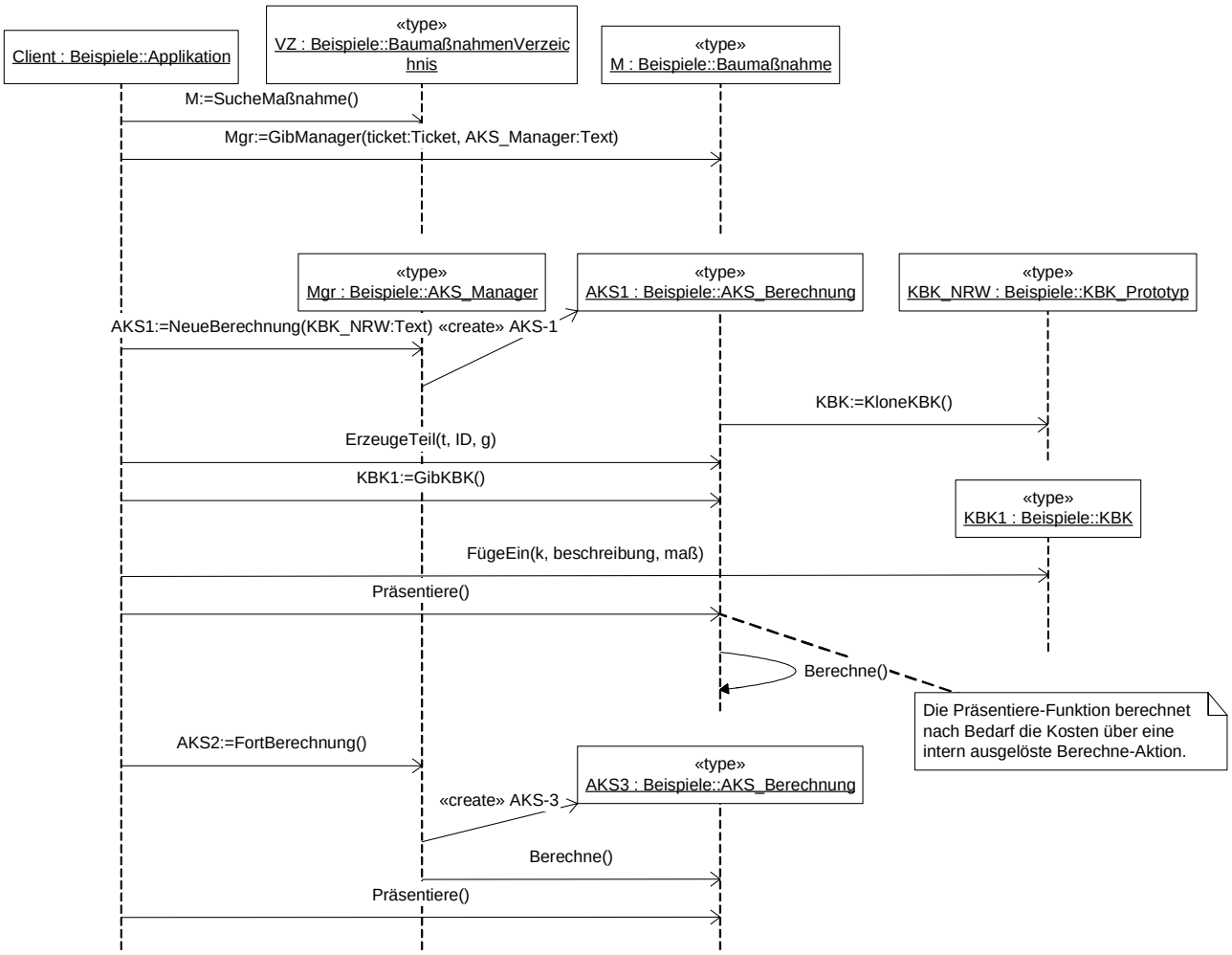


Abbildung 23. Sequenzdiagramm AKS-Berechnung.

Die eigentliche Durchführung der Berechnung kann erst dann vorgenommen werden, wenn die AKS_Berechnung vollständig konfiguriert ist. Da der Umfang der Konfiguration nicht bekannt ist, muss die Berechnung spätestens dann erfolgen, wenn ihre Ergebnisse durch *Präsentiere()* von außen abgerufen werden. Dieser Fall ist hier eingezeichnet: *Präsentiere()* löst intern eine *Berechne()*-Operation aus.

Die Fortschreibung einer Berechnung läuft wie folgt: Die dafür zuständige Operation *Fort-Berechnung* erzeugt zunächst ein neues *AKS_Berechnung*-Objekt. Allerdings wird kein leeres Objekt dieses Typs erzeugt, sondern die Konfiguration wird von der letzten Berechnung übernommen, damit die alte und die fortgeschriebene Berechnung vergleichbar bleiben:

- Der *KBK* des noch aktuellen Objektes wird durch Kopie übernommen.
- Die Ansammlung der *AKS_Teil*e wird übernommen.
- Die Verknüpfung zu *Variante* wird übernommen.

Nach Erzeugung des Objektes wird die Neuberechnung durch die Operation *Berechnung()* gestartet, um eine aktualisierte Kostenberechnung zu erstellen. *Präsentiere()* gibt dann später das neue Berechnungsergebnis.

A.3.3.7 Durchführung der Mengen- und Preisberechnung

Die Logik der Mangen- und Preisberechnung ist in einem weiteren Sequenzdiagramm (Abb. 23) dargestellt.

Die Kostenberechnung für ein AKS-Teil einer Neubaumaßnahme geht nun so vor sich, dass alle Fachobjekte, die zum Teil gehören, nach ihren KBK-Positionen befragt werden müssen.

Berechne() befragt also *AKS_Teil-* für *AKS_Teil*-Objekt über *GibTeilgeometrie()* nach der räumlichen Ausdehnung des Teiles. Über die *SucheFachobjekte()*-Operation des mit der *AKS_Berechnung* verbundenen *Variante*-Objektes erhält man nun alle Fachobjekte aus dem Entwurfsbereich.

Diese werden nacheinander aufgezählt und per *FürKBKid()* einzeln nach den KBK-Nummern befragt, zu denen sie Kosten beitragen.

Für jede dieser KBK-Nummern wird nun die entsprechende Menge berechnet: *GibMenge().* Die Logik für diese Mengenberechnung lässt sich nun für geometrisch beschreibbare Fachobjekte in zwei Schritte zerlegen:

- *GibMenge* für das *AKS_Fachobjekt* führt die Operation *GibGeometrie* aus. Das geht, denn die Implementation von *GibMenge* für die konkrete Fachobjektklasse, z.B. die *Tragschicht*, weiß ja, das es sich um ein *FachobjektMitGeometrie* handelt.
- Für die abgelieferte Geometrie wird *BerechneMenge* ausgeführt.

Ist die Mengenberechnung für alle Objekte abgeschlossen, sind als nächstes die Preise für die KBK-Positionen zu ermitteln. Dies ist im zweiten Teil des Sequenz-Diagramms dargestellt. Für jede KBK-Nummer, zu der es eine Menge gibt, passiert folgendes:

- Zuerst wird das *KBK_Position*-Objekt zur *KBKid* geholt. Dies ist eine Operation auf dem Aggregat, das hinter der im Klassendiagramm vermerkten Komposition zwischen *KBK* und *KBK-Position* steckt. Technische Details sind hier der Verständlichkeit des Konzeptes halber nicht dargestellt – sie würden die Verwendung der *FürAlle*-Operation des Aggregates samt Anstoß einer Callback-Funktion zeigen.
- Der mengenabhängige Einheitspreis wird der *KBK_Positi*on über *GibPreis()* entnommen. Falls dort eine Preisdokumentation vom Typ *KBK_Preisdoku* als Quelle für die Preise angegeben ist, wird die Preisermittlung dorthin weiterdelegiert – es wird *GibPreis()* in der Preisdoku angerufen. Andernfalls enthält die *KBK_Position* den Preis selbst.

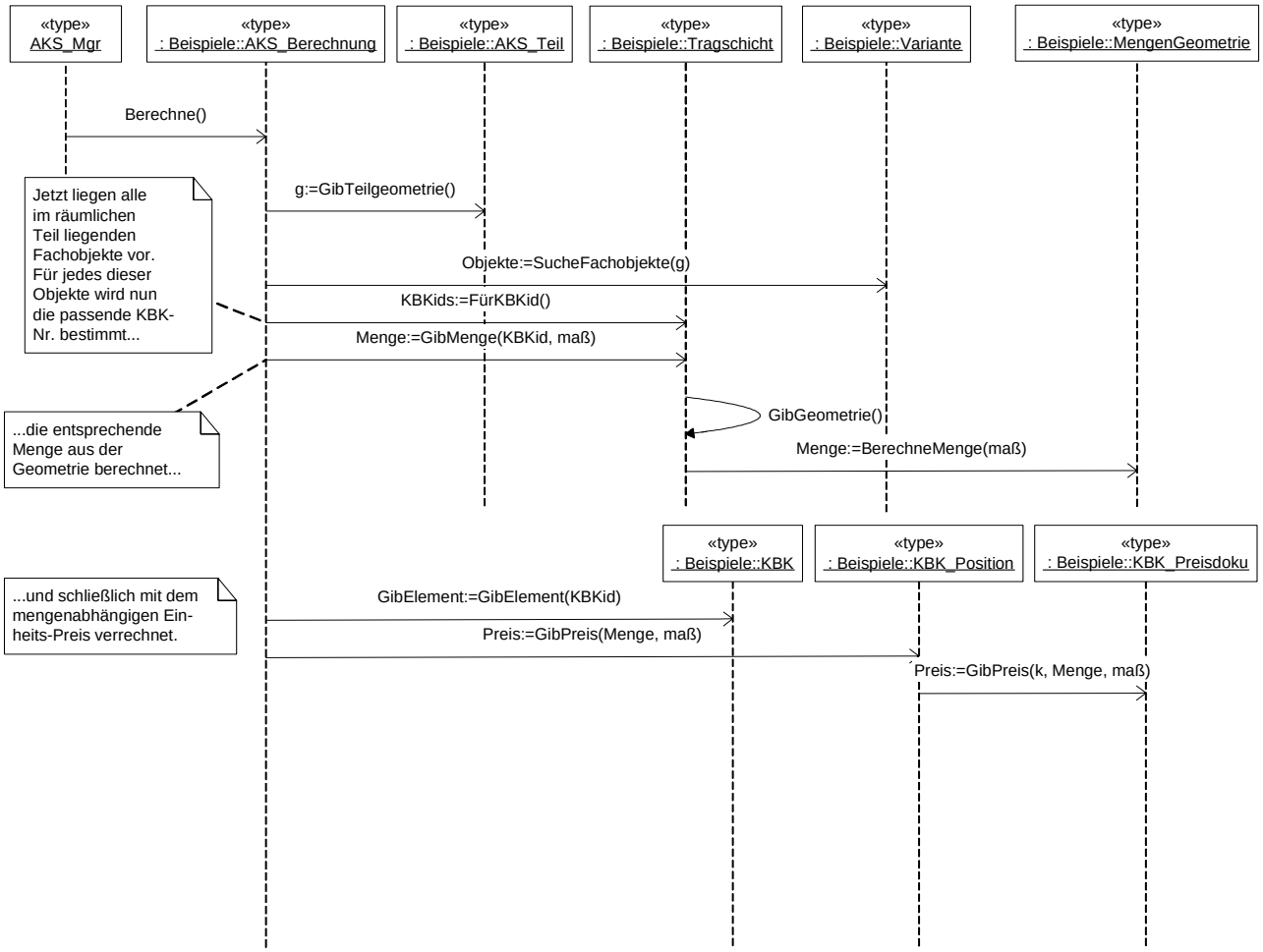


Abbildung 24. Sequenzdiagramm Mengen und Preise.

A.3.3.8 Grunderwerbskosten

Zur Kostenberechnung gehört noch die Ermittlung der Grunderwerbskosten. Das zugehörige Klassendiagramm findet sich in Abb. 24. Hierzu sind folgende Bemerkungen zu machen:

1) Die Kostenberechnung für den Grunderwerb wird innerhalb des Grunderwerbs-Bereichs selbst vorgenommen. Die Modellierung für die Preisfestlegung ist im GE-Bereich des OKSTRA® nachzutragen. Als Beispiel: Jede einzelne Erwerbsfläche muss ihren eigenen Preis haben können. Dieser wird entweder einer Bodenrichtwertkarte entnommen oder es ist der vereinbarte Kaufpreis (bzw. bei temporärer Nutzung die vereinbarte Miete o. dgl.)

2) Die AKS sieht eine Gliederung der GE-Kosten vor in

   - Erwerb von Grundstücken
   - Erwerb von Gebäuden und Anlagen

- Sonstige Entschädigungen
- Vermessung und Vermarkung

mit weiterer Feingliederung. Es sind für die einzelnen Positionen Gesamtpreise erforderlich, es erfolgt keine Aufteilung nach einzelnen Erwerbsobjekten. Daher kann die Kostenanfrage unmittelbar vom *GE_Maßnahme*-Objekt beantwortet werden. Gemäß der inkrementellen Erweiterungs-Strategie für den OKSTRA® wird dazu ein Typ ***AKS_GE*** geschaffen, von dem *GE_Maßnahme* zur Unterstützung des AKS-Prozesses zusätzlich abgeleitet wird. Dieser Typ verlangt die Operation ***GibGEKosten(kbkid)***, die die Kosten für die durch *kbkid* gegebene Position der Hauptgruppe 1 – GE-Kosten – der AKS liefert.

3) *Berechne()* wendet sich zur Kostenberechnung für den Grunderwerb an *GibGE-Kosten()* in den *GE_Maßnahme*-Objekten, welche über *Variante* von *AKS_Berechnung* erreichbar sind.

Das folgende UML-Klassendiagramm zeigt die benötigten Typen:

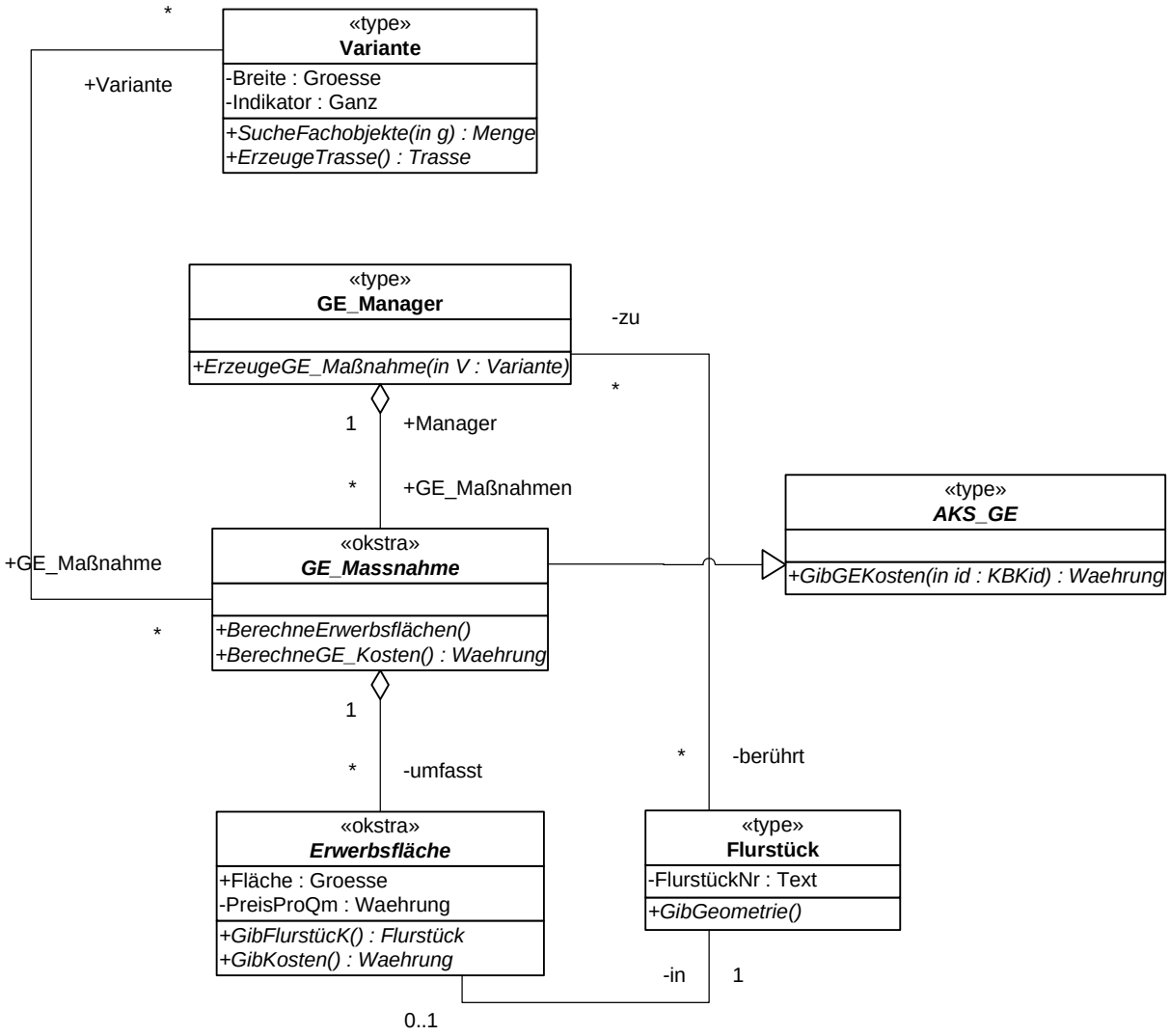


Abbildung 25. Grunderwerbskostenberechnung.

A.3.3.9 Anmerkungen zur Mengenberechnung

Das Modell berücksichtigt exemplarisch den praktisch wichtigen Fall, wo Mengen aus geometrischen Daten berechnet werden. Daneben sind jedoch auch andere Fälle zu bedenken:

- die automatische Berechnung von Stückzahlen
- bereits in den Objekten vorhandene Mengenangaben bzw. Attribute zur Berechnung

Der erste Fall kann analog zu *FachobjektMitGeometrie* durch einen entsprechenden Type ***FachobjektMitZählmenge*** in das Modell eingeführt werden. Dieser würde dem Fachobjekt einen Bezug zu einer *Ansammlung* von zu zählenden Objekten abverlangen, analog zur Assoziation *FachobjektMitGeometrie* - *MengenGeometrie*. Die Implementation von *GibMenge()* in *AKS_Fachobjekt* würde einfach die *Anzahl*-Operation von *Ansammlung* bemühen.

Für den zweiten Fall muss gar nichts weiter modelliert werden. *GibMenge()* rechnet die Menge einfach aus dem internen Objektzustand aus.

Eine zweite Bemerkung betrifft die Konfiguration von *Mengenberechnungsverfahren.*

Für ein und dieselbe Geometrieklasse müssen von Fall zu Fall verschiedene Mengenberechnungsverfahren (MBV) eingesetzt werden. Notwendig ist das u.A., weil Mengenberechnungsverfahren bei der Vergabe vereinbart werden können.

Es sei voraus gesetzt, dass alle Geometrieobjekte derselben Klasse mit demselben Mengenberechnungsverfahren arbeiten sollen, dieses aber gegebenenfalls ausgetauscht werden soll. Die ist ein typischer Fall für den Einsatz von Strategie-Objekten.

Das Modell löst dies dadurch, dass über einen Typ ***MBV_Konfiguration*** ein *Attribut* ***MBV*** *mit klassenweiter Sichtbarkeit* in das Fachobjekt (im folgenden Beispiel: ***Bodenmasse***) eingeführt wird. Die Akzessor-Operationen eines solchen Attributes sind, wie zu vermuten, Klassenoperationen. Das Attribut kann als Werte Mengenberechnungsverfahren annehmen, die als abstrakter Datentyp ***MBV*** modelliert sind. Die Ableitungen daraus sind die konkreten Mengenberechnungsverfahren, z.B. nach REB 21.033: ***MBV_REB_21_033***.

Die Mengenberechnungsverfahren können als Datentyp modelliert werden, weil sie die Bedingungen dafür erfüllen: unendliche Lebensdauer, keine inneren Zustände. Sie gestatten eine Operation ***BerechneMenge(g, maß)***.

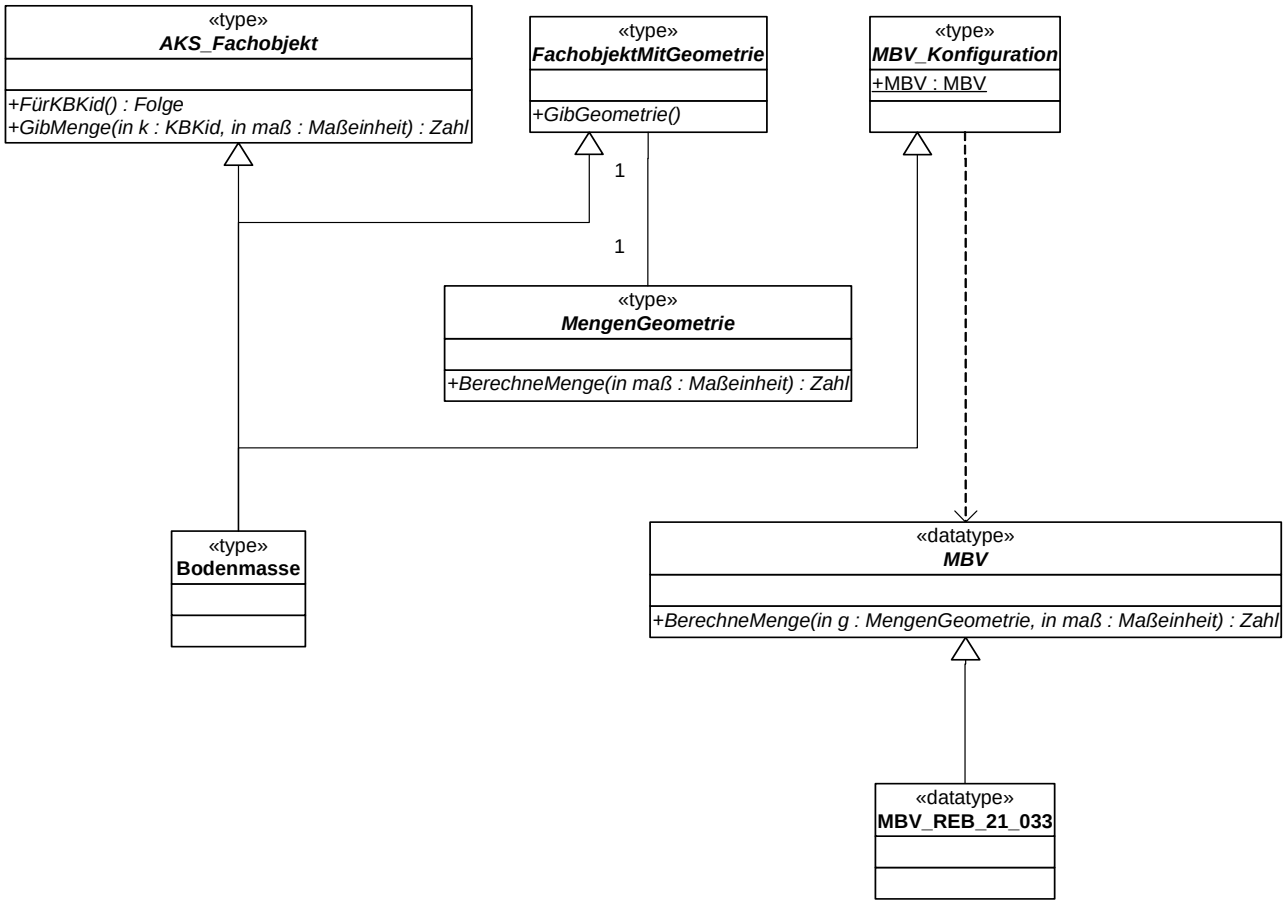


Abbildung 26. Konfigurierbare Mengenberechnung.

MBV_Konfiguration stattet im Beispiel *Bodenmasse* mit der Möglichkeit aus, das Mengenberechnungsverfahren zu ändern. Dazu muss ja nur der Wert des Attributes *MBV* umgesetzt werden. Die Mengenberechnung selbst läuft nun so ab, dass *GibMenge()*, wie gehabt, *GibGeometrie()* benutzt, um die Mengengeometrie zu holen. Diese wird aber jetzt an die Operation *BerechneMenge()* des Mengenberechnungsverfahrens aus dem Attribut *MBV* übergeben.

A.3.3.10 Umsetzung STLK-KBK

Solange AKS und STLK noch nebeneinander existieren, muss man sich auch über die Verbindung der zwei Leistungskataloge Gedanken machen. Angenommen, es gibt ein STLK-Preisdokumentationsobjekt eines Typs *STLK_Preisdoku*. Man kann diese für die *KBK_Preisdoku* nutzbar machen.

Betroffen ist davon nur die *GibPreis()*-Operation von *KBK_Preisdoku*. Falls diese keinen eigenen Preis in einer Position vorfindet, würde sie über eine Tabelle die zur betreffenden KBK-Nummer passende STLK-Nummer suchen (nach den vorliegenden Vorschlägen zur Abbildung der beiden Kataloge aufeinander) und dann die Anfrage an ein *STLK_Preisdoku*-Objekt delegieren. *KBK_Preisdoku* muss dazu nur um eine Operation erweitert werden, die es gestattet, eine Verknüpfung mit einer *STLK_Preisdoku* herzustellen.

A.3.4 Zur Problematik der existierenden Fachobjekte

Um Mengen berechnen zu können, so wie sie von den Geschäftsprozessen innerhalb der Baumaßnahme gebraucht werden, reicht die vorhandene fachliche Modellierung aus dem Entwurfsbereich des OKSTRA® nicht aus. Die dortige Spezifikation beschreibt nämlich nur den geometrischen Aspekt. Die Schicht der geometrisch orientierten OKSTRA®-Entwurfsobjekte ist vielmehr zu ergänzen um eine darüber liegende Schicht, die die beim Bau tatsächlich erzeugten Bestandteile des Werkes Straße beschreibt.

Zur Illustration diene die Situation bei Lärmschutzbauwerken. Es gibt hierfür eine ausgezeichnete Modellierung im Bestandsbereich des OKSTRA®. Diese ist jedoch auf das Netz bezogen und kann daher für den Entwurfsbereich (und damit für die Kostenberechnung) nicht genutzt werden, denn für geplante Straßen existieren keine Netzobjekte. Die Modellierung der Ingenieurbauwerke im Entwurfsbereich ist andererseits wegen der fehlenden Geometriebezüge und Detaillierung auch nicht geeignet für die Kostenberechnung.

Letztlich fehlt eine einheitliche Modellierung der Fachobjekte, die für alle Geschäftsprozesse nutzbar ist, u.U. vermittelt durch prozessbezogene, einschränkende Sichten

Ein möglicher Anhaltspunkt für diese Modellierung ergibt sich z.B. aus einer systematischen Aufbereitung der Standardleistungskataloge (STLK); schließlich sind es die dort verzeichneten Objekte, die ausgeschrieben, hergestellt und bezahlt werden. Aus den Grund- und Folgetexten des STLK können die benötigten Typen und ihre Attributierung abgeleitet werden. Dies wird aber vollständig nur die Sicht der Bereiche AKS und AVA abdecken, für andere Geschäftsprozesse sind andere Quellen hinzuzuziehen und miteinander zu harmonisieren.

A.3.5 Ereignisse

Von großem praktischem Nutzen ist die Möglichkeit, die Kostenberechnung darüber zu benachrichtigen, dass sich im selben oder in anderen Geschäftsbereichen Änderungen ergeben haben, die eine Neuberechnung der Kosten erforderlich machen. Die Möglichkeit solcher Benachrichtigungssysteme erschließt den Nutzen verteilter Dienste in Form von besserer Verfügbarkeit aktueller und zuverlässiger Information: Änderungen in dezentralen Datenbeständen können zeitnah zu ihrer Ursache und vollständig vorgenommen werden, so dass Anfragen und Auswertungen aktuelle Zustände der Realität berücksichtigen.

Typische Fälle sind:

- In einem Preisdokumentation-Objekt wurden Preise geändert.
- Die Grunderwerbskosten haben sich geändert.
- Der Entwurf wurde geändert (wobei bei weiterer Aufgliederung der Geschäftsbereiche entsprechende Spezialisierungen notwendig sind, z.B. für Straßenentwurf, Bauwerksentwurf, Landschaftsplanung, usw.)

Die Benachrichtigung selbst lässt sich über *das im Leitfaden zur Modellierung des objektorientierten OKSTRA* angegebene Ereignismodell abwickeln: Der *AKS_Manager* ist als *Bereichsmanager* ein *Ereignisverarbeiter*. Er würde bei seiner Instanzierung die Ereignisse, über die er informiert werden möchte – z.B. *Straßenentwurf geändert* – bei seiner *Baumaßnahme* registrieren. Wird nun der Entwurf geändert, sendet die geänderte Variante ein Ereignis an die *Baumaßnahme*, die es u.a. an den AKS_Manager weiterdelegiert. Das wird im folgenden Sequenzdiagramm (Abb. 27) gezeigt.

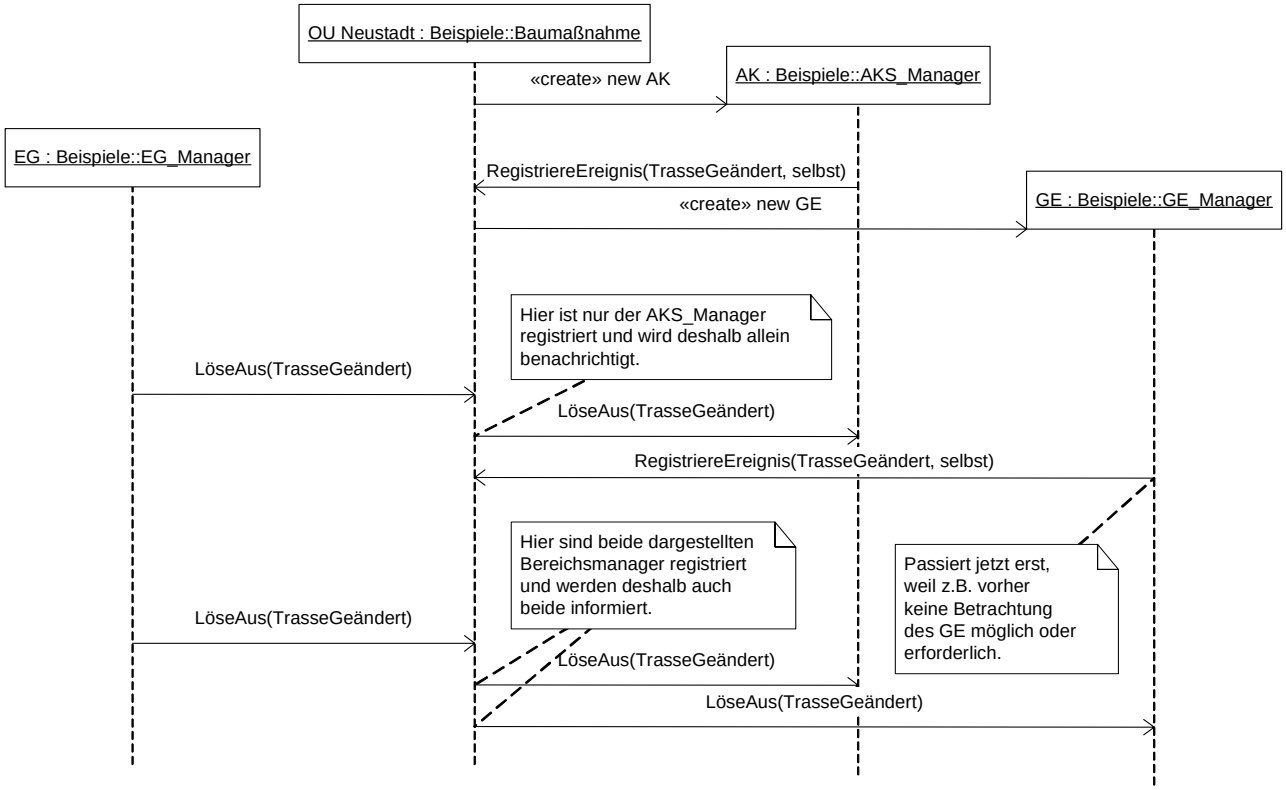


Abbildung 27. Ereignisübermittlung.

Dieser würde sich zunächst nur merken, dass die letzte Kostenberechnung nicht mehr aktuell ist. Bei der nächsten *Präsentiere()*-Aufforderung für das aktuelle *AKS_Berechnung*-Objekte würde die nutzende Anwendung darüber informiert, dass die Berechnungsgrundlagen sich geändert haben und daher die Berechnung zu aktualisieren ist.

A.4 Die prototypische Realisierung als verteiltes System

Um die Brauchbarkeit des Modells für die praktische Unterstützung von Geschäftsprozessen zu demonstrieren, wurde ein Prototyp entworfen und implementiert.

A.4.1 Implementationstechnik

Der Systementwurf dafür sah ein auf mehrere Server verteiltes System, bestehend aus Web-Applikationen und Web-Diensten - *Web Services* - vor. Die Realisierung erfolgte größtenteils unter Verwendung der Microsoft .NET Technologie in der neuen Programmiersprache C# . Die verwendete Version der Programmentwicklungsumgebung und .NET Bibliotheken war Beta 2.

Web Services versprechen u.A. die leichte Überbrückung von ansonsten gravierenden Plattformunterschieden. Die Technologie scheint daher für viele verteilte Anwendungen die ansonsten fällige Wahl zwischen einer COM-Lösung aus der Microsoft-Welt und einer eher UNIX-zentrierten CORBA-Lösung überflüssig zu machen und stattdessen den Aufbau von Architekturen auf herstellerunabhängigen und durchgängig unterstützten Standards zu gestatten.

Um die versprochene Interoperabilität zu zeigen, wurde deshalb ein Web Service in Java auf einem Linux-System implementiert und in den Prototypen integriert.

Auch die Einbindung von bestehenden Software-Komponenten konnte am Beispiel einer Integration einer COM-Komponente gezeigt werden.

Anmerkung: Eine analoge Demonstration für CORBA wurde nicht durchgeführt, da keine CORBA-Implementation zur Verfügung stand. Wegen der bekannt guten Anbindung von CORBA an Java wird darin aber kein Nachteil gesehen.

A.4.1.1 Web-Applikationen und Web Services

Eine Web-Applikation ist ein Dienst, der einem Nutzer über das World Wide Web im Internet zur Verfügung steht. Der Anwender braucht zur Nutzung der Applikation nur einen *Internet-Browser*, z.B. Internet Explorer, Netscape, Mozilla oder Opera. Die Applikation besteht typischerweise aus statischen HTML- und Grafikdateien sowie aus Programmen, die daraus, aus Datenbankinhalt und Berechnungen *dynamisch* HTML-Seiten erzeugen.

Anfragen von einem Browser landen zunächst bei einem *Webserver*. Er wird durch die bekannten URLs, die Adressen im World Wide Web, identifiziert. Bekannte Webserver Software-Pakete sind Apache und Internet Information Server. Diese Software startet dann die angeforderte Applikation. Das Ergebnis der Anfrage wird vom Webserver als Antwort an den Browser zur Präsentation in Bild und Ton übertragen.

Zur Übertragung der Anfrage und des Ergebnisses zwischen Browser und Server dient das Internet-Protocol HTTP (Hypertext Transfer Protocol). Die Anfrage wird als HTTP-Request, die Antwort als HTTP-Response bezeichnet.

Die gleiche Maschinerie kann aber auch zwischen *zwei Servern* eingesetzt werden. Eine Applikation auf dem einen Server kann einen Teil der Aufgabe an einen anderen delegieren. Hier ist es offenbar nicht mehr nötig, dass die Daten in Formaten übertragen werden, die ein Browser interpretieren kann, das ist meist sogar gar nicht erwünscht. Ein Pro-

gramm, dass auf diese Art Dienste über Web-Technologie für andere Applikationen bereitstellt, heißt *Web Service*. Web Services können offenbar nicht nur von Web-Applikationen, sondern auch von anderen Web Services aus genutzt werden.

Die Technologie bietet sich bei geeigneter Auslegung offenbar dazu an, verteilte Systeme zu konstruieren, deren Teile auf weit auseinander liegenden, vielleicht nur über das Internet verbundenen Systemen installiert sind. Insbesondere können Web Services von den Stellen eingerichtet werden, die über die für die Realisierung eines Dienstes nötige fachliche Kompetenz und entsprechende Datenbestände verfügen. Sie müssen dabei nicht auf den Rest der Welt Rücksicht nehmen, sondern ihre Dienste nur gemäß gewisser offen standardisierter Protokolle anbieten.

A.4.1.2 XML und SOAP

Web-Applikationen und Web-Dienste kommunizieren miteinander, in dem sie Dokumente austauschen, die in der ***Extensible Markup Language XML*** verfasst sind. Diese Dokumente enthalten den Namen der auszuführenden Operation sowie die Objekte, die als Parameter und Ergebnisse zu übergeben sind. XML strukturiert Dokumente hierarchisch in Blöcke, die durch sogenannte ***Tags*** geklammert werden, das sind Marken, die die Nutzinformation am Anfang und am Ende begrenzen und die durch ihren Namen die Bedeutung (den Typ) der Nutzinformation bezeichnen.

Die hierarchische Organisation gestattet es, die Nutzinformation eines bestimmten Typs weiter in kleinere Blöcke zu zerlegen. Offensichtlich ist diese Strukturierung dafür geeignet, ineinander geschachtelte Objekte zu kodieren, in dem man die Tags so gestaltet, dass sie den Objekttypen entsprechen. Referenzen zwischen Objekten können auch dargestellt werden, und zwar durch *Universal Resource Identifiers (URIs).* URIs sind eine Verallgemeinerung der bekannten, als URL bezeichneten WorldWideWeb-Internet-Adressen.

Die Nutzung von XML zur Kodierung von Objekten und Referenzen ist in den folgenden Standards geregelt:

- XML Linking Language (XLink) Version 1.0
  27 June 2001, Steve DeRose, Eve Maler, David Orchard
- XML Schema Part 0: Primer
  02 May 2001, David C. Fallside
- XML Schema Part 1: Structures
  02 May 2001, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn
- XML Schema Part 2: Datatypes
  02 May 2001, Paul V. Biron, Ashok Malhotra
- Canonical XML Version 1.0
- Extensible Markup Language (XML) 1.0 (Second Edition)
  6 October 2000, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler
  This specification is a revision of the XML 1.0 Recommendation published on 10 February 1998.

Eine vollständige Liste aller zz. verabschiedeten und in Arbeit befindlichen Standards, die auf XML aufbauen, findet sich in http://www.w3.org/TR/Overview.

Die genannten Standards regeln zunächst nur die Verpackung der zu übertragenden Objekte. Die Struktur der Dokumente, die die komplette Operationsanforderung und –antwort beschreiben, ist zusätzlich festzulegen, ebenso wie die Signalisierung von Ausnahmen (Fehlerinformation) und die Übertragung von Authentisierungsinformation mit der Operationsanforderung.

Eines der zz. am weitesten verbreiteten Protokolle, das diesen Regelungsbedarf befriedigt, ist das *Simple Object Access Protocol* (SOAP). Dies ist zwar noch nicht standardisiert (die Standardisierungsbemühung steht noch in der Entwurfsphase), aber die weite Verfügbarkeit von Software dafür macht es zu einem guten Kanddaten für ein prototypisches Szenario einer Kollaboration von Web Services und Web-Applikationen.

SOAP übermittelt die Operationsanforderungen zwischen Objekten wie gesagt dadurch, dass Objektidentifikation, Name der Operation und Werte der Parameter in ein SOAP-spezifisches XML-Dokument verpackt werden und dann über HTTP an den Server gesendet werden, der das gewünschte Objekt beherbergt.

Im Detail funktioniert das so: Auf dem Server A befinde sich ein Objekt X, das eine Operation O von einem Objekt Y verlangt, welches sich jedoch auf einem Server B befindet.

Die Nutzung von HTTP bedingt, dass der Server B zunächst einmal über eine URL identifizierbar sein muss, also z.B. als http://zentrale.strasse.de. Der von Y offerierte Dienst selbst wird durch Ansprache einer Ressource auf diesem Server angestoßen: http://zentrale.strasse.de/preisdoku.asmx

Die Typen der Objekte X und Y seien z.B. *KBK* und *KBK_Preisdoku,* die Operation *GibPreis(kbkid, menge, mass).*

Auf dem Server A befindet sich eine Implementation des Typ *KBK_Preisdoku*, die selbst gar nicht die Preisdokumentation ist, sondern die die Operationsanforderungen an die Preisdokumentation in ein XML-Dokument verpackt. In diesem Beispiel gibt es dort also eine Methode *GibPreis(kbkid, menge, mass)*, die folgendes (etwas vereinfachte) Dokument erzeugt und dann an den Server B schickt, wenn das *KBK*-Objekt auf Server A diese Methode aufruft:

```
<SOAP-ENV:Envelope>
   <SOAP-ENV:Body>
     <namespace:GibPreis xmlns:namespace=”urn:preisdoku”>
      <kbkid xsi:type=”okstra:KBKid”>333040</kbkid>
      <menge xsi:type=”xsd:float”>2658.0</menge>
      <mass  xsi:type=okstra:Masseinheit>Quadratmeter</mass>
     </namespace:GibPreis>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ein derartiger Softwarebaustein, der eine Klasse lokal simuliert, die in Wirklichkeit weit weg auf einem anderen Computer implementiert ist, nennt sich *Proxy*. Der Proxy sendet also dieses XML-Dokument per HTTP-Request an den Server B, auf dem ein XML-fähiger Webserver installiert ist, und wartet auf eine Antwort von dort.

Der Webserver entpackt das Dokument und macht daraus einen Aufruf einer Methode oder Prozedur aus einer wie auch immer implementierten Bibliothek (binär z.B. als .NET, COM-, CORBA- oder Java Bean-Komponente, oder als Script in Perl, PHP, Python,Tcl). Die Methode wird nun abgearbeitet und erzeugt ein Ergebnis, das vom Webserver ebenfalls in ein XML-Dokument umgewandelt wird und dem Proxy auf A als HTTP-Response zurückgesendet wird. Der Proxy wandelt das XML-Ergebnisdokument in die für seine Programmumgebung passende Form um und gibt das Ergebnis an das Objekt X zurück.

A.4.2 Komponentenbildung

Die Gesamtlogik des Prototypen gliedert sich nach dem bekannten 3-Ebenen-Client-Server-Modell (bzw. MVC-Modell) in Geschäftslogik (Controller), Präsentationslogik (View) und Datenlogik (Model).

Für die Geschäftslogik wurde natürlich das Modell des Kapitel 3 herangezogen, denn dieses galt es ja zu implementieren.

Die Datenlogik ist für den Prototypen von sehr geringer Komplexität. Sie wurde durch sehr einfache Datenbank- bzw. Datei-Strukturen realisiert. Sie hat die Aufgabe, die Persistenz der Objekte sicherzustellen und wurde folglich so implementiert, dass jedem Objekttyp eine Datenbanktabelle entspricht, die ein 1:1 Abbild der inneren Objektstruktur ist. Auf Einzelheiten wird wegen dieser einfachen Strukturierung nicht näher eingegangen.

Danach musste der Funktionsumfang des Prototypen bezüglich der Anwendungsfälle aus Kapitel 2 festgelegt werden. Nicht alle Anwendungsfälle der Kostenberechnung konnten berücksichtigt werden. Die verbleibenden wurden in Szenarien entwickelt. Daraus wurden dann vier Web-Applikationen abgeleitet, die das Benutzerinterface und einige sekundäre Teile der Geschäftslogik, wie z.B. Summenbildungen, enthalten.

A.4.2.1 Web Services

Das Modell des Kapitel 3 wurde sodann in Komponenten gruppiert, wobei jeweils enge Kollaborationen von Klassen zusammengefasst wurden:

Komponente OO-EG-Manager

Enthält die Klassen: EG_Manager, Variante, Trasse, Lärmschutz (als Beispiel für ein von der Trasse abhängiges Fachobjekt, siehe folgendes UML-Diagramm).

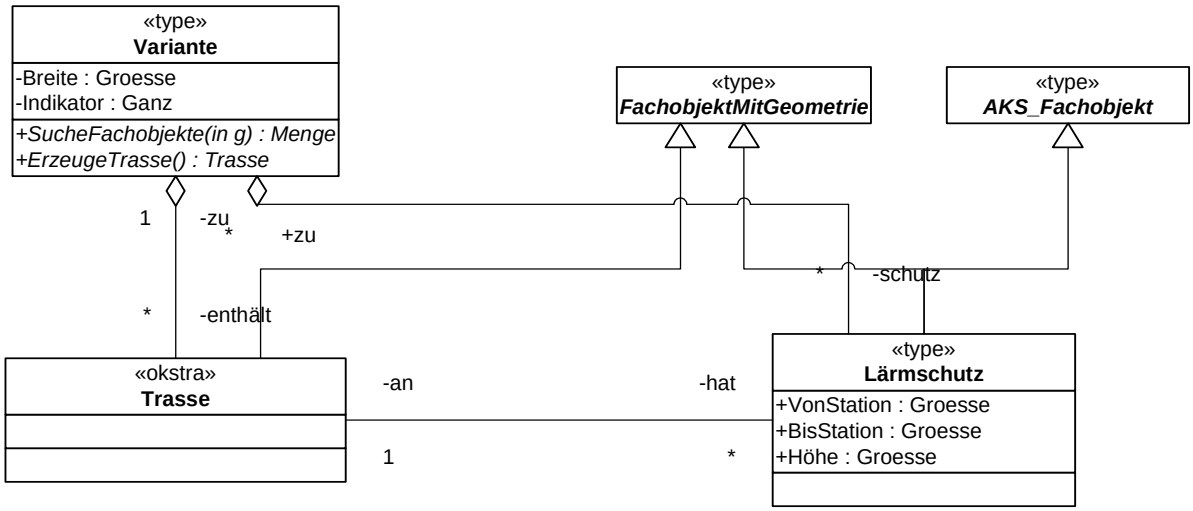


Abbildung 28. Fachobjekte im Prototypen.

Die Verwendung von OKSTRA®-konformen Objekten wäre zu wünschen gewesen. Eine für das Konzept *Variante* passende Objektart gibt es jedoch noch nicht. Auf die Verhältnisse bezüglich der Klasse *Lärmschutzbauwerk* ist weiter oben bereits eingegangen worden.

Komponente OO-GE-Manager

Enthält die Klassen: *GE_Manager, GE_Maßnahme, Flurstück, Erwerbsfläche*.

Komponente OO-VP-Manager

Enthält die Klassen: *VP_Manager*. Die Klasse wurde nachträglich eingeführt, um eventuell den Anwendungsfall Kostenschätzung zu integrieren, allerdings unterblieb ein weiterer Ausbau, weil Kostenschätzung von den befragten Fachleuten als intuitiver, nicht automatisierbarer Anwendungsfall charakterisiert wurde, so dass hier nur Dokumentationshilfe angeboten werden kann. Der Dienst ist über die Web-Applikationen OO-AP-Base ansprechbar, die Auslegung ist jedoch fachlich nicht sehr gelungen.

Komponente MAVZ

Enthält die Klasse: *MVZE* (implementiert die Klasse BaumaßnahmeVerzeichnis des Modells).

Die Verwendung eines anderen Namens hat keine besonderen technischen Gründe, sie ergab sich aus der Weiterentwicklung eines vorab realisierten Testaufbaus zur Ansprache von Java-basierten Web Services und wurde beibehalten, um fehleranfällige manuelle Umbenennungen zu vermeiden.

Komponente OO-Massnahme

Enthält die Klasse: *Massnahme* (implementiert die Klasse *Baumaßnahme* des Modells)

Komponente OO-AKS-Manager

Enthält die Klassen: *AKS_Manager, AKS_Berechnung, AKS_Teil, KBK, KBK_Position*

Komponente OO-KBK-Prototyp

Enthält die Klasse: *KBK_Prototyp*

Komponente OO-KBK-Preisdoku

Enthält die Klasse*: KBK_Preisdoku*

Sonstige Klassen

Die Klassen des Rahmenmodells, die benötigt wurden – für die Handhabung von Ansammlungen, Ereignissen und für die zentrale Namensvergabe – wurden nach Bedarf in die Komponenten integriert, die sie nutzen.

Zu jeder Komponente gehört außerdem eine *Interface-Klasse*, deren Operationen nach außen sichtbar sind. Diese Klasse delegiert die an sie gerichteten Nachrichten an die Kollaboration, die in der Komponente implementiert ist. Operationen des Modells, die nicht von anderen Komponenten benötigt wurden, sind auch nicht auf dem Interface der Komponente sichtbar. Sofern *Factory*-Operationen benötigt wurden, um Objekte von außen erzeugen zu können, wurden diese ebenfalls in der Interface-Klasse angesiedelt.

Diagramm

Die herausgefundenen Komponenten sind im UML-Komponentendiagramm unten grafisch dargestellt.

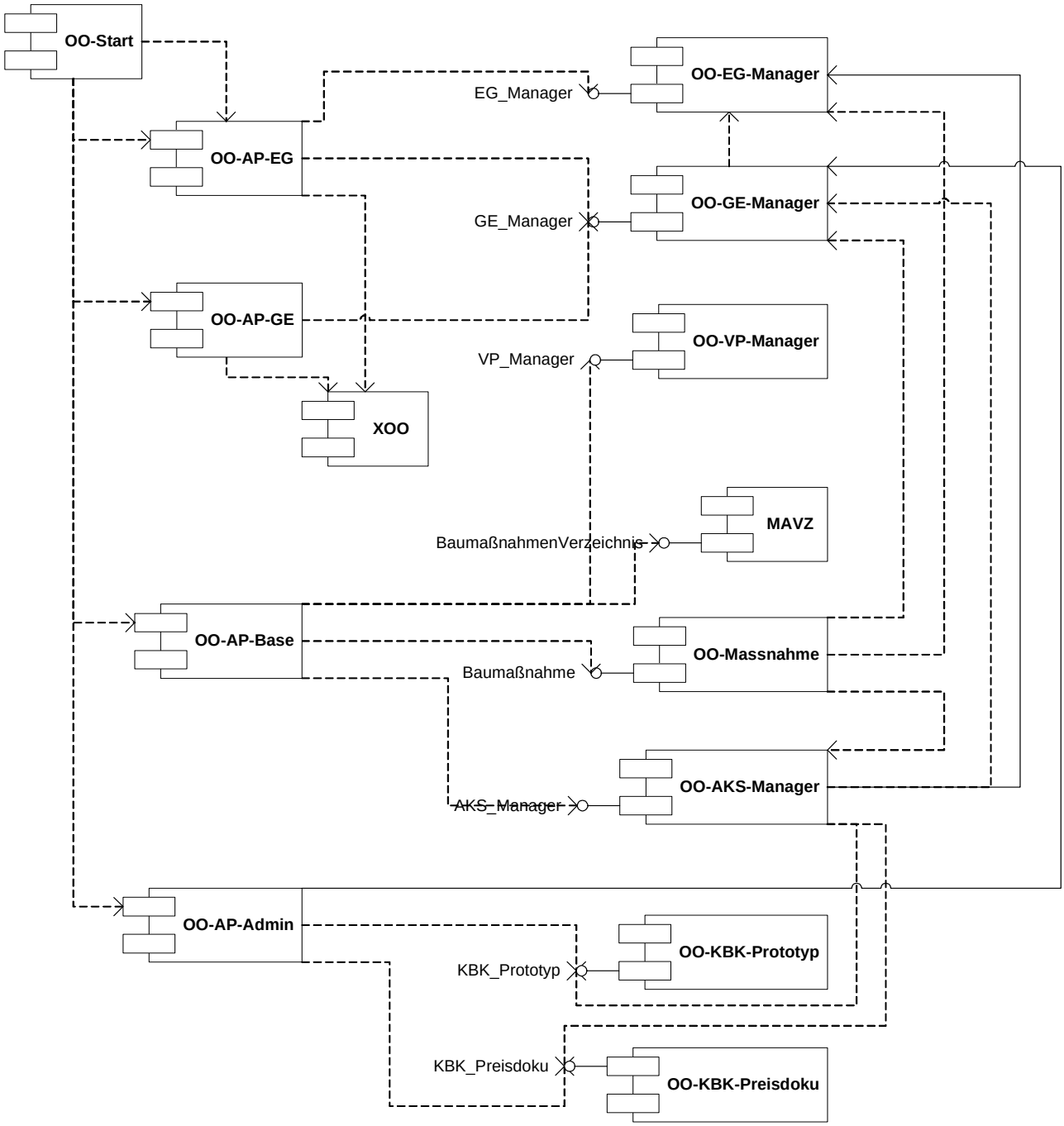


Abbildung 29. Komponenten des Prototypen.

A.4.2.2 Die Web-Applikationen

Um zu demonstrieren, wie die implementierten Dienste für die Erledigung von Aufgaben aus dem untersuchten Prozess Kostenberechnung genutzt werden können, wurden vier Web-Applikationen implementiert. Zunächst wurden die Anwendungsfälle des Modells in Szenarien entwickelt.

Dabei mussten bestimmte Teilaufgaben ausgeklammert werden. So wurde keine komplette AKS-Berechnung erstellt, da das bedeutet hätte, viele zusätzliche Fachobjekte zu definieren, ohne dass sich daraus neue Erkenntnisse ergeben hätten. Ebenfalls verzichtet wurde auf eine Definition von räumlichen Teilen nach AKS. Hierzu hätte eine wesentlich reichhaltigere Menge von *FachobjektenMitGeometrie* simuliert werden müssen; es hätten

umfangreiche grafische Editiermöglichkeiten eingebaut werden müssen. Um dies zu vermeiden, wurde für die Änderung der Trassengeometrie eine grobe Simulation durch lineare Verzerrung der Geometrie realisiert. Die Entwicklung eines noch so einfachen webbasierten grafischen Editors ist eine nichttriviale, aufwändige Sache, die zum Thema der Forschungsarbeit nichts beigetragen hätte.

Interessant wäre die Anbindung eines Web Map Servers für die Kartierung des Hintergrundes der simulierten Trassenobjekte gewesen; eine entsprechende Untersuchung führte zu dem Ergebnis, dass .NET-basierte Applikationen Web Map Server nach OGC-Standard zz. nicht komfortabel nutzen können, weil es für letztere zur Zeit der Untersuchung (Oktober 2001) noch keine brauchbaren Interface-Beschreibungen in WSDL gab.

Komponente OO-AP-Base

Diese Applikation soll folgende Anwendungsfälle simulieren:

- Maßnahmen neu einrichten und löschen
- Einsicht in das Maßnahmenverzeichnis (MAV)
- Öffnen einer Maßnahme für eine Kostenberechnung nach AKS
- Neuerstellung einer Kostenberechnung
- Abrufen und Anzeigen der aktuellen Kostenberechnung. Benachrichtigung, falls diese nicht mehr gültig ist, weil sich die Datengrundlage geändert hat.
- Fortschreiben der Kostenberechnung (Neuberechnung auf Grund der geänderten Datengrundlage)
- Historisierung der Kostenberechnung (Ablegen nicht mehr aktueller Berechnungen zu Dokumentationszwecken)
- Steuerung des Projektzustandes

Dazu folgende *Szenarien*:

- Maßnahme einrichten: Es erscheint ein Web-Form mit Eingabemöglichkeit für PROJIS-Nr, Straßenbezeichnung, VKE-Bezeichnung, geschätzter Länge und Kostenansatz (siehe BVWP). Der OO-Massnahme-Dienst wird aufgefordert, die Maßnahme anzulegen, das Maßnahmenverzeichnis wird entsprechend fortgeschrieben. Die neue Maßnahme ist im Zustand OP (ohne Planungsauftrag).
- Maßnahmen anzeigen: Suchauftragsfelder für Straßen-Nr. und VKE-Bezeichnung (ungenaue Suche!). Aus dem MAV werden die entsprechenden PROJIS-Nr. ermittelt und es werden die zugehörigen Maßnahmen gesucht und mit ihren Daten in einer Tabelle angezeigt.
- Maßnahme löschen: Das Massnahme-Objekt wird beseitigt, das Maßnahmenverzeichnis aktualisiert.
- In der Tabelle kann eine Maßnahme ausgewählt werden. Ein Knopf Kostenermittlung schlägt die Kostenermittlungsseite auf. Außerdem ist eine Möglichkeit einzurichten, den Projektzustand zu ändern. Es werden folgende Projektzustände unterstützt:
    - OP: ohne Planungsauftrag. Es existieren keine Varianten. In diesem Zustand werden als Kosten einfach die Ansätze des BVWP nachgewiesen.

- VP: Vorplanung hat begonnen. Der *EG_Manager* zur Maßnahme wird erzeugt. Ab hier können Varianten erzeugt werden, vorher nicht. Die Kostenschätzung verwendet die Länge der Variante in km sowie einen in der Variante gehaltenen Indikator, der über eine Tabelle in M€/km umgerechnet wird.
- VE: Vorentwurf hat begonnen. *GE_Manager* und *AKS_Manager* werden erzeugt. Die Kosten werden nach der AKS berechnet. Ab diesem Zustand können Grunderwerbsobjekte eingerichtet werden.
- PA: Planfeststellung beantragt. Es sind nur noch lesende Operationen möglich.

Die Manager-Objekte müssen eine Möglichkeit haben, diesen Zustand zu erfragen.

- Sind keine Kostenermittlungen vorhanden, wird eine Tabelle der existierenden Varianten angeboten, mit Knöpfen, um die Ermittlung zu starten.
- Auf der Kostenermittlungsseite wird eine Tabelle der existierenden Kostenermittlungen angezeigt. Gezeigt wird: Art der KE(S=Schätzung, B=Berechnung), Variantenbezeichnung, Aufstellungsdatum, Gültigkeit (aktuell, historisch, ungültig), Baukosten, GE-Kosten, Gesamtkosten.
- Für Kostenberechnungen können die Zustandsübergänge: aktuell -> ungültig oder historisch -> historisch ausgelöst werden. Historische Kostenberechnungen können nur noch angezeigt, nicht mehr aktualisiert werden.
- Ein Knopf gestattet die Detailanzeige einer Kostenermittlung. Ist diese ungültig, wird gefragt, ob eine Neuberechnung vorgenommen werden soll. Bei positiver Antwort wird diese durchgeführt.
- Zur Berechnung wird ein *KBK* benötigt. Standardwert ist der bei der vorherigen Berechnung für dieselbe Variante verwendete, sofern eine solche existiert, ansonsten wird ein interaktiv wählbarer Prototyp als Muster eingesetzt. Der benötigte KBK wird als Kopie erzeugt. Um den KBK verwenden zu können, müssen Preise eingetragen werden. Dazu wird in die KBK-Positionen entweder eine Verknüpfung (URL) eingetragen, welche Preisdoku verwendet werden soll, oder es wird manuell ein Einheits- oder Pauschalpreis eingetragen. Es wird eine Möglichkeit angeboten, manuell einen solchen einzutragen.
- Beim Übergang der Maßnahme in der Zustand VE wird der *GE_Manager* erzeugt. Er muss sich zunächst die Flurstücke des Baugebietes besorgen. Im Prototypen erledigt man dies durch eine feste Menge von simulierten Flurstücken.
- Für die Ermittelung der GE-Kosten befragt die Kostenberechnung den *GE_Manager* unter Nennung der Variante. Dieser hält alle evtl. in Frage kommenden Flurstücke gegen die Variantengeometrie und ermittelt daraus die GE Kosten.

Komponente OO-AP-EG

Diese Komponente soll folgendes erlauben:

- Öffnen der Maßnahme für Entwurfserstellung und -änderung
- Öffnen und Ändern von Entwurfsobjekten mit Änderungs-Nachricht, d.h.
  - die Trassen-Geometrie einer Variante laden, anzeigen und ändern
  - ein straßenbegleitendes Bauwerk hinzufügen und verändern

- Anzeige von Flurstücken zum Entwurf

Szenarien:

- Die Eingangsseite bietet wie bei OO-AP-Base die Möglichkeit der Einsicht in das MAV und der Auswahl einer Maßnahme.
- Die Entwurfslogik gestattet zunächst die Erzeugung einer Variante. Dies übernimmt der *EG_Manager*. Die Geometrie wird aus einer angebbaren Datei oder als Kopie der Geometrie einer vorhandenen Variante erzeugt.
- Zur Bearbeitung einer Variante kann aus einer Liste eine ausgewählt werden. Sie wird durch Befragen des *EG_Managers* erstellt. Jede Variante kennt die Originalgeometrie und die in Bearbeitung befindliche. Im Ruhezustand gibt es nur die Originalgeometrie. Bei Beginn der Bearbeitung wird die Originalgeometrie in die Bearbeitungsgeometrie kopiert und die Originalgeometrie wird präsentiert. Eine Variante mit aktiver Bearbeitungsgeometrie kann nicht ein zweites Mal zur Bearbeitung geöffnet werden.
- Um die die Geometrie der Variante zu ändern, kann ein Vektor V mit Länge und Winkel vorgegeben werden. Die Änderung wird an die Komponente XOO (siehe weiter unten) delegiert. Nach Aufforderung wird die geänderte zusammen mit der originalen Geometrie präsentiert. Jede Änderung wirkt immer auf die Bearbeitungsgeometrie. Ein Knopf gestattet das Rücksetzen der Bearbeitungsgeometrie auf das Original. Bei Abschluss der Bearbeitung wird die Bearbeitungsgeometrie zur neuen Originalgeometrie. Wurde die Geometrie geändert, wird die Kostenberechnung verständigt. Nach Änderung wird der alte und der neue Zustand präsentiert.
- OO-AP-EG soll die Flurstücke im Baugebiet präsentieren können. Dazu befragt sie den *GE_Manager*. Die zurückerhaltenen Polygone werden über XOO dargestellt.
- Um die Kostenberechnung für Fachobjekte zu demonstrieren, erzeugen man interaktiv ein Lärmschutzbauwerk-Objekt (Anforderung wird an *EG_Manager* unter Angabe der Variante delegiert). Die Ausdehnung wird durch Anfangs- und Endstation angegeben, zusätzlich wird die Höhe eingetragen. Das Objekt wird mit der Trasse verknüpft. Die Bauwerksgeometrie wird nicht im Objekt gehalten, sondern bei Bedarf aus der Geometrie des Trassen-Objektes generiert
- OO-AP-EG besorgt sich die Geometrie des Bauwerks und präsentiert dieses.
- Beim Ändern der Geometrie der Variante wird das Bauwerk auf Wunsch mit geändert (Skalierung der Stationen um NeueLänge/AlteLänge).
- Das Bauwerk kann auch manuell neue Enden sowie eine neue Höhe bekommen, so dass die Kostenberechnung mit einer anderen KBKid arbeitet. Jede Änderung des Bauwerks wird der Kostenberechnung signalisiert.

Komponente OO-AP-GE

Diese Komponente soll folgendes erlauben:

- Nachweis der von einer Variante berührten Flurstücke
- Ermittlung des betroffenen Anteils (Erwerbsfläche) der Flurstücke
- Nachweis der Kosten für die Erwerbsflächen
- Preisänderung für Erwerbsflächen mit Nachricht an den *AKS_Manager*.

Wie weiter oben dargestellt, müssen an sich nur die Gesamtkosten nachgewiesen werden. Um die Wirksamkeit der Ereignislogik besser zu demonstrieren, wurden jedoch Einzelkosten ermittelt und nachgewiesen. Wird die Trasse geändert, so dass teils neue Flurstücke betroffen sind, teils andere nicht mehr betroffen sind, so lässt sich dies damit gut verdeutlichen.

Szenarien:

- OO-AP-GE soll eine Tabelle von Erwerbsflächen zur Variante erzeugen können. Der *GE_Manager* findet das zur Variante gehörende *GE_Maßnahme*-Objekt. Dieses berechnet die Erwerbsflächen samt Kosten. Jede Flurstücksfläche des Baugebietes wird mit dem Flächenschlauch um die Geometrie des Trassenobjektes der Variante verschnitten. Als Rückgabe bekommt man die Erwerbsfläche in qm. Die Anzeige enthält: Flurstücksnummer, Flurstücksfläche, Preis pro qm (wie man ihn z.B. aus einer Bodenrichtwertkarte übertragen könnte), Erwerbsfläche in qm, Kosten für die Fläche in €. Gesamtkosten.
- Um Priese für Erwerbsflächen zu ändern, werden die Flurstücke als Liste angezeigt mit der Möglichkeit, eines auszuwählen und den zugehörigen Preis zu ändern.
- Eine Schnittstelle in *GE_Maßnahme* ist einzurichten, um nur die Kosten zu bekommen. Diese wird von der Kostenberechnung angesprochen.

Komponente OO-AP-Admin

Diese Komponente soll folgendes erlauben:

- Einrichten einer Preisdokumentation.
- Eintragen von mengenabhängigen Einheitspreisen in die Dokumentation
- Ändern von Preisen und Nachricht, dass Kostenberechnung dadurch u.U. zu aktualisieren.
- Konfigurieren eines Kostenberechnungskatalog-Prototypen: Eintragung von den KBK-Nummern, die der Prototyp enthalten soll; Entfernen solcher Einträge

Szenarien:

- Die Applikation kann eine *KBK_Preisdoku* auswählen, anzeigen und ändern. Sie hat die Form einer Tabelle mit: KBK-Nr., Einheit, Beschreibung, Menge, Preis/Einheit
- Bei der Durchführung der Kostenberechnung kann erkannt werden, dass ein Preis fehlt. Dies soll durch eine Fehlermeldung und Abbruch der Berechnung angezeigt werden.
- Bei Änderung einer Preisdoku wird der *AKS_Manager* benachrichtigt, um die darauf basierenden aktuellen Kostenberechnungen mit einem Hinweis zu versehen, dass sie u.U. neu zu erstellen sind.
- Die Applikation kann einen *KBK_Prototypen* auswählen und als Tabelle mit KBK-Nr., Leistungsbeschreibung, Maßeinheit anzeigen.
- Es kann auf Knopfdruck eine neue KBK-Nr. samt Beschreibung eingetragen werden.
- Aus der Tabelle kann eine KBK-Nr. entfernt werden.

Komponente XOO

OO-AP_EG und OO-AP-GE verwenden eine Komponente XOO, die zur Behandlung und Präsentation von Geometrien dient. Diese Komponente kann:

- Eine Geometrie aus dem Inhalt einer Geometrie-Datei erzeugen. Geometrien werden über Geo-Identifikatoren identifiziert.
- Eine Geometrie ganz oder teilweise kopieren und dabei nach links oder rechts parallel versetzen.
- Eine Geometrie deformieren: Jeder Punkt der Geometrie wird um 2*a*V verschoben, wobei a = min(Abstände zu den Enden)/Länge und V ein Einheits-Vektor.
- Eine Geometrie in einer Datei speichern
- Die Länge einer Geometrie mitteilen
- Die Fläche des Durchschnittes eines Polygons mit einem Schlauch wählbarer Breite um eine Geometrie berechnen.
- Eine Geometrie mit einer Bildebene verknüpfen
- Eine Geometrie aus einer Bildebene herauslösen
- Ein Bild aus den Geometrien seiner Bildebenen erzeugen
- Eine Hintergrundbildebene mit Flurstücksinformation (Polygonen) füllen.

Die Bildebenen:

- Aktuelle Trasse
- Geänderte Trasse
- Aktuelles Bauwerk
- Geändertes Bauwerk
- Hintergrund

Komponente OO-Start

Um bequem auf alle Applikationen über eine URL zugreifen zu können, wurde zusätzlich eine Start-Seiten-Komponente mit vier Navigations-Knöpfen erstellt.

A.4.3 Verteilung auf Standorte

Eines der Anliegen des Prototypen ist es, zu zeugen, dass auch weit voneinander entfernte verteilte Komponenten zusammenarbeiten können. Es wurde letztlich folgende Verteilung hergestellt:

Server 1

Standort: IT-Zentrum Landschaftverband Westfalen-Lippe, Münster

Basissoftware:

- OS: MS Windows 2000 Server
- RDBMS: MS SQL Server 2000
- Webserver: MS Internet Information Server
- Web-Technologie: ASP.NET Beta 2

- Programmiersprache: C#

Installierte Komponenten: OO-AP-GE, OO-GE-Manager, KBK-Preisdoku, XOO

Server 2

Standort: Bundesanstalt für Straßenwesen, Bergisch Gladbach

Basissoftware:

- OS: SuSE Linux 7.3
- RDBMS: mySQL
- Webserver: Apache, Tomcat, Apache SOAP Kit
- Web-Technologie: Java Servlets, Java Web Services
- Programmiersprache: Java

Installierte Komponenten: MAVZ

Server 3

Standort: interactive instruments GmbH, Bonn

Basissoftware:

- OS: MS Windows 2000 Workstation
- RDBMS: MS SQL Server 2000
- Webserver: MS Internet Information Server, Apache Proxy Server
- Web-Technologie: ASP.NET Beta 2
- Programmiersprache: C#

Installierte Komponenten: OO-Start, OO-AP-Base. OO-AP-EG, OO-AP-Admin, OO-Massnahme, OO-EG-Manager, OO-VP-Manager, OO-AKS-Manager, KBK-Prototyp, XOO

Die Konzentration der Komponenten auf Server 3 ist nur dadurch begründet, dass alle Server zentral von Bonn aus administriert werden mussten, d.h. alle Software-Änderungen, die nach der Verteilung und Aufstellung auf die verschiedenen Server vorgenommen notwendig wurden, waren ferngesteuert durchzuführen. Um den Umfang dieser relativ langsamen Fernsteuerung zu reduzieren, wurde die Verteilung wie oben festgelegt.

Bei einem für die Produktion realisierten System würde jeder Server durch Personal vor Ort betreut werden können. Einer Verteilung nach Kompetenz und Kontrolle der Datengrundlage, wie es das eigentliche Ziel in verteilten Informationssystemen ist, würde dann nichts im Wege stehen.

Die Kommunikation der Server erfolgte über die bei den Teilnehmern vorhandenen Internet-Zugänge. Alle Server waren dem Internet gegenüber durch vorgeschaltete Paket-Filter bzw. Firewalls geschützt. Der Server beim Auftragnehmer wurde zudem durch Delegation über einen Proxy Server abgeschirmt.

Die Verhältnisse sind nochmals grafisch in folgendem Verteilungsdiagramm dargestellt.

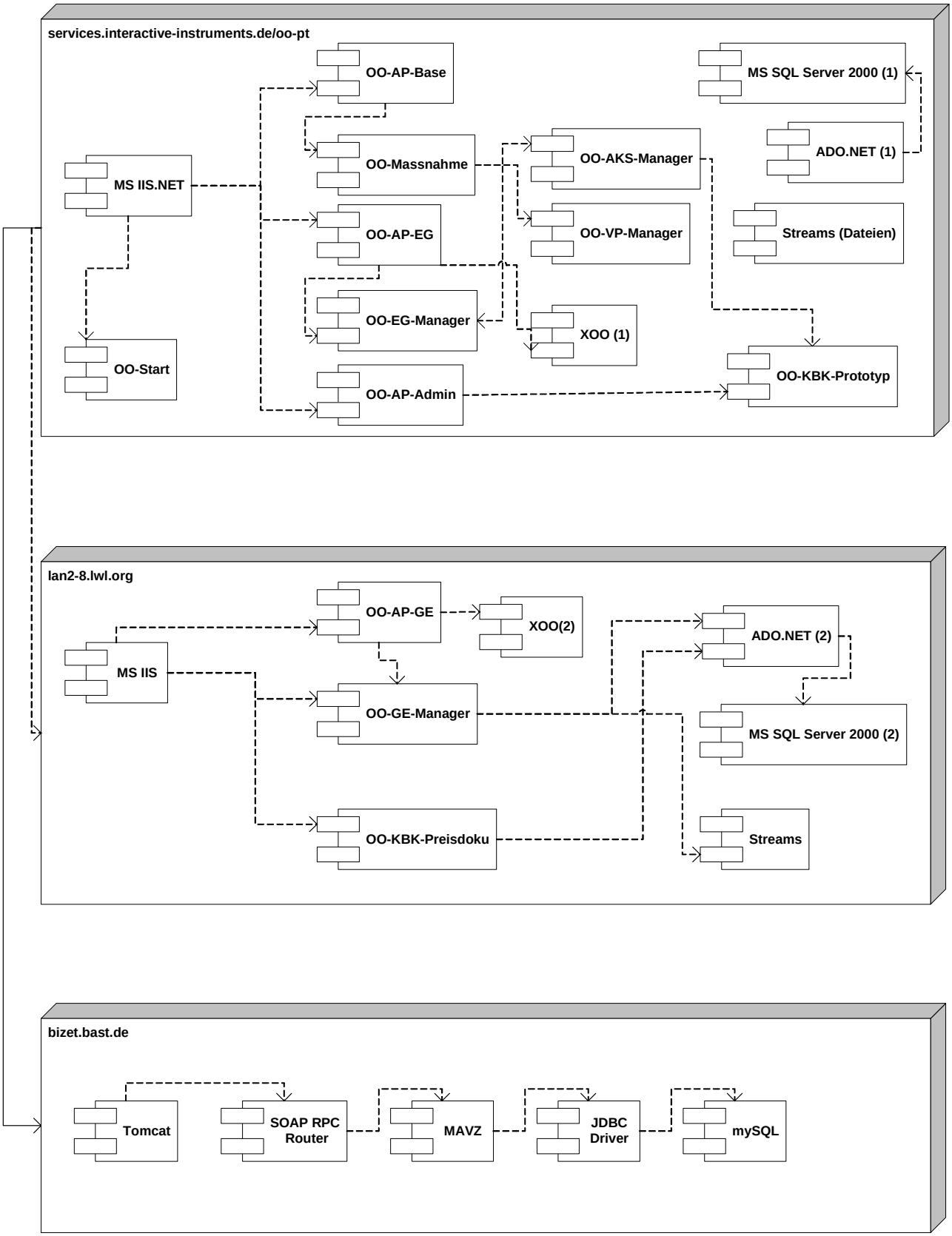


Abbildung 30. Verteilung der Komponenten.

A.5 Zusammenfassung

Hier folgt nochmals eine verkürzte Gesamtdarstellung des Arbeitsablaufs der Modellierung des Prozesses Kostenberechnung. Zwischendurch sind Absätze mit Bewertungen der verschiedenen Phasen eingefügt.

- Identifizierung der relevanten Regelwerke und ihre Analyse, um Anwendungsfälle zu finden; Befragung von Fachleuten, um Unterschiede zwischen aktueller Arbeitsweise und Beschreibung im Regelwerks ausfindig zu machen. Solche Unterschiede existieren einerseits wegen unzureichender Spezifikation innerhalb des Regelwerks, andererseits wegen mangelnder Handhabbarkeit. Ein Beispiel für letzteres ist die praxisferne Kategorisierung der Grunderwerbskosten in der (AKS85).
- Identifikation der benötigten Datenbasis. Im vorliegenden Falle sind dies Mengen und Preise. Mengen müssen aus den Datenbeständen anderer Geschäftsbereiche gewonnen werden (z.B. Straßenentwurf, Grunderwerb). Folglich müssen Kommunikationspfade zu diesen Geschäftsbereichen existieren.
- Strukturierung und Dokumentation aller gefundenen Anwendungsfälle als UML-Diagramm. Diskussion und Abstimmung des Anwendungsfall-Modells. Verabschiedung des Modells.

Diese Phase der Modellierung verlief weitgehend problemlos. Die zu Rate gezogenen Fachleute konnten wertvolle Hinweise zur Ausgestaltung des Modells geben und waren auch in der Lage, das am Ende gewonnene Anwendungsfallmodell zu prüfen.

Die nächste Phase war die Entwicklung des Objektmodells.

- Aufbau der Objektwelt. Fachobjekt-Typen, Prozessobjekt-Typen, Ereignisse
- Ausstattung der Objekt-Typen mit Operationen.
- Dokumentation als UML-Klassen- und Sequenzdiagramm.

Beim Aufbau des Objektmodells fiel zunächst auf, dass die meisten der für die Kostenberechnung benötigten Fachobjektklassen im OKSTRA nicht existieren oder in einer Form vorliegen, die die benötigte Information nicht bereitstellt. Die Vermittlung des Modells an die Mitglieder des betreuenden Arbeitskreises gestaltete aus sich aus einer Reihe von Gründen jedoch unerwartet schwierig:

- Im objektorientierten Modell tauchen Attribute und Relationen nicht mehr als eigene Konstrukte auf, sondern als besondere Operationen. Bei Attributen ist diese konzeptionelle Übersetzung noch klar, die Wiedergabe von Relationen führt jedoch zu abstrakten, als zu „mathematisch-informationstechnisch“ empfundenen Kostrukten wie Mengen, Folgen usw. Die ersten Modellansätze berücksichtigten diese Objekttypen noch explizit, zusammen mit Verwaltungstypen wie Iteratoren, die zur Bearbeitung solcher Objekte dienen. Für die in diesem Dokument vorgestellten Modelle wurden diese Typen implizit in den Operationen versteckt, die die Relationen repräsentieren. Dadurch wird das Modell leichter verständlich, es ist stärker konzeptionell geprägt, andererseits müssen die benötigten Konstrukte (Mengen, Folgen usw.) auf Spezifikationsniveau jedoch im Modell definiert sein, weil sonst die Schnittstellen nicht vollständig beschrieben werden können.
- Es entstehen im Modell Typen, die keinen ohne weiters evidenten Fachbezug haben, z.B. die Typen *AKS_Manager, GE_Manager*. Diese entsprechen auf den ersten Blick keinen realen Objekten, mit denen es der Planer zu tun hat, und werden daher als „konstruiert“ und verkomplizierend empfunden. Da trügt jedoch der Schein, diese Ob-

jekttypen repräsentieren vielmehr *Tätigkeiten* des Planers und deren Ergebnisse. Wo immer nicht nur statische Objektbezüge, sondern Arbeitsabläufe modelliert werden, ist mit dem Auftauchen solcher Objekte zu rechnen.

- Auf ähnliche Art ist das Erscheinen von Typen, die *gar keine* Objekte repräsentieren, zunächst ungewohnt. Typen *wie AKS_Fachobjekt, FachobjektMitGeometrie* gehören dazu. Diese Typen sind abstrakt, d.h. man kann keine Objekte dazu herstellen, sondern nur andere Typen daraus ableiten. Sie repräsentieren bestimmte gewünschte Eignungsprofile. So ist ein AKS_Fachobjekt ein Fachobjekt, das für eine AKS-Berechnung einen Beitrag leistet und deshalb muss die Berechnungsoperation es befragen können.

Des öfteren tauchte auch die Frage auf, wie man eigentlich die Operationen findet, und besonders gerade die im Modell. Hierzu einige Bemerkungen.

Zunächst ist das dargestellte Design ist nicht das einzig mögliche. Im Allgemeinen wird man bestrebt sein, die Typen so aufzubauen, dass jede Operation sich nur mit einer einzigen, sehr gut definierten Aufgabe beschäftigt. Aber auch dann bleiben noch Freiheiten übrig. Einige Designrichtlinien finden sich im *Leitfaden LF Modell 02*.

Die Ausstattung der Objekttypen mit Operationen ergibt sich vielfach aus der Aufgabenstellung, d.h. dem Anwendungsfall, unmittelbar, so bei *AKS_Manager, FachobjektMitGeometrie*. Generell lassen sich die Operationen einteilen in solche zur Erzeugung, zur Vernichtung, Inspektion und Modifikation von Objekten.

Anmerkung: Zur Inspektion gehören neben den einfachen Observator-Operationen (die einen einzelnen Wert zurückbringen) natürlich auch Such- und Berechnungsfunktionen.

Aus dieser groben Klassifizierung lässt sich häufig schon ersehen, zu welchem Typ welche Operationen gehören. Für die meisten der im Modell gezeigten Operationen ist das bereits ausreichend. Ein anderer gängiger Weg ist die Ausarbeitung eines Szenarios für einen Arbeitsgang.

Anmerkung: Dies kann man vorwärts oder rückwärts angehen. Die Vorwärts-Strategie geht von den Eingangsinformationen aus und schreitet vorwärts bis zum Ergebnis, die Rückwärts-Strategie geht vom Ergebnis aus und fragt, welche Informationen dafür gebraucht werden und woher man sie bekommt.

Noch ein Beispiel: Die Operationen des Typs *AKS_Fachobjekt* ergeben sich aus folgenden Überlegungen: Es dürfen erstens nur Operationen sein, die explizit etwas mit der AKS zu tun haben, denn der Typ soll ja gerade das enthalten, was für die AKS-Berechnungen gebraucht wird. Zweitens ist die vom Fachobjekt benötigte Information eine Menge. Drittens ist auf Grund der Struktur des Kostenberechnungskataloges die Beziehung zwischen Katalogposition und Objekttyp aber nicht 1:1. Welche Position betroffen ist, ergibt sich auf der einen Seite z.B. aus Dimensionierungen und Materialart, also *Attributen* des Fachobjekts, die sich im Planungsverlauf ja auch ändern können, auf der anderen Seite daraus, ob mit *Global*- oder mit *Detailpositionen* im KBK gearbeitet wird. In der Konsequenz braucht man also eine Operation, die einem die KBK-Nummern mitteilt, zu denen Information vorliegt, und zum anderen eine, die abhängig zur KBK-Nummer die Mengenangabe mitteilt. Ein alternatives Design käme mit nur einer Operation aus, die dann beides leisten würde durch Abliefern einer kleinen Tabelle aus Paaren (KBK-Nummer, Menge).

Die Aufstellung der Sequenzdiagramme dient in aller erster Linie dem Zweck, nachzuprüfen, ob die gestellte Aufgabe mit dem erarbeiteten Operationsinventar gelöst werden kann.

Zur Entwicklung des Prototypen:

- Zuerst werden die Klassen in *Kollaborationen* eingegliedert. Aus jeder Kollaboration wird dann eine *Komponente* mit eigener Schnittstelle. Diese ergibt sich aus der Zusammenfassung der Operationen der in der Komponente implementierten Klassen. Die Klassen, die die Typen des Modells implementieren, werden im Regelfall neben den als Operationen des Modells geforderten Methoden noch zusätzliche enthalten.
- Nach der Komponentenbildung können diese implementiert und getestet werden. Am Ende wird jede Komponente auf dem ihr zugedachten Server installiert. Zur Demonstration müssen dann noch die Anwendungen entwickelt werden.

Leider mussten alle OKSTRA-Objekte simuliert werden, teils weil die Definitionen im OKSTRA fehlen oder zu schwach ausgelegt sind, teils, weil keine für den Zweck brauchbaren Daten verfügbar waren. Dies fiel insbesondere für die Mengenberechnung auf. Die ursprünglich ins Auge gefasste Intergration von REB-Methoden erwies sich daher und wegen der Struktur der existierenden REB-Software als nicht praktikabel, weil zu aufwändig.

Bei der Demonstration fiel insbesondere der Nutzen des Ereignis-Benachrichtigungssystems positiv auf. Überzeugend wirkte auch der Nachweis plattformübergreifender Kommunikation mittels Web-Protokollen. Die provisorische Natur der verwendeten Entwicklungsplattform führte jedoch leider auch dazu, dass einige Anwendungsfälle nicht fehlerfrei demonstriert werden konnten.

A.6 Literatur

Kürzel	Quelle
AKS 85	Bundesminister für Verkehr, Abt. Straßenbau: Anweisung zur Kostenberechnung von Straßenbaumaßnahmen AKS 1985. Abgedruckt in: (Straßenbau AZ)
And 01	Anderson,R. et al., Professional ASP.NET. Wrox Press Ltd.. 2001.
ARS 13/90	Bundesministerium für Verkehr: Kostenaufstellung und -fortschreibung bei Baumaßnahmen an Bundesfernstraßen. Rundschreiben 13/1990. Abgedruckt in: (Straßenbau AZ)
Boo 99	Booch,G., J.Rumbaugh, I.Jacobson, The Unified Modeling Language User Guide. Addison Wesley. 1999.
Fow 00	Fowler,M., K.Scott, UML Distilled, Second Edition. Addison Wesley. 2000
GAEB 99	Verfahrensbeschreibung für die elektronische Bauabrechnung, GAEB-VB 23.004 Allgemeine Mengenberechnung. 1999. Beuth Verlag, Berlin.
Gam 95	Gamma,E., R.Helm, R.Johnson, J.Vlissides, Design Patterns. Addison Wesley. 1995
GP-Neu 02	Geschäftsprozesskatalog zum Prozess Neubaumaßnahme. Bericht zu FE 09.119. 2002. interactive instruments, Bonn.
HOAI 96	Honorarordnung für Architekten und Ingenieure. Bundesanzeiger Verlag, Köln, 1995.
HVA F-StB 99	Handbuch für die Vergabe und Ausführung von freiberuflichen Leistungen der Ingenieure und Landschaftsarchitekten im Straßen- und Brückenbau (HVA F-StB). Stand 08/99.
KBK-STLK 00	Arbeitskreis AKS/STLK der Straßenbauverwaltungen von Bund und Ländern: Verbesserungsmöglichkeiten für die Erarbeitung von Kostenfortschreibungen sowie die Überführung von STLK-Positionen in KBK-Nummern. Bericht über die Ausschusstätigkeit. 2000. Bonn.
LF Modell 02	Leitfaden zur objektorientierten Modellierung des OKSTRA. Bericht zu FE 09.119. 2002. interactive instruments, Bonn.
OKSTRA 95	Erstling, Reinhard, und Clemens Portele: Standardisierung graphischer Daten im Verkehrswesen Teil 1 – Studie. Forschung Straßenbau und Straßenverkehrstechnik, Heft 724.
OKSTRA 00	Portele, Clemens, und Dietmar König: Standardisierung graphischer Daten im Verkehrswesen – Realisierung. 2000. Teilbericht A: Forschung Straßenbau und Straßenverkehrstechnik, Heft 782. Teilbericht B - Ergebnisse der Teilprojekte: http://www.okstra.de/docs.html
OesOOSE 98	Oestereich, Bernd: Objektorientierte Softwareentwicklung. 1998. Oldenbourg Verlag, München, Wien.
RAB-BRÜ 92	Bundesministerium für Verkehr: Richtlinien für das Aufstellen von Bauwerksentwürfen. Loseblattsammlung. Verkehrsblatt-Verlag, Dortmund.
REB 97	Sammlung REB Regelungen für die elektronische Bauabrechnung (Sammlung). 1997. FGSV Verlag, Köln, FGSV-Nr. 929.
Smi 02	Smith,R.E., Authentication. Addison Wesley. 2002

Sne 02	Snell,J., D.Tidwell, P.Kulchenko, Programming Web Services with SOAP. O'Reilly. 2002
UML 99	OMG Unified Modeling Language Specification Version 1.3. 1999

Anmerkung: [LF Modell 02] ist das Vorgängerdokument zum hier vorliegenden Leitfaden.

Anhang B
Modellierung von Fortführungsprozessen

Verantwortlich für den Inhalt:

Dipl.-Ing. Nikolaus Kemper
Dipl.-Ing. Michael Paetzmann
Niedersächsisches Landesamt für Straßenbau, Hannover

Dipl.-Phys. Bernd Weidner
interactive instruments GmbH, Bonn

Redaktion: Bernd Weidner (interactive instruments)

B.1 Einleitung

B.1.1 Motivation

Der vorliegende Leitfaden wurde auf Grund der Erfahrungen verfasst, die bei der Erstellung des Geschäftsprozesskataloges [GP-Neu 02] und der Modellierung der Kostenermittlung (Anhang A) gewonnen wurden. Die in diesem Anhang dokumentierte Modellierung diente dem Nachweis, dass die im Leitfaden niedergelegte Methodologie auch für die Lösung aktueller Fragestellungen aus der praktischen Arbeit anwendbar ist.

Zur Mitarbeit konnten Fachleute des Niedersächsischen Landesamtes für Straßenbau (NLStB) gewonnen werden.

B.1.2 Ziele und Umfang

Herr Kemper (NLStB Hannover) schlug als konkretes Projekt die Modellierung von Objekten des Bestandes u.a. im Hinblick auf das Projekt TURIN (Tank- und Rastanlagen-Informationssystem Niedersachsen) vor. Die Modellierungsziele hierfür sind:

- Untersuchung der Fortführungsprozesse für Bestandsdaten. Die nebeneinander existierenden Informationssysteme für Bestandsdaten verwenden je nach Anwendung unterschiedliche Darstellungen für dieselben realen Objekte. Hier sind die Anforderungen für eine Harmonisierung zu untersuchen und Modelle zu entwickeln, die dieselben Objekte für alle Anwendungen nutzbar machen.
- Untersuchung des Lebenszyklus der betroffenen Objekte. Es ist zu untersuchen, wo die Objekte zuerst entstehen und wie sie sich bis hin zur Bestandsfortführung entwickeln müssen, damit die auf dem Bestand operierenden Anwendungen arbeiten können. Hierbei ist insbesondere der Informationstransport in der Kette Planung-Ausführung-Bestandsdokumentation-Anwendung (d.h. TURIN) zu untersuchen.

B.1.3 Vorgehensweise

Die durch den Leitfaden vorgegebene Vorgehensweise wurde zunächst wie folgt konkretisiert:

1. Zunächst werden die betroffenen Informationssysteme (Datenbanken, existierende Anwendungssoftware usw.) ermittelt. Außerdem werden alle relevanten Dokumente (Regelwerke, Dienstanweisungen usw.) ermittelt. Die im Prozess tätigen Mitarbeiter werden nach Bedarf befragt, welche Aufgaben sie darin wahrnehmen. Auch die Auswahl der Werkzeuge für die Modelldokumentation ist hier zu leisten. (Angestrebt wird die Nutzung eines Produktes, das UML grafisch darstellen kann und als XML-Datei ausgeben kann.)
2. Daraus werden die Anwendungsfälle abgeleitet. Sie werden in UML dokumentiert.
3. Die Kommunikationskanäle zu benachbarten Bereichen werden untersucht. Dies betrifft den Datentransport (Wo werden Daten her bezogen?) sowie insbesondere Ereignisse (Wann werden Anwendungsfälle ausgelöst?). Der Gesamtablauf wird als Aktivitätsdiagramm dargestellt. Die Kontaktperson wird zu deren Erstellung und Fortschreibung angeleitet.

4. Die Objektwelt wird aufgebaut. Es werden nach den Regeln des Leitfadens alle benötigten Objektarten zusammengetragen. Nutzbare Objekte aus dem bestehenden OKSTRA werden identifiziert.
5. Die Rolle der Objekte in den Anwendungsfällen sowie ihr Lebenszyklus – auch außerhalb der Anwendungsfälle – wird untersucht. (Wer ist für die Erzeugung zuständig? Wer füllt die Objekte mit der in den Anwendungsfällen benötigten Information? Was geschieht mit dieser Information bei der Anwendung?). Diese Untersuchung beantwortet die Frage, welche Verantwortlichkeiten jedes Objekt hat (z.B. Auskünfte erteilen, andere Objekte sammeln, Fassade für ein Informationssystem exponieren...) und damit, welche Operationen es erlauben muss, wie diese aussehen und was sie leisten müssen.
6. Das Klassendiagramm für die Objektwelt wird entwickelt.
7. Die Objekte werden zu Komponenten zusammengefasst, die die benötigten Dienste anbieten. Soweit möglich werden die Komponenten mit bestehenden Informationssystemen identifiziert. Über die hierfür zu fordernden Schnittstellen können dann die für die Anwendungsfälle benötigten Applikationen transparent und integriert die jeweils notwendigen Informationen beziehen bzw. fortführen. Der Endanwender würde idealerweise pro Fortführungsfall nur eine Aktion durchführen müssen, die alle beteiligten Informationssysteme parallel nachführt.
8. Endgültige Aufbereitung und Dokumentation des gesamten Modells

B.2 Ergebnisse der Arbeitsphasen

B.2.1 Geschäftsbereichsanalyse

B.2.1.1 Bestimmung des Geschäftsbereichs

Laut Leitfaden beginnt die Modellierung mit der Analyse des zu untersuchenden Geschäftsbereichs, der seinerseits aus einem Geschäftsprozesskatalog bestimmt wird. Im vorliegenden Falle finden wir im Geschäftsprozesskatalog Neubaumaßnahme (GP-Neu 02) auf Seite 40 unter der Bezeichnung [AG.4.9.2.1] den Eintrag „Bestand dokumentieren“ mit Erläuterungen in den Abschnitten 3.4.3.41-47. Bestandsdokumentationen sind aber sicherlich nicht nur für Neubaumaßnahmen notwendig, sie betreffen generell alle Maßnahmen, die die Straße betreffende Objekte (Anm.: Wasserschutzgebiete, Flurstücke) verändern.

Es erscheint daher angebracht, von einem Geschäftsbereich *Bestandsdokumentation* auszugehen, der als Faktorprozess als Bestandteil verschiedener Prozessketten auftaucht (von denen zz. jedoch nur für die Tätigkeiten zu einer Neubaumaßnahme ein Geschäftsprozesskatalog vorliegt). Eine Aufstellung aller für die Bestandsdokumentation relevanten Geschäftsprozesskataloge unterblieb aus Zeit- und Kostengründen.

B.2.1.2 Bestimmung der Geschäftsprozesse des Bereichs

Der nächste laut Leitfaden abzuwickelnde Punkt ist die Identifizierung der Geschäftsprozesse. Hier ist erneut auf die Definition aus Davenport & Short (Sloan Management Review, Sommer 1990, S. 11-27) als „a set of logically related tasks performed to achieve a defined business outcome.“ zurückzugreifen, also eine Menge logisch verknüpfter Arbeitsschritte, um ein definiertes Geschäftsziel zu erreichen.

Die Aufgabe einer Bestandsdokumentation im Straßen- und Verkehrswesen ist:

> Es ist eine Datenbasis herzustellen und aufrecht zu erhalten, die Beschreibungen der Objekte des Straßen- und Verkehrswesens enthält. Diese Datenbasis wird genutzt als Grundlage für Entscheidungen über Ausbau und Pflege des Straßennetzes und seiner Teile, woraus sich die Forderung nach *Vollständigkeit, Aktualität und Konsistenz* der Beschreibungen zu jeder Zeit ableitet.

Aus dieser Definition ergibt sich zunächst, dass die *Nutzung* der Datenbasis nicht Selbstzweck ist, sondern im Kontext von Prozessen anderer Bereiche stattfindet. Für diese ist die Datenbasis ein Hilfsmittel. Die Produktion von Statistiken und Plänen z.B. dient somit der Realisierung von Geschäftszielen anderer Geschäftsbereiche.

Als Prozesse, die zum Bereich Bestandsdokumentation selbst gehören bleiben damit übrig:

- Prozesse zur Aufrechterhaltung der physischen Infrastruktur der Datenbasis, also alle Tätigkeiten zum Schutz der Daten gegen Vernichtung. Diese Tätigkeiten sind nicht fachspezifisch für das Straßen- und Verkehrswesen und geben für die OKSTRA-Modellierung deshalb nichts her.

- Prozesse zur Pflege der Datenbasis, z.B. die Pflege der Straßeninformationsbank (SIB) und der Straßenbestandspläne (SBP).

Das kurze Beispiel aus dem zweiten Punkt zeigt bereits, dass die genannte Datenbasis aus mehreren unterschiedlich organisierten Teildatenbeständen besteht. Die Forderung nach Vollständigkeit, Aktualität und Konsistenz bedeutet aber, dass die Pflege eines Teildatenbestandes nicht unabhängig von den anderen betrachtet werden darf, denn die in den Teildatenbeständen repräsentierten Objekte können miteinander verknüpft sein – d.h. in der Terminologie des Leitfadens durch eine Assoziation verbunden sein – oder es kann sich bei den Repräsentationen um unterschiedliche Sichten auf ein und dasselbe Objekt handeln. Es lässt sich folglich als einziges Geschäftsziel der Pflegeprozesse festhalten:

> Die Bestandsdatenbasis des Straßen- und Verkehrswesens wird *fachlich* vollständig, aktuell und konsistent zur Verwendung durch andere Prozesse jederzeit bereitgehalten.

Damit gibt es prinzipiell im betrachteten Bereich nur einen fachlich relevanten Geschäftsprozess, der sich jedoch in einer Vielzahl von verschiedenen *Geschäftsvorfällen* ausdrücken wird. Dabei ist ein Geschäftsvorfall der durch ein äußeres *Ereignis* hervorgerufene Anstoß des Geschäftsprozesses mit bestimmten Startbedingungen, d.h. eine Instanz des Geschäftsprozesses.

Dieser Prozess wurde zunächst intuitiv und aus gängiger Praxis heraus mit *Fortführung von Bestandsdaten* bezeichnet.

Die nähere Analyse dieses Geschäftsprozesses und insbesondere seine Abgrenzung zu anderen Bereichen (z.B. Bau- oder Verkehrsplanung), die ebenfalls Datenbestände erzeugen und aktualisieren, erfordert jedoch eine genauere Bestimmung der verwendeten Begriffe *Fortführung* und *Bestandsdaten*. Diese Präzisierung ist notwendig, um im folgenden Schritt die zu berücksichtigenden IT-Verfahren, Datenbestände und Regelwerke identifizieren zu können.

Zunächst ist die Realisierung der angesprochenen Datenbasis ohne Belang. Es kann sich z.B. um Bestände von Hardcopy-Dokumenten wie Akten, Karteien, Karten und Plänen, um digitalisierte Karten und Pläne, um digitale Dokumente beliebigen Formats oder um Datenbanken handeln. Für den Fall nicht digitaler Daten müssen die Arbeitsschritte der Fortführung manuell vorgenommen werden, für digitale Daten sind daneben Datenübergabemechanismen unterschiedlichen Automatisierungsgrades möglich.

Ganz allgemein lassen sich *Fortführungsaktionen* durch folgende 3 möglichen Operationen (Elementarfunktionen) beschreiben:

1. Erzeugen eines neuen Objekts und Herstellung der geforderten Beziehungen zu anderen, schon länger bestehenden oder unmittelbar vorher erzeugten Objekten.
2. Beseitigen eines Objektes und Auflösung seiner Beziehungen zu anderen Objekten bzw. deren Beseitigung. Das „Beseitigen“ wird oft keine physische Löschung auf dem Datenträger implizieren, vielmehr ist hiemit ein (evtl. sogar reversibles) Ende der Verfügbarkeit des Objektes für aktuelle Aufgaben gemeint.
3. Ändern der Attribute und/oder Beziehungen eines Objektes und Wiederherstellung der Konsistenz in den verbundenen Objekten. (Anmerkung: Im Sinne des Leitfadens sind Attribute und Beziehungen (Assoziationen) - und übrigens auch in der Sicht von NIAM – nicht wesensverschieden)

Anmerkung: Die traditionelle Unterscheidung zwischen Ersterfassung und Fortführung bereits erfasster Daten führt nicht auf konzeptionell unterschiedliche Fortführungsaktionen, sie kann daher für die Begriffsbestimmung außer acht bleiben. Jedoch können Ersterfassungsvorgänge auf Grund anderer informationstechnischer Rahmenbedingungen zu besonderen Geschäftsvorfällen und damit besonderen Anwendungsfällen führen. Beispiele hierfür sind Datenmigrationen von alten in neue Systeme oder Erfassungsprojekte mit besonderen Instrumenten zur Massendatenerhebung.

Die Begriffsbildung *Bestandsobjekt* ist problematisch. In der objektorientierten Sicht haben viele Objekte einen Lebenslauf, der i.d.R. weit vor seiner physischen Realisierung beginnt. Es ist daher sinnvoller, statt von abgetrennten und eigenständigen Bestandsobjekten von Objekten im *Zustand Bestand* zu sprechen, wenn es sich um ein tatsächlich realisiertes Objekt handelt. Eine weitere Einteilung der Lebenszustände der Objekte ist durch die Werte: Prognose, Planung, Bauausführung, Bestand gegeben. Damit ist klar, dass die Objekte konzeptionell ihre Identität während ihres Lebens beibehalten. Wo eine Historisierung wünschenswert ist, kann ein so definiertes Objekt mit einer eigenen *Erinnerung* ausgestattet werden, das die vorher durchlaufenen Zustände dokumentiert.

Daneben gibt es eine Unterscheidung zwischen Objekten, die die *Struktur* des Straßennetzes beschreiben – diese sollen *Strukturobjekte* heißen –, und solchen, die die *Funktionstüchtigkeit* für den Verkehr im Netz charakterisieren – kurz *Funktionsobjekte*. Strukturobjekte sind zunächst die tangiblen (physischen, berührbaren) Bestandteile der Straßen und ihrer Ausstattung (z.B. die Beschreibung des Straßenaufbaus, Leitungen, Schilder usw.), sodann aber auch Abstraktionen zur Modellierung des Netzes und geometrischer bzw. geodätischer Sachverhalte (z.B. Abschnitte, Netzknoten, Achsen, Flurstücke). Funktionsobjekte sind z.B. Verkehrsstärken oder Unfallobjekte. Das laufende Projekt beschränkt sich im Sinne seiner Zielsetzung auf Strukturobjekte.

Der Begriff Bestandsobjekt wird also fortan als Strukturobjekt im Zustand Bestand interpretiert.

Die Aufgabe *Fortführung des Bestandes* besteht dem zu Folge auf elementarer Ebene aus:

- Erzeugen eines Objektes im Zustand Bestand – ein physisch schon vorhandenes Objekt muss in die Bestandsdatenbasis eingebracht werden. Beispiele ergeben sich aus Fällen von Aufstufungen, wobei Objekte des nachgeordneten Straßennetzes in das übergeordnete der klassifizierten Straßen übernommen werden müssen.
- Überführung eines Objektes in den Zustand Bestand – wobei u.U. eine Änderung der Repräsentation notwendig wird, die die nötigen Sichten für die weitere Nutzung der Objekte in Auswertungen usw. erlaubt.
- Ändern eines Objektes
- Löschen eines Objektes

Die Anwendungsfälle der Fortführung des Bestandes lassen sich durch Ketten solcher Aktionen ausdrücken.

Manchmal wünschenswert ist die Einbringung von Objekten in die Datenbasis auch, wenn ihre Realisierung gesichert ist, obwohl sie noch nicht tatsächlich erfolgt ist. Das ist insbesondere für Objekte im Zustand Baudurchführung der Fall. Bestimmte Anwendungen, z.B. aus dem Bereich Verkehrsanalytik, können solche Objekte in ihrer Funktion dann schon

teilweise nutzen. In der obigen Liste ist dann beim Erzeugen auch der Zustand Baudurchführung zulässig.

B.2.1.3 Datenbasis

Als nächsten Schritt verlangt der Leitfaden die Identifikation der Datenbasis, die im Bereich geführt wird Hierzu wurde eine Bestandsaufnahme aller im NLStB eingesetzten Verfahren und Programme durchgeführt. Das Ergebnis wurde in einer Tabelle zusammengefasst, die knap 100 verschiedene Verfahren bzw. Programme aufführt. Die große vorgefundene Vielfalt ist größtenteils historisch bedingt und zeigt die heterogene Datenbasis. (Die Tabelle ist hier nicht abgedruckt.)

Jeder Eintrag der Tabelle wurden zunächst danach klassifiziert, ob das entsprechende Verfahren etwas mit der Inventarisierung von Objekten zu tun hat – etwa im Gegensatz zu reinen Berechnungswerkzeugen oder zu Verfahren für Planung oder den AVA-Bereich (Ausschreibung, Vergabe, Abrechnung), der lt. Kap. 2.1.1 nicht zum untersuchenden Geschäftsbereich gehört.

Außerdem wurden die Verfahren nach einem „Maß der Koppelung“ beurteilt; dies ist ein qualitatives Merkmal, das angibt, wie stark sich das Verfahren auf Bestandsobjekte stützt. Der Klassifizierung liegt kein präzises Kriterium zu Grunde, sondern ein *educated guess*, also eine aus Erfahrung getroffene Einschätzung.

B.2.1.4 Kommunikationsverbindungen

Aus der Liste wurden nun die ausgewählt, die für die Führung der Bestandsdokumentation zuständig sind. Das ergab die folgende Tabelle. Die „Klassifizierung der Kopplung“ ist am Ende der Tabelle beschrieben.

Maßnahme/ Programm	Bezeichnung	Status	Klassifizierung für Kopplung
BAUMKAT	Baumschadenskataster	in Entwicklung	4
LSA_VERWALT	Verwaltung der Daten von Lichtsignalanlagen (Fachanwendung)	Projektidee	3
NUTZUNG	Verwaltung von Verträgen zur Nutzung bzw. Sondernutzung	in Nutzung	3
NWSIB	Straßeninformationsbank	in Einführung, in Fortentwicklung	4
SBP	Straßenbestands(pläne)	In Nutzung	4
TURIN	Tank- und Rastanlagen-Informationsystem	in Entwicklung	4
VEMAGS	Verfahrensmanagement Großraum- und Schwerlasttransporte	in Planung	3
Durchlasssammelbuch	Dokumentation der Durchlässe im Durchlasssammelbuch	Im Einsatz	4
Leitungen der SBV	Dokumentation der SBV eigenen Leitungen (AUSA, Stromzuführung für Verkehrsbeeinflussungsanlagen, ...)	Im Einsatz	4
WEGWEIS_VERWALT	Verwaltung der Daten von Wegweisern	Projektidee	3

Klassifizierung für die Kopplung der Prozesse

1. keine oder nur begriffliche Verbindung, d.h. die Verfahren tauschen keine Information über daran beteiligte Objekte aus,
2. ein Verfahren nutzt die Informationen aus Bestandsobjekten und muss über Änderungen daran nicht informiert werden, da sie mit dem aktuellen Zustand auskommt (die Nutzung ist „stateless“),
3. ein Verfahren nutzt die Information aus Bestandsobjekten, muss aber über Änderungen daran informiert werden, weil es eigene Sichten oder Objekte daraus ableitet,
4. ein Verfahren bearbeitet selbst Bestandsobjekte und muss deshalb über Änderungen informiert werden als auch selbst informieren.

B.2.1.5 Regelwerke und Verfahrensdokumentationen

Zu den Verfahren der Tabelle aus 2.1.4 wurden folgende Regelwerke und Verfahrensdokumentationen identifiziert:

Bezeichnung	Beschreibung
ASB	Anweisung Straßeninformationsbank, Teilsysteme Netzdaten, Bestandsdaten, vorl. Fassung 8/2002
ATB-BeStra	Allgemeine Technische Bestimmungen für die Benutzung von Straßen durch Leitungen und Telekommunikationslinien, Ausgabe 09/2002
BAUMKAT	Leistungsbeschreibung und Fachliches Konzept, NLStB IuK, 10.10.2002
LSA_VERWALT WEGWEIS_VERWALT	Kurze Notiz zum geplanten Inhalt
NUTZUNG	Handbuch zum Programm, 11/1993
Nutzungsrichtlinien	RdErl d. MW v. 14.1.1994 – 412-31023/1 –
NWSIB	Fortführung in der NWSIB mit 4 Szenarien für die Datenerfassung in exemplarischen Fällen, J. Rentsch, 05.09.2002 Online-Hilfe z. V2.01, 17.06.2002
OKSTRA	Schemata in der jeweils aktuellen Version
Pflege konkurrierender Datenbstände	Diagramm
SBP	Einführung der Richtlinien für die Herstellung und Fortführung von Straßenbestandsplänen, NLStB, 09.02.1990
SBP	Vergabeunterlagen zur Herstellung von Straßenbestandsplänen, NLStB Dez. 26, 05.2002
Signaturen	Graphische Darstellung der Kodierung, NLStB Dez. 26, 01.2000
TURIN	Konzept zur Datenfortführung, IP Syscon, 17.09.2001
VEMAGS	Fachliches Feinkonzept, PG 24, 29.10.2002
Vorgänge Sonstige Nutzung	3 Beispiele für Anträge zur Nutzung durch Leitungen (L477, Verlegung Gasleitung; B441, Kreuzung Abwasserleitung; L387, Kreuzung Trinkwasserleitung)

B.2.1.6 Kommunikation mit anderen Bereichen

Die Bestandsdatenbasis, bestehend aus SIB und den Persistenzdiensten (Dienste zur dauerhaften Speicherung von Objekten, z.B. in Datenbanken oder als Dokumente) der anderen bestandsrelevanten Fachinformationssysteme, stellt die Datengrundlage für eine Vielzahl von analysierenden und darstellenden Anwendungen aller Bereiche des Straßen-

und Verkehrswesens dar. Neben dem unmittelbaren Datenbezug ist hier zusätzlich ein Benachrichtigungsmodell erwünscht, das die daran teilnehmenden Prozesse informiert, sobald eine Fortführung stattgefunden hat (Kopplungstypen 3 und 4 oben).

Auf der versorgenden Seite finden wir die Bereiche Baudurchführung, Unterhaltung/Instandsetzung, Verkehrsdatenerfassung und Zustandsdatenerfassung.

B.2.1.7 Projektgruppe Modellierung

Der Leitfaden empfiehlt, Fachleute zur Modellierung der konzeptionellen Zusammenhänge heranzuziehen. Für das hier beschriebene Vorhaben konnten neben den Kontaktpersonen (Herrn Kemper, Herrn Paetzmann) insbesondere in der ersten Projektphase weitere Experter vom NLStB gewonnen werden:

Herr Jörg Fischer (SBA Wolfenbüttel)

Herr Jürgen Rentsch (Dez 11, NLStB)

Herr Dietmar Thomsik (Dez 22, NLStB)

Herr Volkmar Tönnies (Dez 11, NLStB)

Weitere Experten wurden und werden zu fachlichen Fragen herangezogen.

B.2.2 Einzelprozessanalyse

B.2.2.1 Rahmenbedingungen

Für den Fortführungsprozess ergibt sich aus dem vorigen Schritt im Effekt als gegenwärtige Situation, dass neben der Straßeninformationsbank, die den zentralen Datenbestand zum Netz und später auch zu anderen Beständen repräsentiert, mehrere komplexe und wichtige Systeme z.T. jüngsten Datums existieren, die eine andere und ergänzende Sicht auf die Bestandsobjekte liefern. Während z.B. für die Straßeninformationsbank eine Tank- und Rastanlage ein kaum strukturiertes Objekt entlang eines Abschnittes ist, stellt sie sich in TURIN als hochkomplexes, flächenhaftes, in zahlreiche Teilobjekte aufgelöstes Objekt dar. Diese Situation ist nicht etwa untypisch, sondern eher die Regel - spezialisierte Fachinformationssysteme ergänzen die Inhalte der SIB mit eigenen Informationen. Dem trägt auch der vorliegende Entwurf zur ASB Rechnung, in dem er für viele Objektarten des Bestandes *Objektnummern* als Referenzen auf eigene Teilsysteme vorsieht.

Dabei ist zu wünschen, dass ein Fortführungsvorgang, der sich auf *ein* Objekt bezieht, möglichst die Objektzustände, wie sie in allen Teildatenbeständen vorliegen, über *eine* Aktion von außen parallel verändert, um so Vollständigkeit und Konsistenz aufrecht zu erhalten. Da in der Praxis durchaus „langsame“ Systeme beteiligt sein können, die z.B. intern von Menschenhand gepflegt werden müssen, ist zu berücksichtigen, dass irgendeine Instanz Auskunft darüber geben muss, wie weit die Aktualisierung fortgeschritten ist und in wie weit sie die Nutzung des Objektes erlaubt.

In diesem Vorhaben sollen daher insbesondere solche Anwendungsfälle betrachtet werden, die sich auf mehrere Fachinformationssysteme auswirken. Für einen Einstieg wurden als Kandidaten die Anwendungsfälle für Leitungen und für Bäume ausgewählt. Die folgende Matrix beschreibt die diesbezügliche Situation.

	Leitungen	Bäume
NWSIB	x	x
SBP	x	
TURIN	x	x
BAUMKAT		X

B.2.2.2 Fortführung von Leitungen

Definition von Leitungen:

Leitungen dienen dem Transport von Gasen, Flüssigkeiten und Energie, z.B. Gas, Wasser, Abwasser, Strom, Fernwärme, Öl.

Die *Geschäftsvorfälle* oder *Ereignisse*, die zu einem Fortführungsbedarf bei Leitungsobjekten führen sind nach den Aussagen der hierzu befragten Fachleute:

1. Neuverlegung einer Leitung im Rahmen einer Neu- oder Ausbaumaßnahme. Lage und Ausführung der Leitung sind Unterlagen des für die Verlegung zuständigen Auftragnehmers zu entnehmen.
2. Neuverlegung einer Leitung auf Antrag eines Fremdbetreibers (Energieversorgung, Telekommunikation, Wasser- und Abwasserwirtschaft) mit Übergabe von Angaben zu Lage und Ausführung. Die Fortführung wird nach Schließung des Nutzungsvertrages notwendig.
3. Neuverlegung einer Leitung durch die Straßenbauverwaltung an bestehenden Straßen. Lage und Ausführung werden eigenen Unterlagen entnommen.
4. Übernahme bestehender Leitungen bei Aufstufung einer Straße aus dem nachgeordneten Netz. Lage und Ausführung der Leitung werden Unterlagen des vormaligen Baulastträgers entnommen.
5. Übernahme bestehender, aber undokumentierter Leitungen nach Entdeckung, deren Lage jedoch nur ungefähr bekannt ist
6. Umbau einer Leitung ohne Lageänderung, z.B. durch Neudimensionierung
7. Stilllegung oder Reaktivierung einer Leitung
8. Änderung administrativer Attribute, wie z.B. Name des Betreibers
9. Beseitigung einer Leitung durch Rückbau
10. Übernahme einer Leitung ins nachgeordnete Netz durch Abstufung mit Übergabe der Leitungsbeschreibung (Lage und Ausführung)
11. Der Umbau einer Leitung setzt sich zusammen aus dem Ausbau und dem Neueintrag. Ebenso ist eine Änderung technischer Daten, z.B. Änderung des Rohrdurchmessers, immer verbunden mit dem Ausbau und Neueinbau.

Diese Geschäftsvorfälle wurden von der Projektgruppe Modellierung als Anwendungsfälle formalisiert. Der Ablauf bei komplexeren Geschäftsvorfällen wurde in Form von Aktivitätsdiagrammen dokumentiert.

Die Arbeitsschritte für die Erledigung von Nr. 6-9 bilden sich unmittelbar auf Anwendungsfälle ab, die als sichtbares Ergebnis das Objekt nach der durchgeführten Änderung haben.

Bei der Konstruktion der Anwendungsfälle für die Fälle 1-5 ist zunächst zu beachten, dass die Lage neu aufzunehmender Leitungen entweder geometrisch mit hinreichender Präzision bekannt ist oder aber nur ungefähr über eine Netzknoten-Stationierungsangabe für den betroffenen Abschnitt. Ist die Lage genau genug bekannt, wird zunächst aus den übergegeben Unterlagen ein einheitliches Geometrieformat zu erzeugen sein, zweckmäßigerweise als OKSTRA-konforme Objekte. Für die Realisierung einer entsprechenden Anwendung ist daher ein Importschritt notwendig, der jedoch hier nicht im Detail untersucht werden kann und muss.

Die Unterscheidung zwischen der Leitung eines Fremdbetreibers und einer eigenen Leitung wurde nur vorgenommen, wo sie auf unterschiedliche Anwendungsfälle führt, d.h. wo eine unterschiedliche Behandlung erforderlich ist.

Bei Ereignis Nr.2 sind neben der Leitung selbst die Vertragsdaten zu übernehmen.

Somit können wir damit die nötigen Arbeitsschritte für 1-5 als Aktivitätsdiagramm wie folgt darstellen:

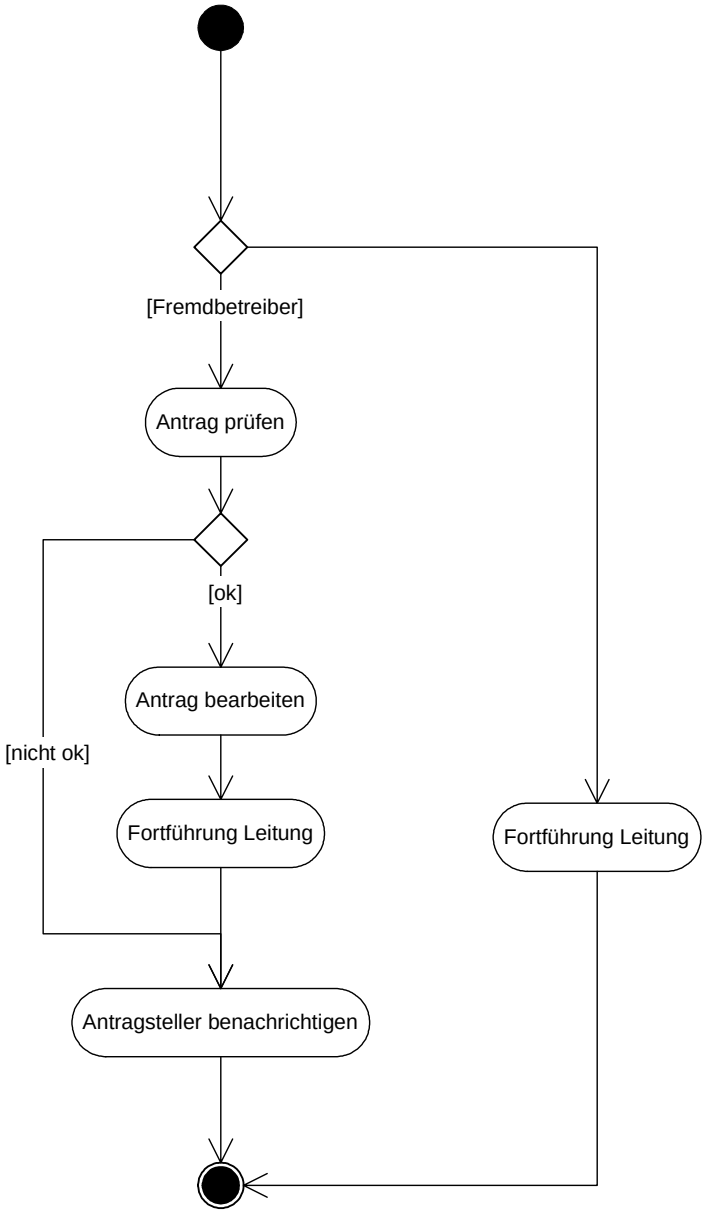


Abbildung 31. Leitungen erzeugen.

Die tatsächliche Fortführung verbirgt sich in der Aktivität *Fortführung Leitung*. Diese lässt sich selbst in Aktionen zerlegen. Zunächst wird die gesamte Fortführung in die der topologischen und geometrischen Repräsentation einerseits und der Sachinformation andererseits aufgespalten und zusätzlich der Fall eines Umbaus (Austausch, Sanierung) berücksichtigt:

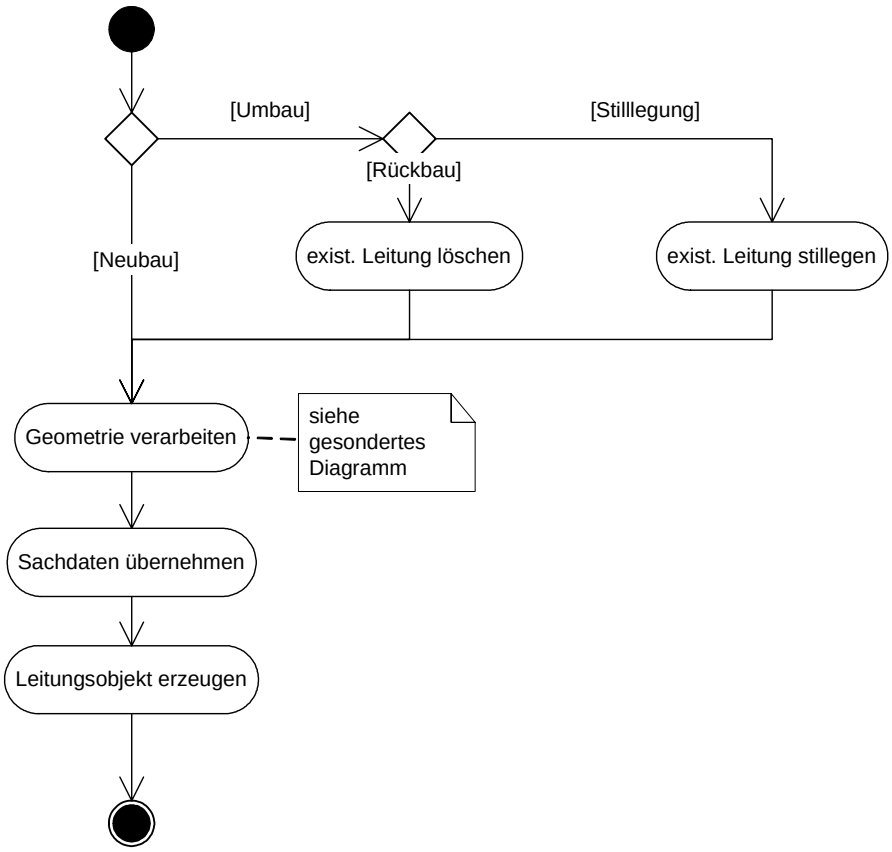


Abbildung 32. Fortführung Leitung.

Für Nr.10 ist zu beachten, dass Übergabeunterlagen erzeugt werden müssen; auch hierfür empfiehlt sich eine OKSTRA-konforme Übergabe.

Für die Geometrie ist zu beachten, dass sowohl Leitungen entlang einer Straße oder diese kreuzend als auch Leitungen, die abseits der Straße, z.B. in einer Tank- und Rastanlage, verlaufen. Im beiden Fällen kann die Geometrie in Form einer Folge von Landeskoordinaten vorliegen. Um den Netzbezug bei Leitungen an Straßen zu erzeugen, ist dann eine Umrechnung in das Netzknoten-Stationierungssystem vonnöten. Sind umgekehrt Stationierungsangaben vorhanden, ist daraus die Landeskoordinatenfolge zu ermitteln, um Systeme versorgen zu können, die sich auf Koordinaten stützen. Schließlich gibt es Sonderfälle, wie die Angabe einer Station zusammen mit einem Kreuzungswinkel.

Die Geometriepräsentation muss dem OKSTRA-Schema D018 (Geometrie) genügen, weshalb bei der Übernahme von Koordinatengeometrie eventuell eine Konvertierung nötig ist.

Die Geometrieübernahme stellt sich damit insgesamt in einem weiteren Diagramm wie folgt dar:

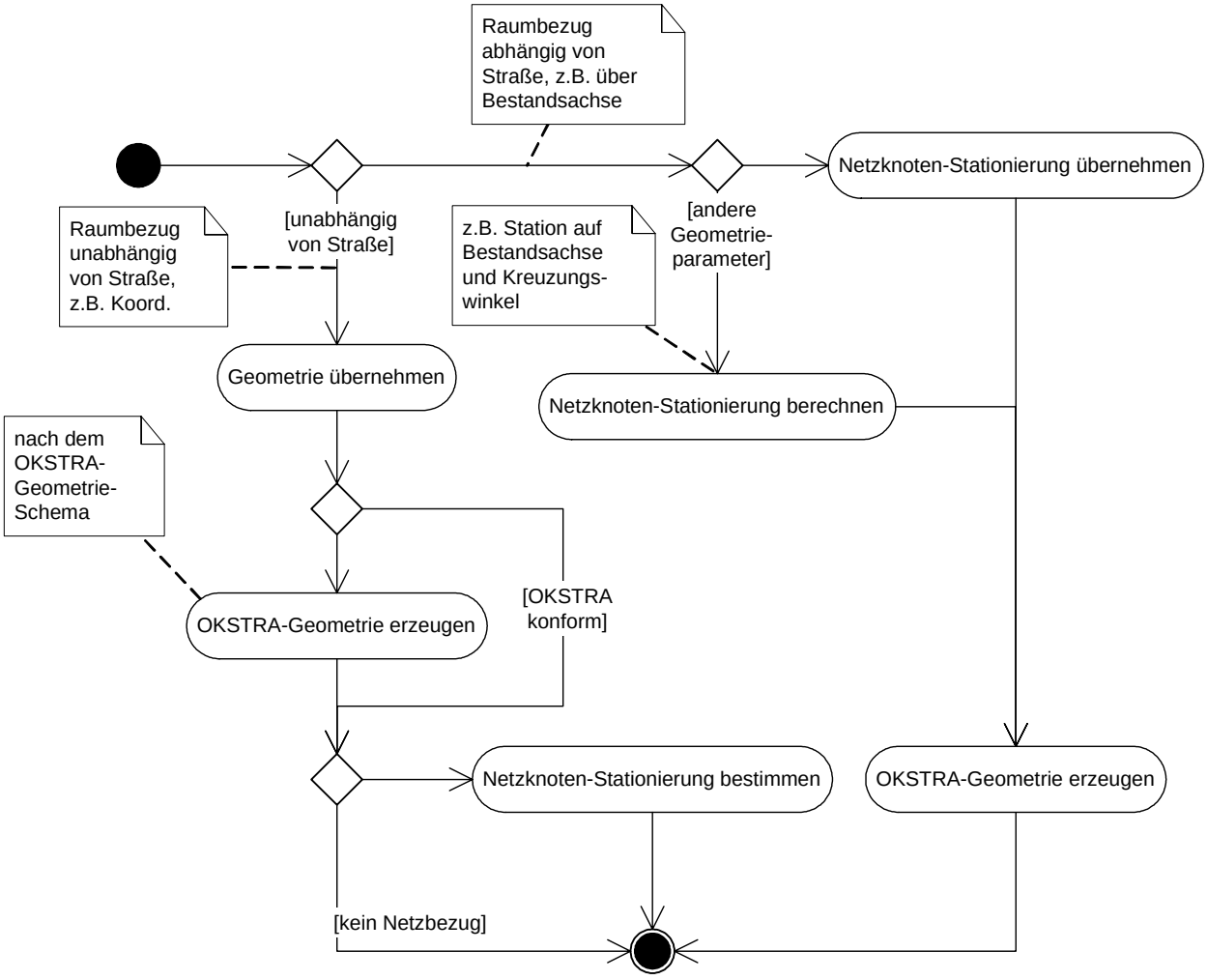


Abbildung 33. Geometrieübernahme.

Aktivitäten wie die Antragsprüfung und der Vertragsabschluss sind der Vollständigkeit halber aufgeführt, wir gehen davon aus, dass das Fortführungssystem sich erst mit vertraglich abgesicherter Information beschäftigt. Als Anwendungsfälle ergeben sich schließlich insgesamt die des folgenden Diagramms.

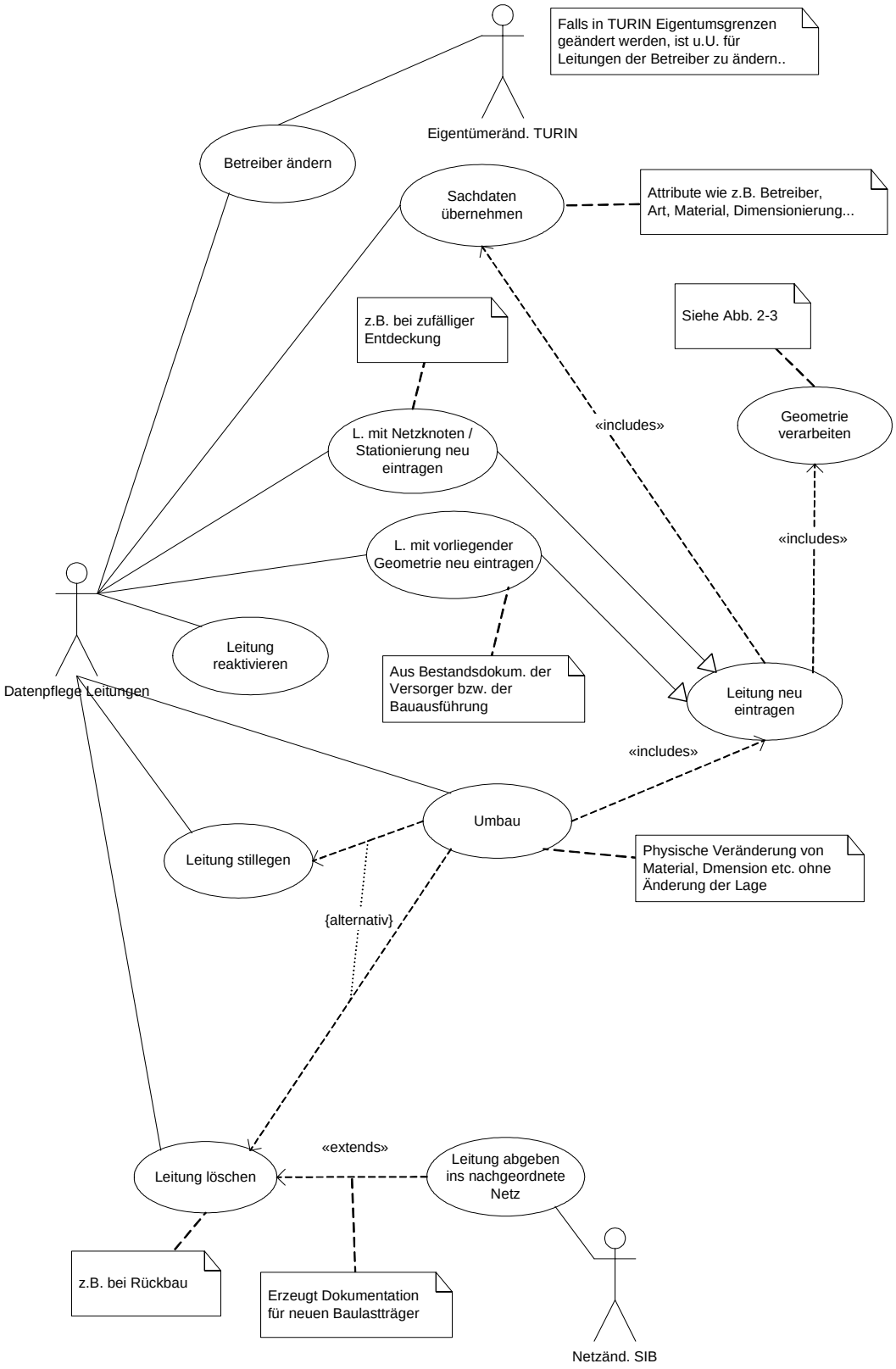


Abbildung 34. Anwendungsfälle Leitungen.

Die Akteure wurden wie folgt ermittelt:

Die *Datenpflege Leitungen* bezeichnet die allgemein für die Verwaltung der Leitungen zuständige Person oder Organsationseinheit. Es sind hierfür unterschiedliche organisatorische Realisierungen denkbar, z.B.

- Zentrales Fortführungsteam für alle oder ausgesuchte Objekte (oder alle Bestandsobjekte, oder...)
- Fortführung unmittelbar durch den für das Leitungswesen zuständigen Sachbearbeiter

Der Akteur *Netzänderung SIB* ist die NWSIB in ihrer Rolle als Netzänderungsinstrument. (Akteure dürfen ja auch IT-Systeme sein.) Die Kommunikation dieses Akteurs mit dem Anwendungsfall *Leitung abgeben...* ist dabei als automatische zu denken.

Fachinformationssysteme wie TURIN können ebenfalls von sich aus Fortführungsfälle auslösen. Im Diagramm wurde dies dadurch berücksichtigt, dass ein entsprechender Akteur eingetragen wurde.

B.2.2.3 Fortführung von Baumobjekten

Neben den Leitungsobjekten wurden als zweites Baumobjekte untersucht. Es ergibt sich, dass die Logik zu Erzeugung, Änderung und Löschung von Baumobjekten große Ähnlichkeiten mit den Verhältnissen bei den Leitungen aufweist:

- Die Geometriebehandlung kann völlig analog erfolgen.
- Bei Abgabe eines Abschnittes sind die Bäume ebenfalls in eine neue Zuständigkeit zu übergeben.

Ausgangspunkt für die Analyse ist das fachliche Konzept zur BAUMKAT-Anwendung. BAUMKAT kennt einen stationären Arbeitsmodus auf PC und einen mobilen Arbeitsmodus mit einer tragbaren Station. Nach den Aussagen des fachlichen Konzeptes unterscheiden sich die Bedienfunktionen nur für die Selektion (ist auf dem mobilen PC eingeschränkt), so dass dieselben Anwendungsfälle abgedeckt werden.

Das von der Projektgruppe erarbeitete Anwendungsfalldiagramm folgt hier:

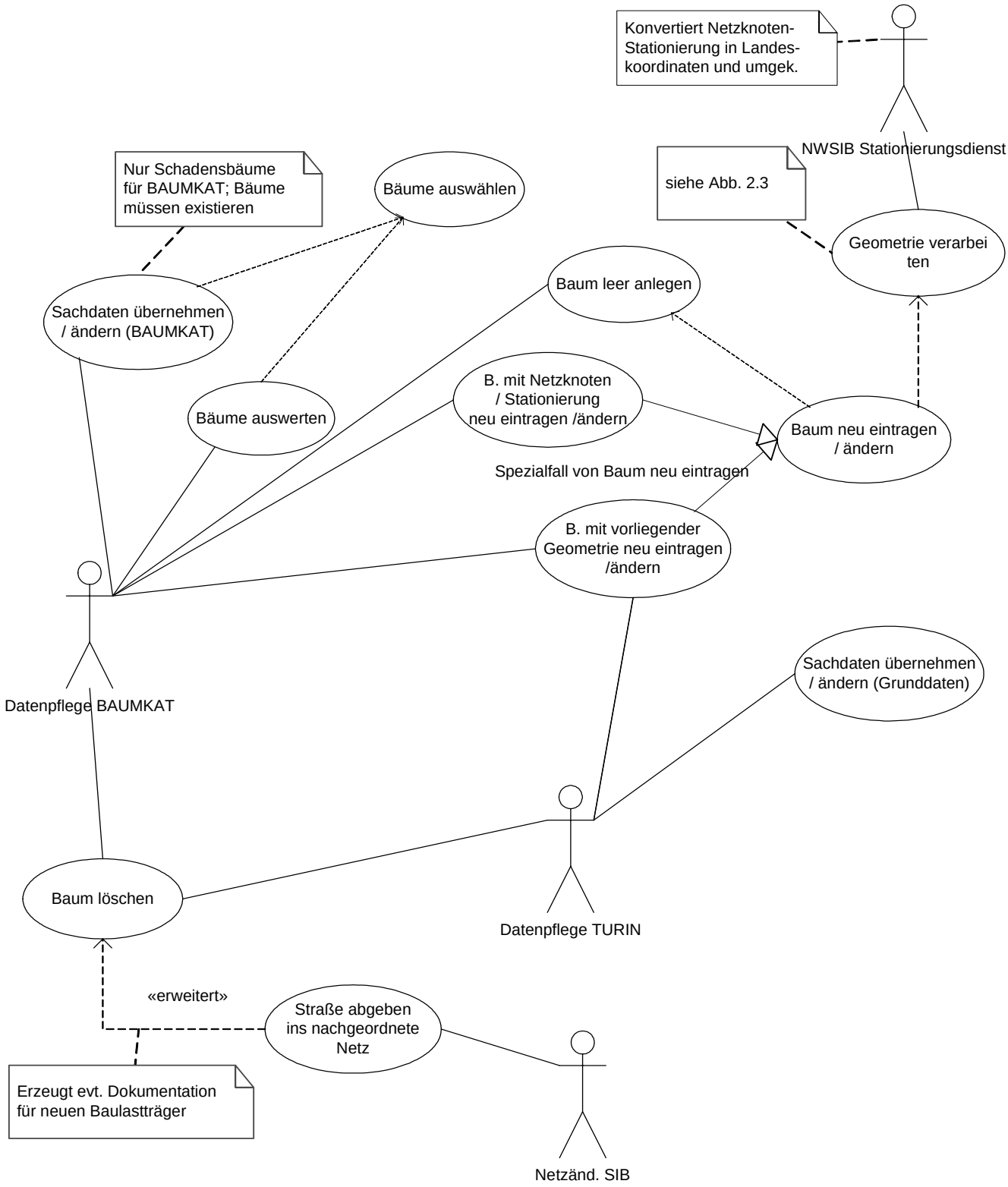


Abbildung 35. Anwendungsfälle Baumobjekte.

Als nächstes wurden für folgende Anwendungsfälle Szenarien aufgestellt.

Baum löschen

- Datenpflege BAUMKAT löscht Baum mit Hilfe der stationären BAUMKAT-Anwendung. (Funktion im BAUMKAT-Konzept nicht aufgeführt.)
- Datenpflege TURIN löscht Baum mit Hilfe der TURIN-Anwendung
- Netzänd. SIB löscht Baum über Datenbank-Funktion

Sachdaten übernehmen / ändern (Grunddaten)

- Datenpflege TURIN trägt die neuen oder geänderten Sachdaten zum Baum mit Hilfe der TURIN-Anwendung ein.

Sachdaten übernehmen / ändern (BAUMKAT)

- Datenpflege BAUMKAT sucht Baum oder legt neuen Baum an
- Datenpflege BAUMKAT trägt die neuen oder geänderten Sachdaten zum Baum mit Hilfe der stationären BAUMKAT-Anwendung ein. (Funktion *Eingabe und Ändern eines Baumes über Masken* in BAUMKAT-CP/EP)

Baum mit Netzknoten-Stationierung neu eintragen/ändern

Dieser Anwendungsfall wird durch das Aktivitätsdiagramm 2-6 wiedergegeben.

Weitere Anwendungsfälle

Die verbleibenden Anwendungsfälle könnten nun ebenso behandelt werden. Auf Grund der Gemeinsamkeiten mit den Verhältnissen bei der Fortführung anderer Objekte sollen die detaillierten Aktivitätsdiagramme jedoch später im Rahmen einer Analyse des Lebenslaufes eines allgemeinen Fachobjektes aufgestellt werden.

B.2.2.4 Fortführung von weiteren Objekten

Die bei den Leitungen und den Bäumen vorgefundenen Verhältnisse bezüglich der Anwendungsfälle sind so typisch, dass sie sich für eine ganze Reihe weiterer Objekte wiederholen. Bei einem ersten Überblick kommen nach Auskunft der befragten Fachleute als weitere Kandidaten in Frage:

- Durchlässe
- Schilder (wegweisend oder verkehrsregelnd)
- Lichtsignalanlagen
- Zählstellen
- Hindernisse
- Schutz- und Leiteinrichtungen
- Lärmschutz
- Stationszeichen
- Bestimmte Arten von Kreuzungen

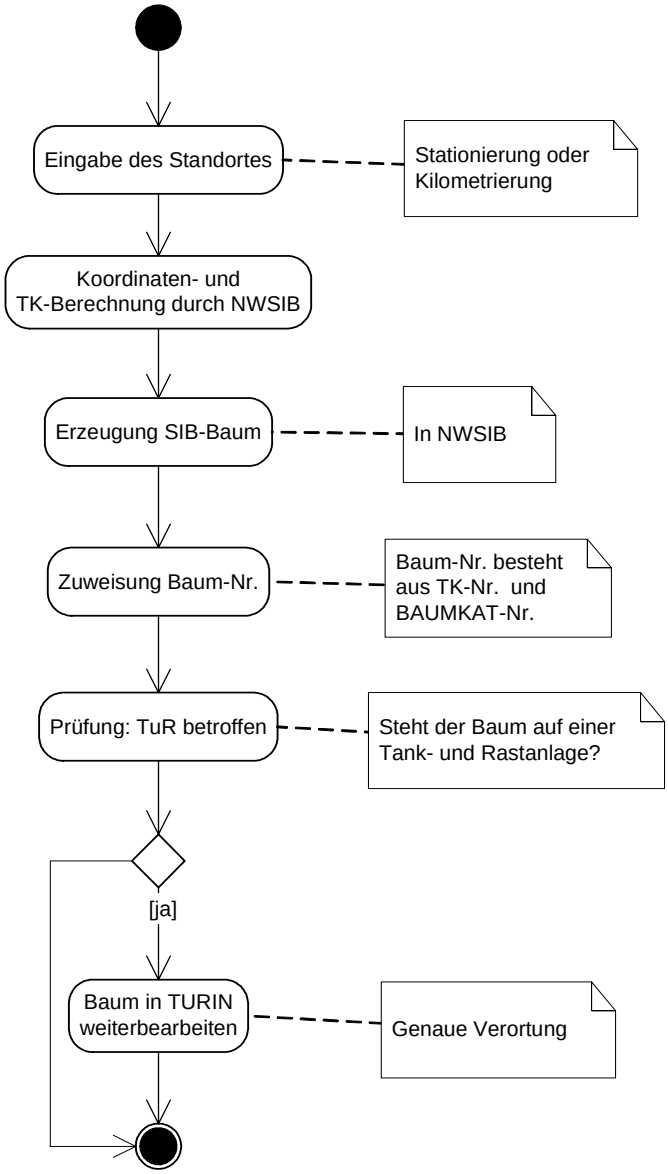


Abbildung 36. Baum neu eintragen/ändern.

B.2.2.5 Ergebnisse der Prozessanalyse

Die Prozessanalyse ergibt folgendes Bild:

- Es gibt Fachinformationssysteme, die für die objekt-spezifischen Aufgaben zuständig sind, z.B. für die Kontrolle geschädigter Bäume oder für die Dokumentation von Leitungsverläufen. Das zu Grunde liegende Eigenschaftsprofil – Attribute und Assoziationen – ist an die anstehenden Fachaufgaben angepasst.
- Daneben tauchen Fachinformationssysteme wie TURIN auf, die Objekte verwalten und pflegen, die sehr komplex sind und die Objekte wie Leitungen und Bäume (u.v.a.m.) „mitbetreuen“, sofern sie als Bestandteile auftauchen.

- Der Netzbezug der Objekte wird wiederum in der Straßeninformationsbank (SIB) gepflegt. Das hinter der SIB stehende Objektmodell der ASB ist darauf zugeschnitten, es werden nur wenige Attribute geführt. Von den Leistungen der Straßeninformationsbank werden typischerweise genutzt:
  - Vergabe von Objektbezeichnern abgeleitet aus TK-Nummern oder Unterstützung dazu
  - Umrechnung von Stationierungsangaben in Landeskoordinaten und zurück

  Die aus der TK-Nummer abgeleiteten Objektbezeichner sind nicht stabil. Bei Korrektur der Position des Objektes kann es notwendig werden, den Bezeichner zu ändern.
- Eine weitere typische Situation ist, dass Objekte bei Abstufung eines Abschnittes aus dem aktuellen Bestand der SIB herausfallen (und dann dort nur noch historisch vorhanden sind). Die Fachinformationssysteme müssen über solche Änderungen informiert werden, wenn solche Objekte an Geschäftsprozessen nicht mehr teilnehmen sollen (z.B. wenn Bäume nicht mehr kontrolliert werden müssen).
- Die Fortführungsmechanismen sind für viele Objektarten sehr ähnlich.
- Eine Objektart wird von mehreren Fachinformationsystemen bearbeitet. Der Gesamtbestand der Objekte kann auf mehrere Fachinformationssysteme aufgeteilt sein. Ein bestimmtes Exemplar kann in mehreren Fachinformationssystemen geführt werden (Schadbäume auf TuR-Anlagen). Die Zahl der Fachinformationssysteme pro Objektart, in denen Objekte parallel fortzuführen sind, ist durchweg gering.
- Der Anstoß zu einer Fortführung geht von einem Fachinformationssystem aus, die anderen fügen den neuen oder geänderten Objekten Information hinzu oder müssen über Löschungen auf dem laufenden gehalten werden. Dabei kann das initiierende Fachinformationssystem von Fall zu Fall ein anderes sein. Es gibt verschiedene Fortführungs-Szenarien, je nach Initiator.

Diese Ergebnisse lege es nahe, statt der konkreten Baumobjekte usw. generalisierte Fachobjekte zu betrachten

Die in so einem Rahmenwerk zu betrachtenden Anwendungsfälle sind:

- Fachobjekt fortführen, die für unterschiedliche Sichten auf die Fachobjekte zuständig sind, mit den Spezialisierungen:
  - Fachobjekt neu erzeugen
  - Fachobjekt ändern
  - Fachobjekt löschen
  - Fachobjekte teilen (z.B. bei Leitungen)
  - Fachobjekte vereinigen (z.B. bei Leitungen)
- Arbeitsablauf für einen konkretes Szenario der Fortführung festlegen
- Arbeitsablauf für ein konkretes Szenario überwachen
- Abfrage nach Fachobjekt(en) mit Hilfe mehrerer Fachinformationssysteme durchführen (z.B. für Auswertungen)

Als Diagramm:

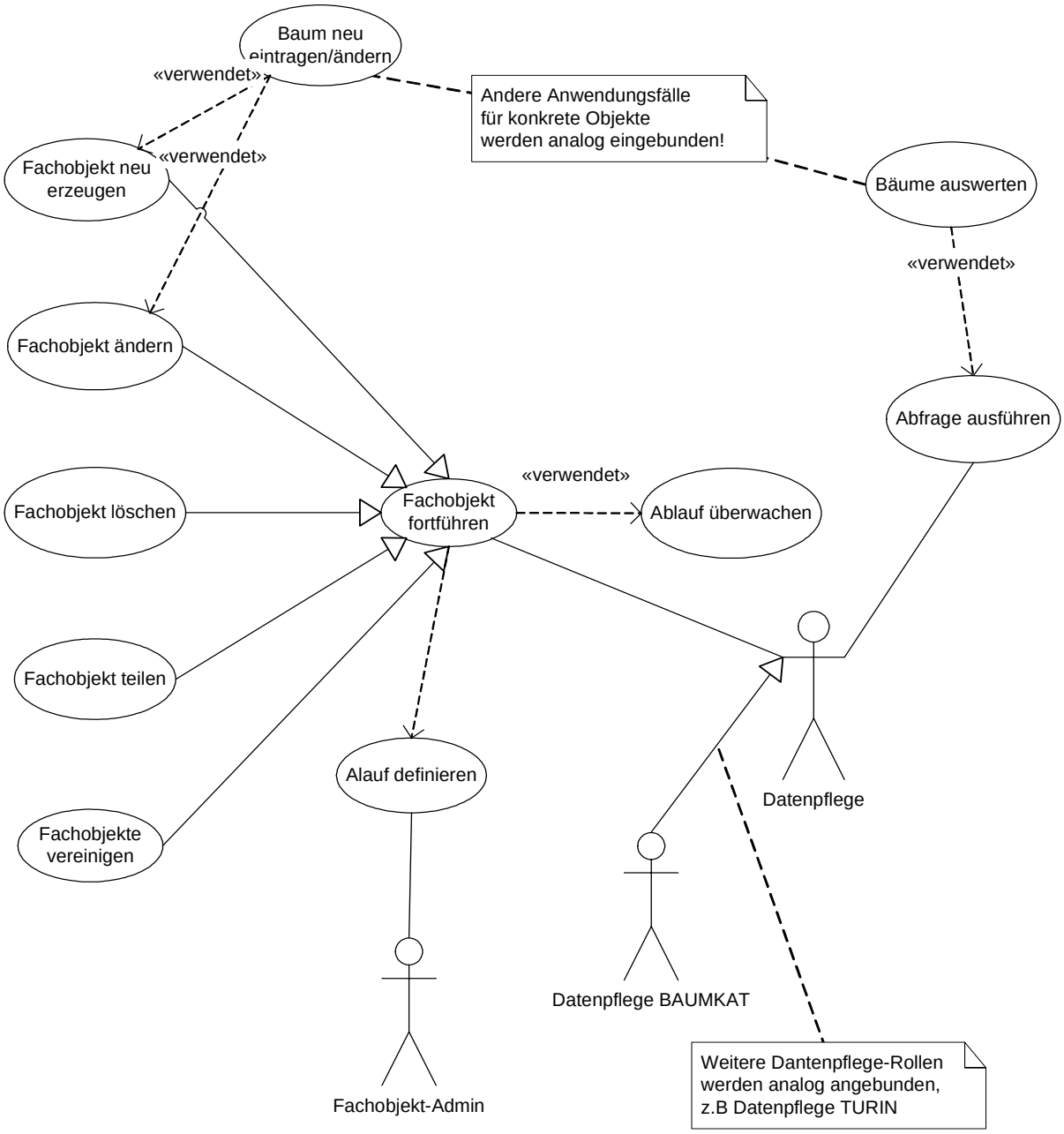


Abbildung 37. Verallgemeinerte Anwendungsfälle und Akteure.

Zur Erläuterung: Der Zusammenhang mit den vorher diskutierten konkreten Anwendungsfällen und Akteuren ist exemplarisch für Datenpflege TURIN und Bäume neu eintragen/ändern durchgeführt. Andere Anwendungsfälle und Akteure sind analog anzubinden.

Spezialisierte Anwendungsfälle erben die Akteure des Basisanwendungsfalles. Die konkreten Anwendungsfälle sind per *verwendet* an die Fachobjekt-Anwendungsfälle gebunden, um die strengeren Einschränkungen der Generalisierung (vollständige Austauschbarkeit der Generalisierung durch die Spezialisierung) zu umgehen.

B.2.3 Bildung der Objektwelt

Für die Bildung der Objektwelt zur Modellierung von Fortführungsprozessen im Rahmen des Forschungsvorhabens sind nach den Erkenntnissen der Zusammenfassung 2.2.5 weniger die fortzuführenden konkreten Fachobjekte wie *Baum* usw. interessant, für die es z.T. auch schon stabile und detaillierte Modelle gibt, als Objekte zur *Koordination des Fortführungsprozesses über mehrere Fachinformationssysteme* hinweg. Die bei der Fortführung anfallenden Aufgaben sind nämlich für viele Objektarten (d.h. Fachobjektklassen) gleich oder sehr ähnlich, wie die Analyse der Anwendungsfälle, Abläufe und Fachinformationssysteme oben zeigt.

Es werden daher die für den Anwendungsfall *Fachobjekt fortführen* zentrale Klasse Fachobjekt und die Klassen, die für Ablaufkontrolle (Anwendungsfall *Ablauf überwachen*) erforderlich sind.

Um den Kontakt zu den konkreten Fachobjekten nicht ganz zu vernachlässigen, folgen zunächst drei Beispiele für Fachobjekte.

B.2.3.1 Beispiele für Fachobjekte

B.2.3.1.1 Bäume

Für Baumobjekte liegt im OKSTRA eine fertige und stabile Modellierung als Punktobjekt vor.

B.2.3.1.2 Leitungen

Bei Betrachtung der Anwendungsfälle für die Leitungen tauchen als Objekte die *Leitungen* sowie die *Nutzungsverträge* auf. Die Beispieldokumente für Nutzungsverträge sowie der Signaturenkatalog des Dez. 26 des NLStB zeigen jedoch, dass es neben den Leitungen zahlreiche zusätzliche Objekte gibt, die zu berücksichtigen sind, z.B. Pumpen, Schächte, Masten etc. Hier ist also zunächst ein fachliches Modell im OKSTRA für alle benötigten realen Objekte notwendig.

Vielfach werden zur Fortführung mehrere Leitungen bzw. Zusatzeinrichtungen angeliefert. Die übergebenen Pläne z.B. enthalten oft nicht nur die Lage *einer* Leitungsführung, sondern mehrerer, und außerdem auch die Verortung aller damit zusammenhängenden Hilfs- und Zusatzeinrichtungen. Genauso gehen bei Wechsel eines Betreibers nicht einzelne Leitungen, sondern ganze Leitungsnetze samt der verbundenen Einrichtungen an den neuen Betreiber über. Beim Einbau neuer Einrichtungen müssen zudem oft vorhandene Leitungen verändert werden.

Es ist daher angebracht, die Anwendungsfälle nicht auf *einzelne* Leitungen zu beziehen, sondern auf *Leitungssysteme* als Aggregate von Leitungswegen und Zusatzeinrichtungen. Das legt für unsere Objektwelt nahe, solche Leitungssysteme als Mengen abstrakter Objekte eines Typs *Leitungssystemelement* zu betrachten. Leitungssystemelemente können sein: *Leitungen*, die sich in ein oder mehrere *Leitungsabschnitte* gliedern können, und *Leitungsabschnittbegrenzungen* (Schächte, Verzweigungen usw.). Diese können sowohl allein als auch an den Enden von Leitungsabschnitten vorhanden sein. Die Arten von Leitungen und Leitungsabschnittsbegrenzungen werden als Attribute mitgeführt. Ein exemplarisches und rudimentäres Klassendiagramm für solche Leitungssysteme folgt als Abb.

38. Die komplette und detaillierte Modellierung im Rahmen des bestehenden OKSTRA steht noch aus.

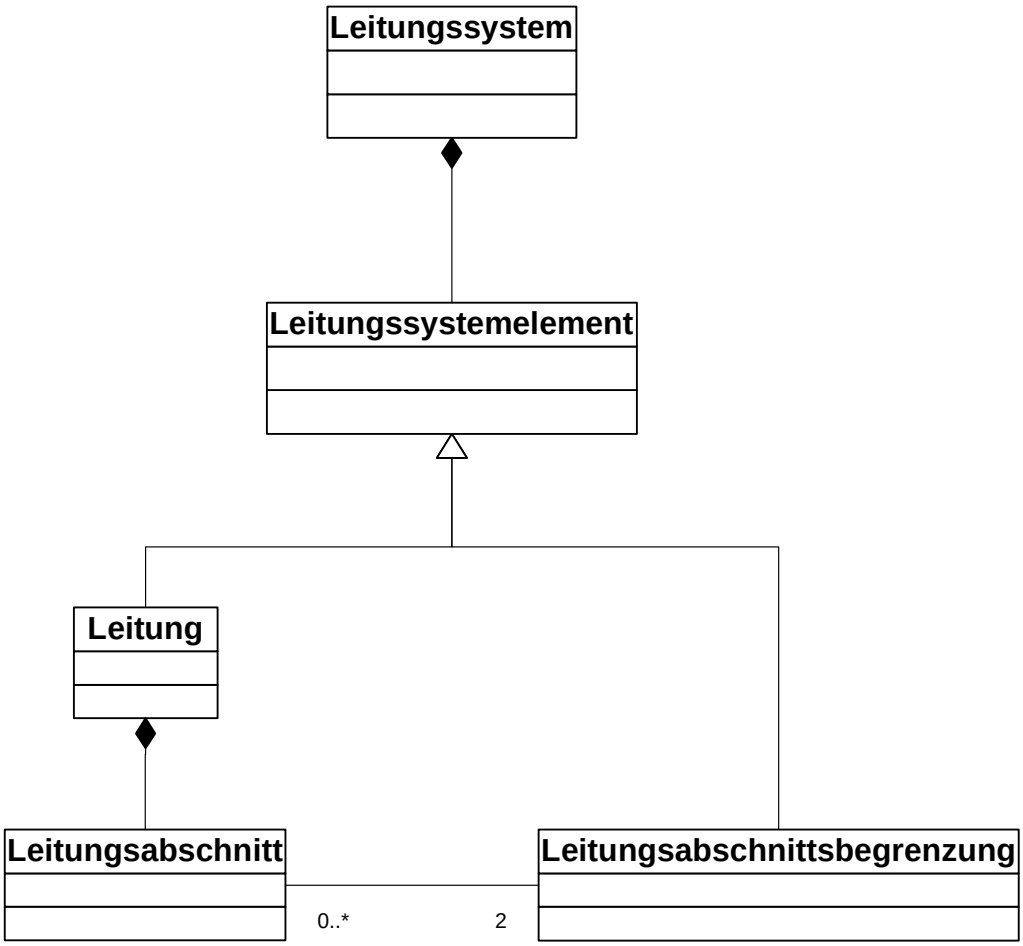


Abbildung 38. Leitungen.

B.2.3.1.3 Tank- und Rastanlagen

Tank- und Rastanlagen werden in Niedersachsen über ein Fachinformationssystem TURIN verwaltet. Sie haben eine komplexe innere Struktur (die zz. noch nicht im OKSTRA modelliert ist). Diese wird in der Sicht des Straßennetzes gemäß der ASB innerhalb der SIB nicht abgebildet. Dort sind Tank- und Rastanlagen als Objekte ohne detaillierte Struktur wiedergegeben.

B.2.3.1.4 Folgerungen

Die drei Beispiele weisen eine sehr unterschiedliche Komplexität auf. Am Beispiel der Tank- und Rastanlagen zeigt sich, dass dort Objekte bestimmter Klassen (z.B. Bäume oder Beleuchtung) als *Bestandteile* fortgeführt werden, deren Lebensdauer durch die des enthaltenden Objektes (nämlich der TuR-Anlage) begrenzt wird.

Der Idealfall wäre ein Verbund von Fachinformationssystemen mit Pflegeverantwortlichkeiten, die sich nicht überschneiden. Es gäbe also ein Baumkataster, das Bäume und Baumreihen exklusiv verwalten würde, ein oder mehrere Leitungskataster für die Leitungssysteme usw. Andere Systeme (z.B. ein hypothetisches neues TURIN) würden im Falle, dass z.B. ein Baum verändert werden müsste, diese Aufgabe an das Baumkataster delegieren und ansonsten auf die dort abgelegte geografische und fachliche Information nur lesend zu Darstellungszwecken zugreifen.

Dieser Idealfall ist aber praktisch nie verwirklicht. Das liegt daran, dass normalerweise Anwendungen für die Subobjekte (z.B. Bäume, Beschilderung, Schutz- und Leiteinrichtungen) gar nicht existieren und daher in begrenztem Umfang mitkonstruiert werden mussten. Ein Umbau dieser Anwendungen, die z.T. ja gerade erst in den Produktionseinsatz gelangen, scheidet i.d.R. aus wirtschaftlichen Gründen aus, auch wenn auf längere Frist die Idealkonstruktion „ein System für eine Aufgabe“ technische und wirtschaftliche Vorteile bieten würde.

Es ergeben sich daraus folgende Anforderungen:

- Die Funktionalität bestehender Komponenten sollte weitestgehend erhalten bleiben. Das bedeutet, dass z.B. TURIN weiterhin Baumobjekte selbst pflegen können soll.
- Neue Anwendungen sollten jedoch nur noch die Objekte in ihrer eigenen Verantwortlichkeit pflegen dürfen und für Bestandteile wie Bäume etc. auf einen entsprechenden Dienst zurückgreifen. Damit wird verhindert, dass „das Rad immer wieder neu erfunden wird“. und unnötiger Harmonisierungsaufwand erzeugt wird.

B.2.3.2 Objekte für die Fortführung

Aus den Ergebnissen der Analyse der Prozesse ergeben sich zunächst folgende Aussagen über den Ist-Zustand, die Aufschluss über die benötigten Objektklassen geben (die Liste ist nicht geordnet). Fettgedruckt sind dabei die Begriffe, die auf benötigte Objekte bzw. Eigenschaften hinweisen.

B.2.3.2.1 Informelle Beschreibung der Objektwelt

- Der Bestand der fortzuführenden ***Fachobjekte*** kann auf ***Teilbestände*** in mehreren ***Fachinformationsystemen*** (FIS) verteilt sein (Beispiel Bäume: TURIN enthält alle Bäume in TuR-Anlagen; BAUMKAT verwaltet alle Schadbäume). Wie das Beispiel zeigt, kann ein Objekt in mehreren Fachinformationssystemen vorkommen. (Schadbäume in TuR-Anlagen!)
- FIS sind spezielle ***Fachanwendungen***, die über Persistenzdienstleistungen („Datenbanken“) verfügen. Fachanwendungen brauchen nicht ständig ***aktiv*** zu sein. Fachanwendungen müssen von FIS über ***Fortführungsereignisse*** benachrichtigt werden können (siehe die Erläuterungen zu den Kommunikationsverbindungen in Abschnit 2.1.4).
- Ein FIS ist auf einem ***Server*** installiert. Da sich Serveradressen ändern können, ist ein Verzeichnis wünschenswert, dass die Zuordnung registriert.
- Unterschiedliche FIS können unterschiedliche ***Sichten*** auf die Objekte repräsentieren, d.h. ein FIS verwaltet u.U. nur einen Teil der Eigenschaften der Objektklasse. (Die NWSIB verwaltet z.B. nur den Netzbezug der Bäume). Dieses FIS ist dann allein nicht

in der Lage, ein vollständiges OKSTRA-konformes Objekt zur Weiterverarbeitung zu bilden.

- Zur Fortführung werden die Objekte von einem oder mehreren FIS bearbeitet. Die Reihenfolge der Bearbeitung ist i.A. nicht festgelegt (ein Baum kann z.B. zuerst in TURIN angelegt werden und wird im „Schadenfall“ dann in BAUMKAT weiterverarbeitet, oder aber bei einer Baumkontrolle wird ein Schadbaum auf einer TuR-Anlage entdeckt, dann wird der Baum zuerst in BAUMKAT angelegt und später in TURIN weiterverarbeitet).
- Bei der Bearbeitung durch mehrere FIS können ***Konflikte*** auftauchen, sowohl zwischen den Daten verschiedener Sichten desselben Objekts als auch in Bezug auf andere Objekte (z.B. kann der Standort für einen neu erfassten Baum in einem der FIS bereits durch ein anders Objekt besetzt sein). Zur Konfliktauflösung ist i.d.R. eine genauere manuelle Untersuchung erforderlich.
- Eine Fortführung kann ***mehrere*** Objekte zugleich betreffen. (Z.B. Löschen eines Baumes aus einer Baumreihe und Aufteilung der Baumreihe in zwei)

B.2.3.2.2 Schema für den Lebenslauf eines Fachobjekts

Zur weiteren Untersuchung wurde als nächstes ein ***Fortführungs-Zustandsmodell*** aufgebaut. Für die Oberklasse *Fachobjekt*, die die betrachteten Fachobjekte umfasst (Bäume, Leitungen, siehe oben) wurde von der Projektgruppe ein Lebenslauf-Modell entwickelt, das im Folgenden in Form mehrerer Aktivitätsdiagramme dargestellt ist. Die Lebenszustände, die das Objekt annehmen kann, sind dort eingetragen. Die Aktivitätsdiagramme dokumentieren gleichzeitig die Struktur der oben eingeführten Anwendungsfälle zur Fortführung von Fachobjekten.

Hierzu einige wichtige Anmerkungen:

- Die Dauer eines Fortführungsvorgangs ist nicht vorhersagbar. Infolgedessen muss ein Objekt, während es fortgeführt wird, in seinem Originalzustand sichtbar bleiben. Alle Fortführungen müssen daher zuerst auf Kopien erfolgen, die wir ***Klone*** genannt haben. Am Ende der Fortführung muss die Information, die sich in den Klonen gesammelt hat, in die Objekte übertragen werden. (Siehe ergänzend auch das Fortführungsmodell der NWSIB).
- Objekte sollen nie endgültig vernichtet werden. Sie werden stattdessen ***eingefroren*** und sind in diesem Zustand nicht mehr im aktuellen Bestand sichtbar, falls historisiert wird, jedoch in historischen Beständen. Eingefrorene Objekte können ggf. wieder ***auftgetaut*** werden. Ob dies möglich ist, hängt von der Fachobjektklasse und/oder dem Objektinhalt (den Daten des Exemplars) ab. Das Modell lässt dies zu, macht aber über die Bedingungen zur Auftaubarkeit keine Aussage. (Ausnahme: Klone werden vernichtet, sobald die Fortführung beendet ist.)
- Ob und, wenn ja, wie eine Historisierung erfolgt, legt das Modell nicht fest.
- Die einzelnen Objektsichten werden nacheinander erfasst. Der Erfassungsvorgang für die einzelne Sicht wird als atomar vorausgesetzt, d.h. eine Sicht wird entweder komplett oder gar nicht bearbeitet. Das schließt z.B. Szenarien aus, die zuerst eine Sicht A teilweise bearbeiten, dann eine Sicht B, und dann die Bearbeitung von Sicht A wieder aufnehmen.

- Die Objekte besitzen eine durch Koordinaten gegebene Objektlage und –geometrie sowie einen Bezug auf das Straßennetz.

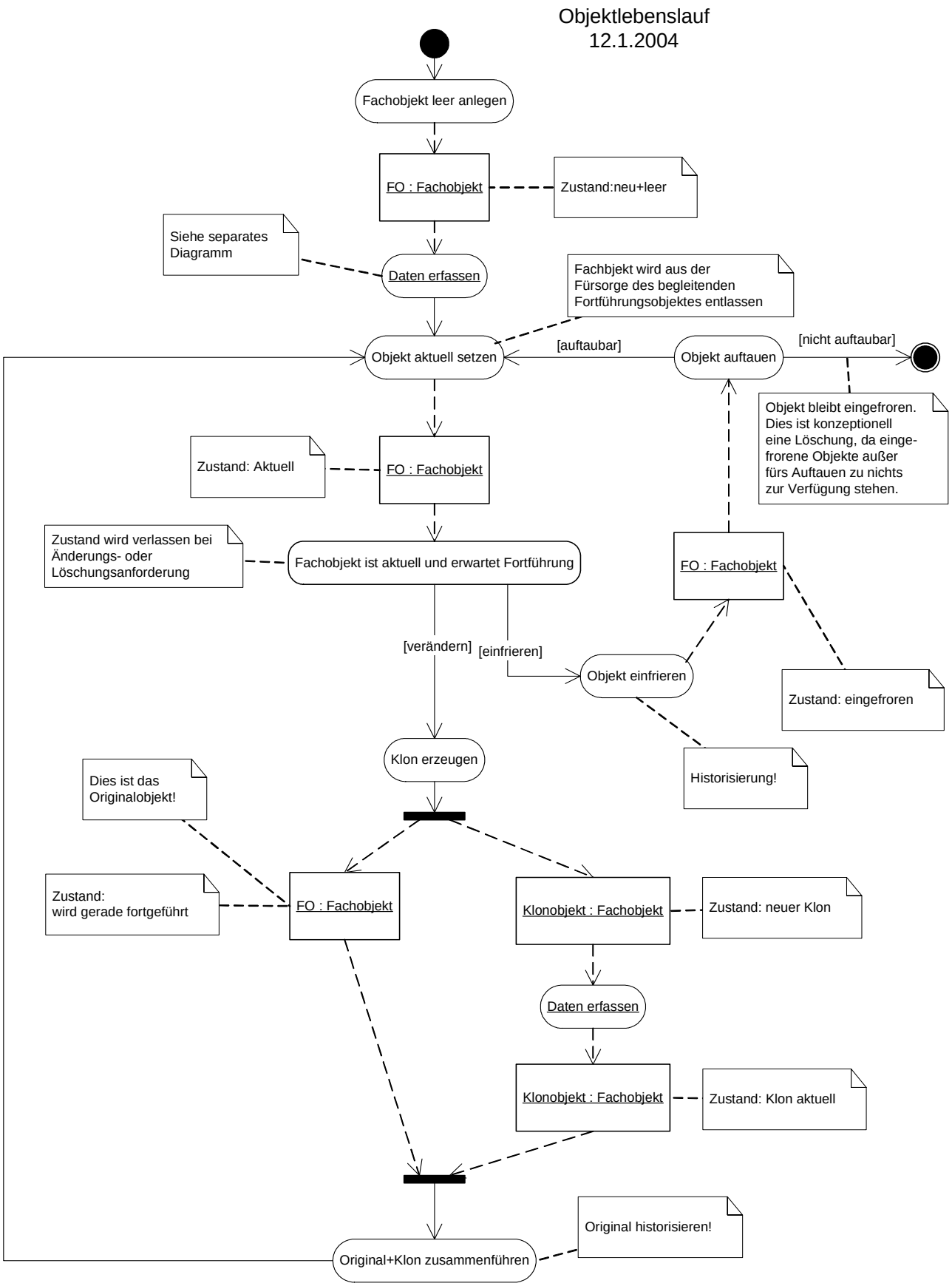


Abbildung 39. Objektlebenslauf.

Daten erfassen
12.1.2004

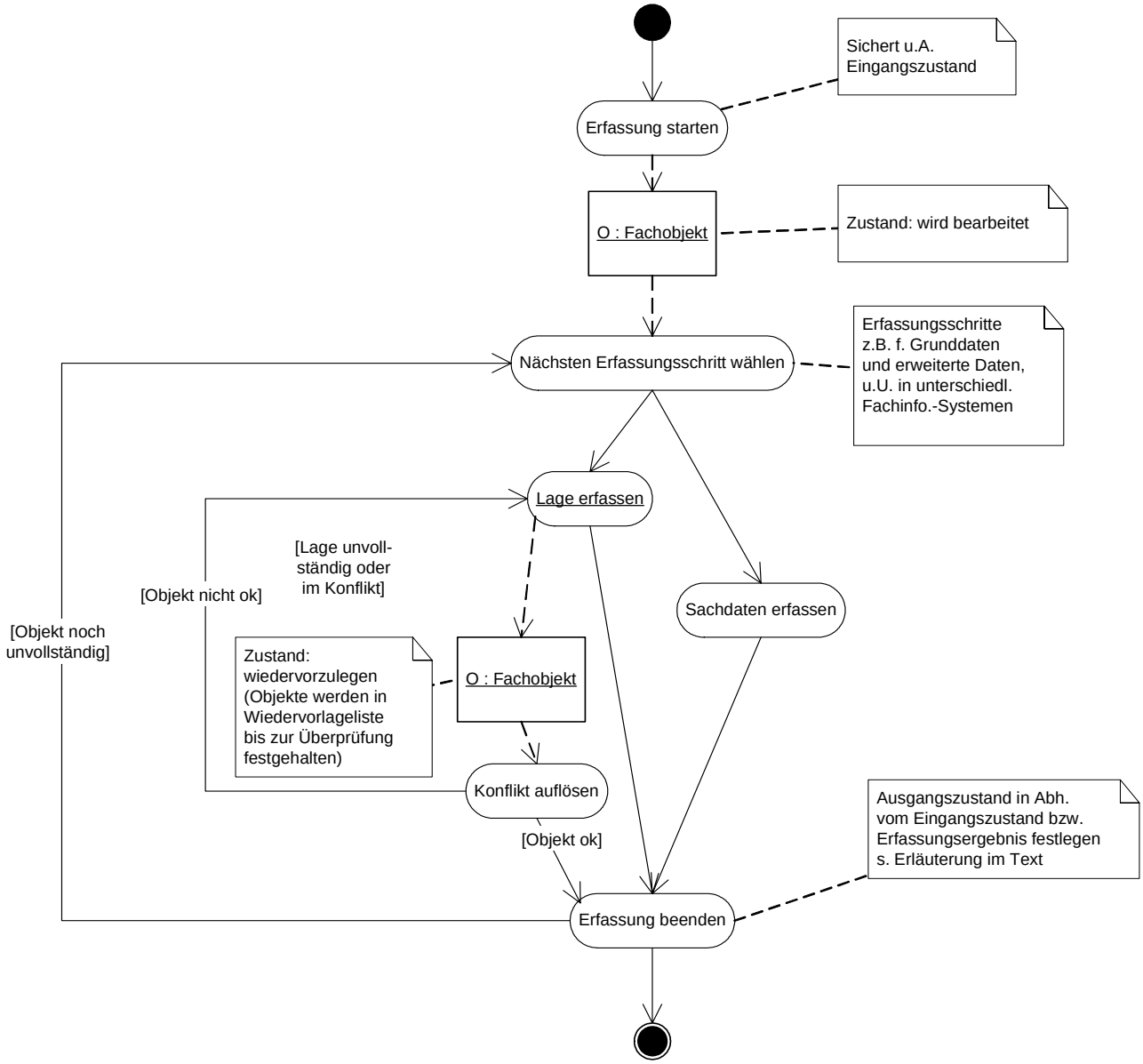


Abbildung 40. Datenerfassung.

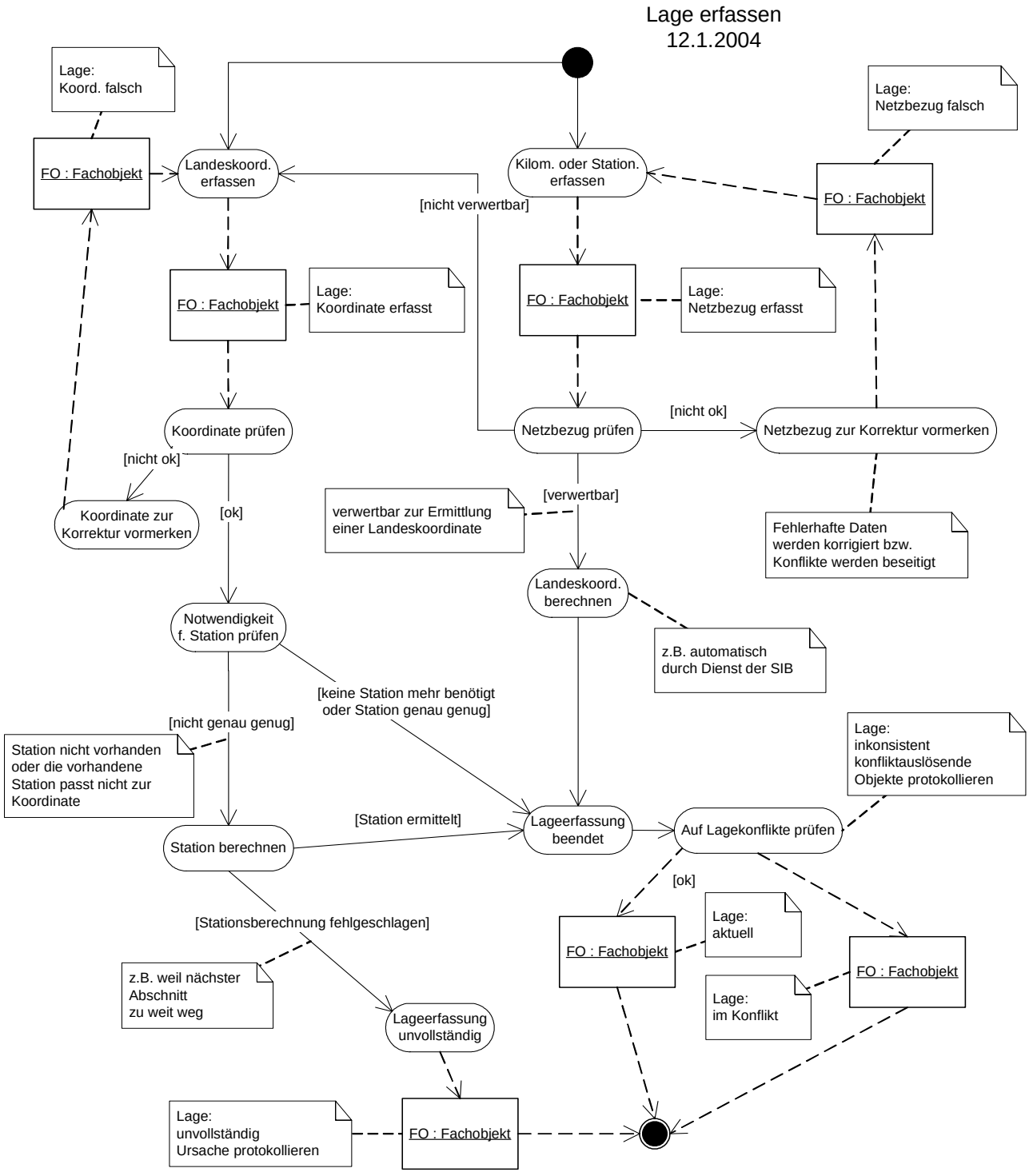


Abbildung 41. Lageinformation.

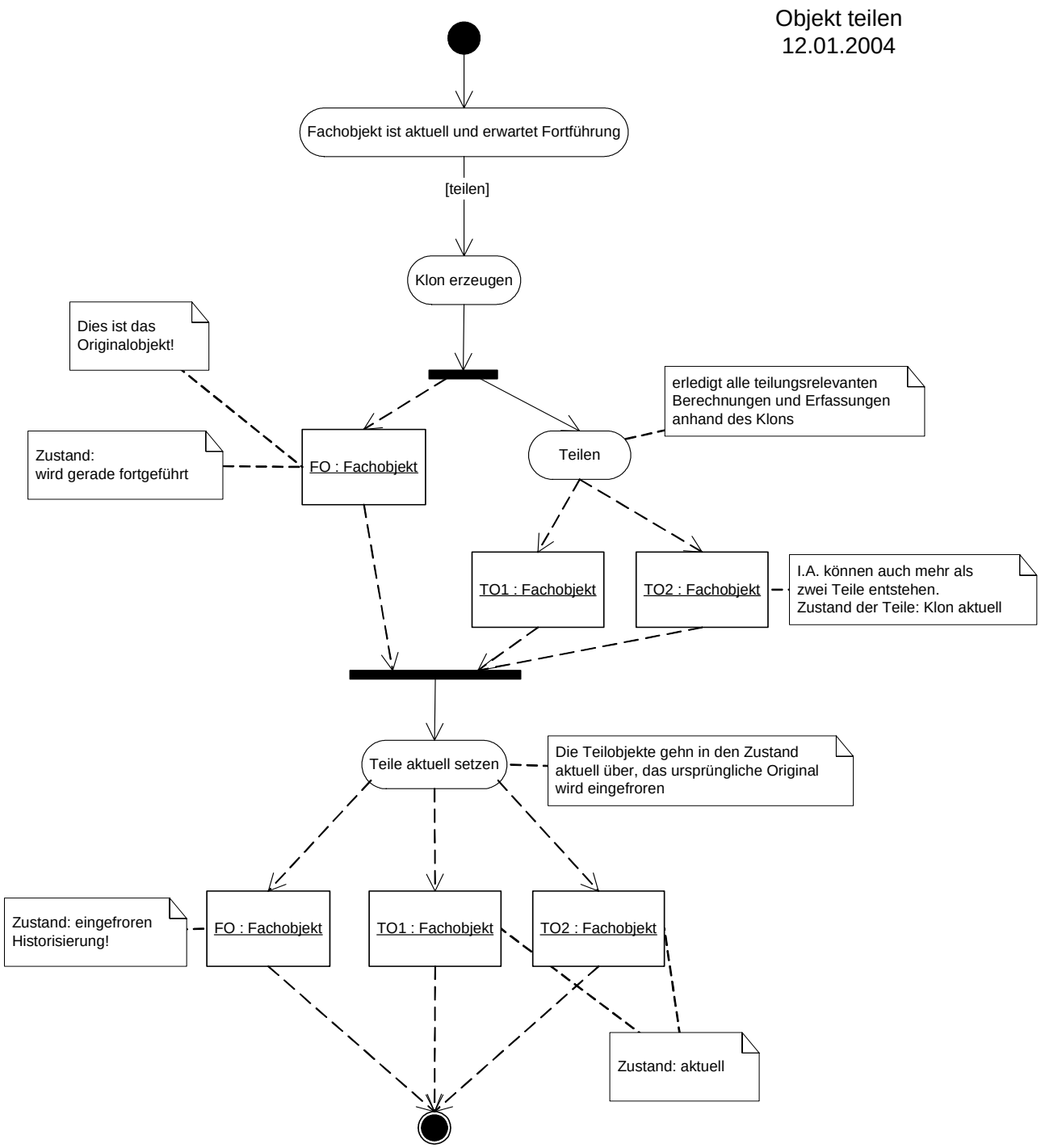


Abbildung 42. Fachobjekt teilen.

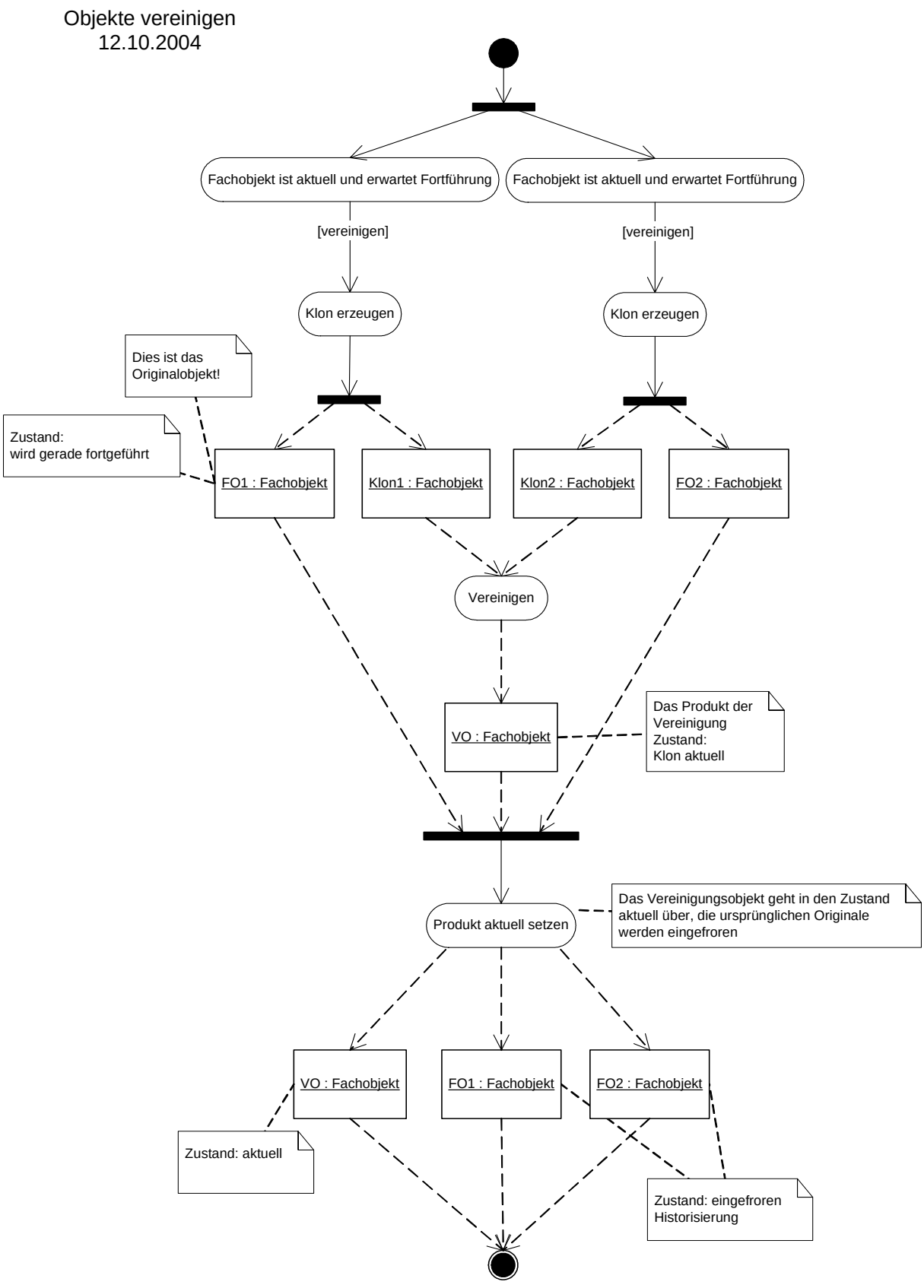


Abbildung 43. Fachobjekte vereinigen.

B.2.3.2.3 Anforderungen an das Fortführungsmodell

Aus den bisherigen Ausführungen ergeben sich folgende Anforderungen:

- Für den Anwendungsfall *Abfrage ausführen* ist eine Integration sowohl der Sichten als auch der Teilbestände nötig. Die Objekte sollten ihnen über OKSTRA-gerechte Dienste zur Verfügung stehen (Eine Anwendung, die Informationen zu Bäumen verarbeitet, z.B. eine thematische Kartierung, sollte „OKSTRA-Bäume“ verwenden.) Für Recherchen (Abfragen, Queries) wird deshalb ein ***Abfragedienst*** benötigt, der aus allen beteiligten FIS die Information für eine bestimmte Objektinstanz sammelt und zu einer gültigen OKSTRA-Repräsentation zusammenbaut, die von den Anwendungen genutzt werden kann. Um Abfragen frei definieren zu können, müssen diese in Sätzen einer Abfragesprache formulierbar sein; die Sätze werden in ***Abfrageobjekten*** aufgehoben.
- Ein am Fortführungsprozess teilnehmendes FIS wird durch ein Objekt der Klasse ***Fachinformationssystem*** mit folgenden Eigenschaften modelliert:
  - Es hat einen Namen und eine Version.
  - Es hat eine Serviceaddresse (z.B. eine URL, wenn es als Web-Service erreichbar ist.)
  - Es ist für eine bestimmte Sicht auf ein Objekt zuständig, die durch ein Teilschema des vollständigen Schemas der entsprechenden Objektklasse gegeben ist.

Aus der Betrachtung der speziellen Anwendungsfälle zu *Fachobjekt fortführen* ergibt sich:

- Ein FIS muss wenigstens die bekannten Operationen zur Herstellung von Persistenz, in diesem Modell ***Erzeugen, Einfrieren, Ändern*** analog zu ***Insert, Delete, Update*** unterstützen, wie sie in RDBMS oder auch in der OGC-WFS Schnittstelle gefordert werden. Daneben muss es die oben verlangte ***Auftau***-Operation geben. Die Operationen werden durch die Akteure, die die Fortführung durchführen, angesprochen; sie müssen nicht zwingend als Schnittstelle eines Dienstes angeboten werden, der von anderen Anwendungen genutzt werden kann, sondern können z.B. auch rein interaktiv implementiert sein, wenn nur menschliche Akteure bedient werden müssen. Operationen zum ***Teilen*** und ***Vereinigen*** können hinzutreten.
- Eine verteilte Fortführung erfordert zusätzliche Überlegungen in Bezug auf die Vollständigkeit und Konsistenz der Objekteigenschaften:
  - Werden Objekte neu erzeugt, sind sie möglicherweise nicht für alle beteiligten FIS interessant (z.B. Schadbäume, die nicht auf TuR-Anlagen stehen). Andererseits könnten Objekte, die zunächst nicht für ein FIS von Interesse waren, es durch eine Änderung werden.
  - Möglicherweise entstehen während der Fortführung Konflikte zwischen den in den FIS gepflegten Objektsichten, d.h. Eigenschaften, die verschiedene FIS beisteuern, sind unverträglich, oder es werden Genauigkeitsanforderungen einzelner FIS verletzt.

  Solche Situationen können durch eine ***Voruntersuchung*** oder ***Prüfung*** aufgefangen werden – zwischen bestandener Prüfung und endgültiger Fortführung darf an dem Objekt dann nichts mehr geändert werden. Bei nicht bestandener Prüfung sind zwei Ausgänge denkbar:

- Eine Fortführung wird abgelehnt.
- Die Erzeugung oder Änderung wird soweit wie möglich durchgeführt, das Objekt wird aber als ***zur Korrektur*** oder auch, bei Unvollständigkeit, ***zur Ergänzung*** (mit Termin) gekennzeichnet. Voraussetzung hierfür ist, dass der erreichte Zustand wenigstens in sich und gegenüber anderen Objekten konsistent ist, d.h. das Objekt muss verwendbar sein, wenn auch mit der Einschränkung, dass bestimmte Informationen nicht oder ungenau vorliegen und daher bestimmte Operationen nur eingeschränkt mit den Objekten durchführbar sind.

  In beiden Fällen werden Daten, die für die Konfliktbeseitigung wichtig sind, protokolliert.

  Das oben geschilderte Klonierungsverfahren erlaubt eine einfache Durchführung solcher Prüfungen: Die Fortführungsoperationen werden über alle beteiligten FIS hinweg zunächst auf die Klone losgelassen. Verlangt ein FIS die Ablehnung der Fortführung, können die Klone beseitigt und der Originalzustand wiederhergestellt werden. Im Falle des Gelingens, auch mit dem Vorbehalt der Korrektur bzw. Ergänzung, steht am Ende der Fortführung ein aktualisierter Satz von Klonobjekten zur Verfügung, die dann noch in den Bestand übernommen werden müssen. Es wird folglich in der Schnittstelle der Fachinformationsssystem-Objekte eine Operation zur Übernahme der Klonobjekte gebraucht.

- Da manche Fortführungsszenarien mehrere voneinander abhängige Objekte zusammen bearbeiten, brauchen wir Objekte, die solche Objektgruppen zusammenhalten.

B.2.3.2.4 Zustandskontrolle der Fachobjekte

Während der Fortführung ist mit einem Fachobjekt offenbar zuätzlicher Kontext verbunden. So muss etwa die oben beschriebene Klonierung überwacht werden, der Zustand der Lageermittlung und es muss festgehalten werden, welches FIS das Objekt gerade bearbeitet. Da aktuelle Objekte diese Information nicht benötigen, ist es zweckmäßig, allgemeine ***Fortführungsobjekte*** einzuführen, die nur während der Fortführung leben.

Ein Fortführungsobjekt muss offenbar dann erzeugt werden, sobald

- ein neues Fachobjekt benötigt wird,
- ein Fachobjekt die Aktion „ist aktuell und erwartet Fortführung“, getrieben durch ein äußeres Ereignis (d.h. ein Akteur wird tätig!), verlässt;
- ein eingefrorenes Fachobjekt aufgetaut werden soll.

In den letzten beiden Fällen wird das Fachobjekt mit dem neuen Fortführungsobjekt verknüpft und geht in den Zustand „wird gerade fortgeführt“ über. Die anderen beiden Zustände für Fachobjekte sind „aktuell“ und „eingefroren“.

Die Klasse Fachobjekt bekommt folglich ein Attribut ***Zustand*** vom Datentyp ***Zustandstyp***, einer Aufzählung, die die drei genannten Werte umfasst, sowie eine 0..1-Assoziation zur Klasse ***Fortführungsobjekt***.

Als nächstes werden die Klonierungsregeln betrachtet:

- Geht ein Fachobjekt vom Zustand „aktuell“ in den Zustand „wird gerade fortgeführt“ über, wird ein Klon angelegt.

- Ein neues Fachobjekt darf erst dann für externe Anwendungen sichtbar werden, wenn es vollständig aufgebaut ist. Als erster Schritt wir daher ein leerer Klon angelegt; der in den Aktivitätsdiagrammen auftauchende Zustand „neu+leer“ besteht also darin, dass das Fortführungobjekt nur ein Klonobjekt ohne Original aufweist.
- Beim Einfrieren und Auftauen gibt es zwei Möglichkeiten: Kann der Zustand unmittelbar zwischen „eingefroren“ und „aktuell“ wechseln, muss kein Klon angelegt werden. Läuft der Übergang über „wird gerade fortgeführt“, wird ein Klon angelegt. Da unbekannt ist, welche der beiden Möglichkeiten realisierbar ist, sollte das Modell beide zulassen.

Ein Fortführungsobjekt ist solange im *Besitz* eines FIS, wie dieses benötigt, um seine Objektsicht herzustellen.

B.2.3.2.5 Zustandskontrolle der Fachinformationssysteme

Fachinformationssysteme müssen nicht immer „laufen“: Es ist entweder ***aktiv*** oder ***nicht aktiv***. Wenn das FIS nicht aktiv ist, müssen deshalb Bearbeitungsanfragen in einer *Warteschlange* aufbewahrt werden. Über setzbare Parameter, wie z.B. die max. Downtime der Anwendung oder Wiederhol-Intervalle, kann gesteuert werden können, wann eine Bearbeitung dadurch gescheitert ist, dass das zuständige FIS nicht verfügbar ist. In einem derartigen Fall könnte z.B. automatisch eine Nachricht an den zuständigen Administrator abgesetzt werden.

Ein Fachinformationssystem hat zwei Zustandsbeschreibungen, die unabhängig vom Aktivitätszustand verfügbar sein müssen:

- konfigurierende Eigenschaften, die von außen abfragbar sein muss, um das FIS ansprechen zu können, z.B. die Serveradresse, die Sichtbeschreibung usw.,
- dynamische Eigenschaften, die nur für die Ablaufüberwachung von Interesse sind.

Auch andere, rein auswertende ***Fachanwendungen*** können durch konfigurierende Eigenschaften gekennzeichnet werden, so dass es sich anbietet, sie in einem Objekt ***Fachanwendungsbeschreibung*** zusammenzufassen. Diese Beschreibungen werden in einem ständig verfügbaren Verzeichnis, dem ***Fachanwendungskatalog***, zusammengefasst. Ein Fachinformationssystem wird dann als Fachanwendung betrachtet, die die Möglichkeit zur Fortführung von Objekten anbietet.

Die dynamischen Zustände werden dagegen von einem Objekt zu verwalten sein, dass unabhägig von den speziellen Fähigkeiten der FIS die gemeinsamen Koordinations- und Informationsdienste anbietet. Dieses Objekt bezeichnen wir als ***Fortführungsmanager***. Es ist für die Anwendungsfälle *Ablauf definieren* und *Ablauf überwachen* zuständig. Um Abläufe flexibel definieren zu können, müssen sie in Sätzen einer Ablaufbeschreibungssprache (z.B. BPEL4WS = Business Process Execution Language for Web Services) beschrieben werden können; die Sätze werden in einem ***Workflowobjekt*** aufbewahrt.

B.2.4 Operationen

Aus den Szenarien und den anderen Vorbetrachtungen bilden wir nun die Operationen. Zu beachten ist dabei zunächst, dass der FFM, die Fortführungsobjekte und die Fortführungsvorgangsobjekte eine Kollaboration bilden, in der die Manipulation der Objekte der

letzten beiden genannten Klassen nur duch den FFM vorgenommen wird. Das gleiche gilt für die dauerhaft verfügbare Information zu den FIS, die in diesen nicht selbst untergebracht werden kann, weil nicht gewährleistet ist, dass sie immer „laufen“.

B.2.4.1 Verantwortlichkeiten

Die Verantwortlichkeiten der Objekte ergeben sich wie folgt:

Fachobjekte

sind die Informationsträger der räumlichen und sachlichen Information. Diese muss setzbar und erfragbar sein.

Fortführungsobjekte

kontrollieren den Zustand der Fachobjekte während einer Fortführung. Der Zustand muss unabhängig von dem jeweils mit der Fortführung beschäftigten FIS verwaltet werden.

Fachanwendungen

müssen für Fortführungsnachrichten erreichbar sein, in dem sie diese abonnieren.

Fachanwendungsbeschreibungen

enthalten Konfigurationsdaten von Fachanwendungen.

Fachanwendungskatalog

erlaubt das Eintragen und Abfragen von Fachanwendungsbeschreibungen.

Fachinformationssysteme

sind Fachanwendungen,

- die Fortführungen von Sichten von Fachobjekten als Beteiligte von Fortführungsszenarien erlauben,
- die Abfragen von Sichten von Fachobjekte erlauben.

Abfragedienst

erlaubt Abfragen von Fachobjektmengen.

Abfrageobjekt

erlaubt Definition eines Abfragesatzes in einer Abfragesprache.

Fortführungsmanager

- überwacht den Ablauf definierbarer Fortführungsszenarien,
- erlaubt Fachinformationssystemen Einblick in Sichten der Fachobjekte während der Fortführung
- vergibt Objektidentifikatoren,
- benachrichtigt auf Wunsch Fachanwendungen über durchgeführte Fortführungen,
- gibt Auskunft über Hilfsdienste zur Fortführung.

Workflowobjekt

erlaubt Definition eines Ablaufbeschreibungssatzes in einer Ablaufbeschreibungssprache.

B.2.4.2 Nachrichtenaustausch

Um das Zusammenspiel von FFM und FIS und den anderen eingeführten Objekten zu analysieren, betrachten wir einige typische Szenarien.

Zunächst ein Szenario für die verteilte Erzeugung von Fachobjekten.

1. Ausgelöst durch ein Fachinformationssystem, z.B. BAUMKAT, soll ein neues Baum-Objekt erzeugt werden. Ein entsprechendes Signal „Objekt muss erzeugt werden“ geht an den FFM. Dieser erzeugt ein Fortführungsobjekt sowie eine neue Objekt-Id #x.
2. Der FFM verfügt über eine Liste von FIS, die über das Ereignis informiert werden müssen. Jedes FIS daraus bekommt ein Signal „Objekt mit Id #x erzeugen“.
3. Jedes FIS erzeugt die Sicht des Objektes, für die es zuständig ist und sendet ein Quittungs-Signal „#x erzeugt“ an den FFM.
4. Bei Vorliegen aller Quittungen notiert der FFM im Fortführungsobjekt, dass das Fachobjekt #x jetzt im Zustand neu+leer vorliegt.

Nun die Änderung eines bestehenden Objektes:

1. Das auslösende FIS signalisiert dem FFM den Beginn einer Fortführung für Objekt #x auf. Der FFM erzeugt daraufhin ein Fortführungsobjekt, in dem es #x einträgt. Zustandsänderung für Objekt #x: wird gerade fortgeführt. Zusätzlich wird eine Objekt-Id #y für einen Klon erzeugt.
2. Alle betroffenen FIS werden vom FFM über ein Signal „Objekt #x fortführen“ benachrichtigt und für die Teilnahme an der Fortführung für dieses Objekt registriert.
3. Jedes FIS reagiert vorbereitend darauf, in dem es z.B. selbst die notwendige Sicht für den Klon leer anlegt, mglw. aber auch gar nichts tut.
4. Nachdem das auslösende FIS mit der Bearbeitung seiner Sicht fertig ist, erzeugt es ein Signal „Sichtfortführung fertig“, das vom FFM an die anderen FIS übermittelt wird. Außerdem deregistriert es sich damit beim FFM für den weiteren Fortführungsprozess an diesem Objekt. Falls die Fortführung schon hier mißlingt, wird der Prozess abgebrochen.
5. Erst jetzt können diese nacheinander weitere Sichten aktualisieren. Der FFM befragt sie nacheinander, ob sie zur Übernahme des Objektes zur weiteren bearbeitung bereit sind. Eines der FIS übernimmt Besitz an dem Fortführungsobjekt. Jetzt gibt es folgende Möglichkeiten:
    a. die Aktualisierung der nächsten Sicht gelingt, dann passiert das gleiche wie nach der Auslösung (Signal Sichtfortführung fertig, Deregistrierung),
    b. oder die Fortführung wird abgebrochen, die Umstände werden protokolliert
    c. oder sie wird zurückgestellt (Wiedervorlageliste, das betroffene FIS darf nicht sofort wieder den Besitz übernehmen!). Die Buchhaltung hierüber sowie das Erkennen endloser Fortführungszyklen obliegt dem FFM. Signal an den Akteur, der für die Fachobjektklasse zuständig ist.
6. Schritt 5 wird für die FIS solange wiederholt, bis ein vollständiger Klon vorliegt.

7. Nun müssen alle FIS aufgefordert werden, die Fortführung sichtbar zu machen. (Signal „Klon mit Original zusammenführen“). Der Klon übernimmt nun die Identität des ehemaligen Originals, dieses wird beseitigt oder in einen historischen Bestand (wie auch immer) eingetragen. Auch dies kann nochmal schiefgehen, z.B. durch Serverausfall.

Diese Szenarien sind noch um Nachrichten zur Informationsübertragung zwischen FIS sowie zur Benachrichtigung anderer Fachanwendungen zu ergänzen. Dies zeigt das folgende Sequenzdiagramm:

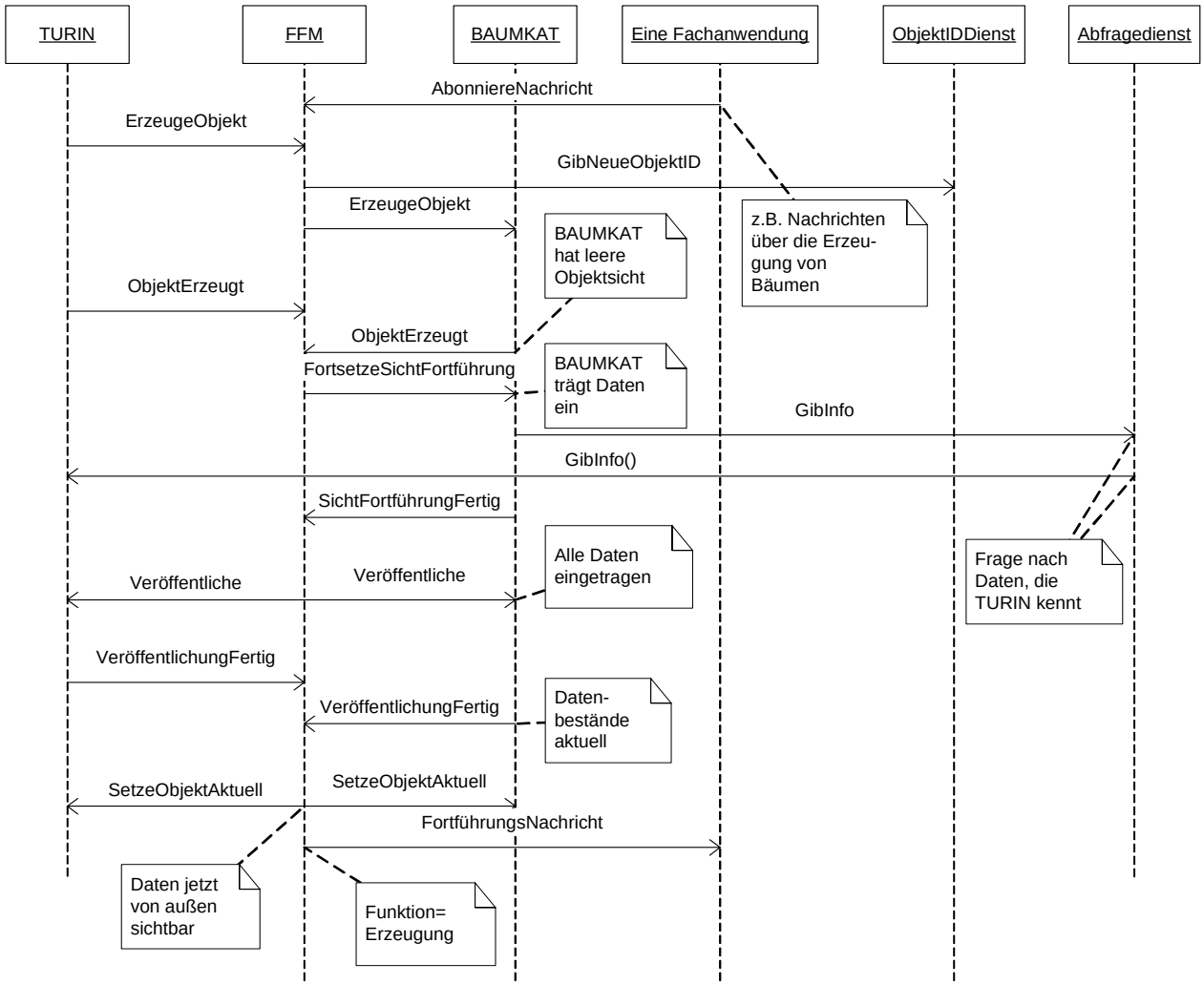


Abbildung 44. TURIN erzeugt neuen Baum.

Es folgen Bemerkungen zu Fortführungen für Objekte mit Abhängigkeiten von anderen Objekten.

Zunächst ist die Frage zu klären, wann eine Abhängigkeit vorliegt. Dies kann nicht aus dem Vorhandensein einer Assoziation zwischen den Klassen der Objekte geschlossen werden. Verfolgt man nämlich Verknüpfungen (d.h. Instanzen von Assoziationen) blind immer weiter, wird ein so konstruierter Abhängigkeitsgraph sehr groß, was dazu führen würde, dass eine lokale Änderung weite Teilbestände aller möglichen Objektarten von

Fortführungen ausschließen würde. Man benötigt daher andere Kriterien für das Bestehen von Abhängigkeiten.

Wer diese Abhängigkeiten feststellt, ist eine offene Frage; genauso, ob, wann, wodurch Abhängigkeiten aufgelöst werden. Technisch können einmal festgestellte Abhängigkeiten etwa in einer Objekt-Registrierung aufgezeichnet werden, die im Falle einer Fortführung befragt wird.

Leider wurden im Forschungsprojekt keine wirklich guten Beispiele für solche kombinierten Fortführungen gefunden, die durch Verfahrensvorgaben (Richtlinien, Umsetzung in IT-Anwendungen) erschließbar sind. Als Beispielidee wurden Verkehrsschilder an Beleuchtungsmasten diskutiert, es ist aber gänzlich unbekannt, wie z.B. der Fall einer räumlichen Versetzung oder der Beseitigung eines solchen Objektes gehandhabt wird. Aus diesem Grunde wurde auch kein Szenario aufgestellt.

Die verschiedenen Szenarien für die Erzeugung, Änderung usw. sind nicht unbedingt festgelegt. Die Abläufe müssen vielmehr selbst konfigurierbar sein. Dazu bekommt der Fortführungsmanager für jedes Szenario eine formale Beschreibung, die spezifiziert, welche FIS in welcher Reihenfolge angesprochen werden müssen. Ob und wie ein FIS ansprechbar ist, sagt der Fachanwendungskatalog, Den Vorgang der Registrierung und ihrer Nutzung zeigt das folgende Sequenzdiagramm:

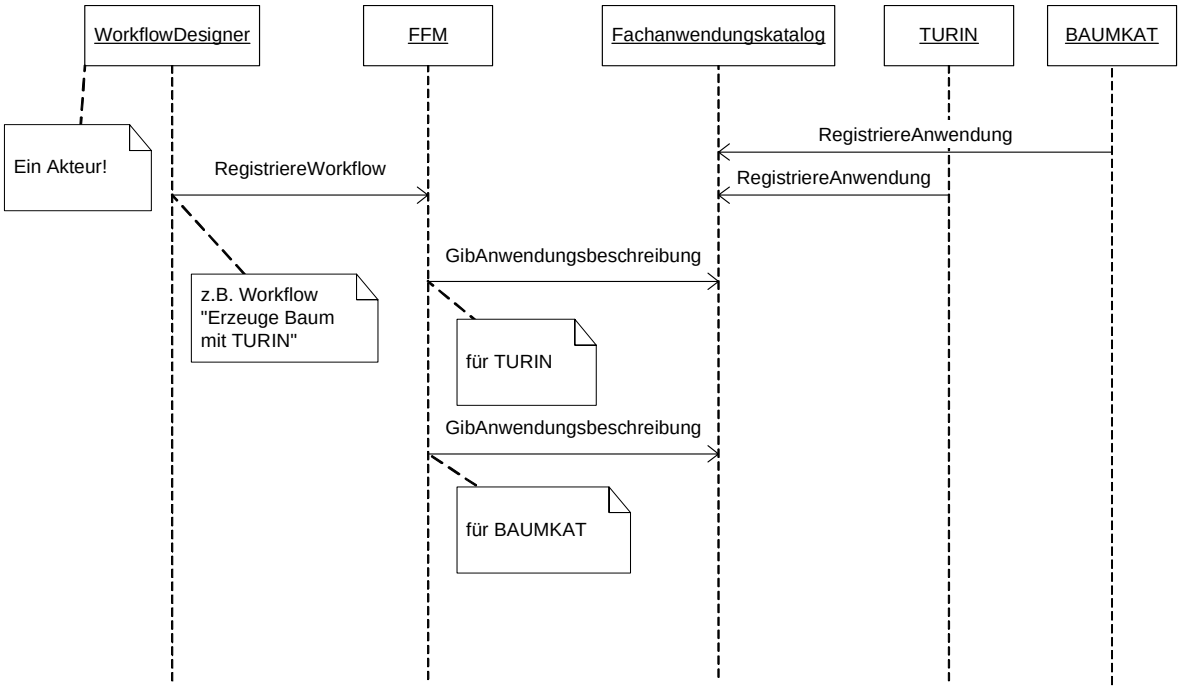


Abbildung 45. Verwendung des Fachanwendungskataloges.

B.2.4.3 Operationsbeschreibungen

Die Operationsbeschreibungen sind konzeptionell. Sie geben nicht immer den Typ der Parameter an. Außerdem sind zu den Operationen auch andere, polymorphe Varianten möglich, die dieselbe Semantik haben (dasselbe Operationskonzept), aber zu unterschiedlichen Vorbedingungen, so dass sich eine andere Parametrisierung ergibt.

B.2.4.3.1 Verwendete Datentypen

Außer den im Leitfaden genannten Basisdatentypen (Leitfaden 7.1) werden allgemein benötigt:

Funktion ist eine Aufzählung der möglichen Fortführungsfunktionen**, z**.B. *Erzeugen, Ändern, Einfrieren, Auftauen, Teilen, Vereinigen*.

OKSTRA_ID, wie im Leitfaden 7.2 erläutert.

OKSTRA_Typ, wie im Leitfaden 7.9 erläutert.

Zur Vereinfachung der Notation wird für die folgenden Beschreibungen vereinbart:

Parameter mit Namen, die auf _**ID** enden, sind vom Typ OKSTRA_ID.

Parameter mit dem Namen **Funktion** sind von Typ Funktion.

Parameter mit dem Namen **Klasse** sind vom Typ OKSTRA_Typ.

B.2.4.3.2 Fachobjekt

Dies ist eine abstrakte Klasse, die unmittelbar von der Klasse ***OKSTRA_Objekt*** (s. Leitfaden 7.2) abgeleitet wird. Sie verfügt selbst über keine Operationen.

B.2.4.3.3 Fortführungsobjekt

Diese Objekte brauchen außerhalb des Fortführungsmanagers nicht sichtbar sein. Daher müssen sie auch keine Operationen für andere Objekte zur Verfügung stellen.

B.2.4.3.4 Fachanwendung

Die einzige Operation, die benötigt wird, ist

FortführungsNachricht(Objek_ID, Funktion)

Teilt der Anwendung mit, dass das Objekt mit der *Objekt_ID* unter Verwendung der *Funktion* bearbeitet wurde.

B.2.4.3.5 Fachanwendungsbeschreibungen

Benötigt werden folgende Attribute:

Name: Der Name der Fachanwendung.

Adresse: Die Adresse, unter der die Anwendung erreichbar ist, z.B. eine URI.

Klassen: Eine Menge von OKSTRA_Typ-Objekten. Die beschriebene Fachanwendung verarbeitet Sichten von Objekten dieser Typen.

Verfügbarkeit: Parameter, die angeben, zu welchen Zeiten bzw. unter welchen Bedingungen die Anwendung erreichbar ist und wann eine Verbindung zu ihr als abgebrochen zu werten ist. Eine genaue Modellierung unterbleibt hier.

AdminKontakt: Daten zur Benachrichtigung des für die Anwendung zuständigen Administrators.

Es können weitere Attribute hinzugefügt werden, z.B. für Sicherheitseinstellungen.

B.2.4.3.6 Fachanwendungskatalog

RegistriereAnwendung(F:Fachanwendungsbeschreibung):FA_ID

Registriert eine Fachanwendung. Die Operation wird von Fachanwendungen bei Installation oder Veränderung der Konfigurationsdaten genutzt, d.h. die Anwendungen registrieren sich selbst.

Jede Fachanwendung ist selbst dafür verantwortlich, die erhaltene *FA_ID*, die die Fachanwendung eindeutig gegenüber den Nutzern identifiziert, dauerhaft und sicher zu speichern.

GibAnwendungsbeschreibung(FA_ID):Fachanwendungsbeschreibung

GibAnwendungsbeschreibung(Name):Fachanwendungsbeschreibung

Mehrere polymorphe Varianten nach Suchkriterium.

B.2.4.3.7 Abfragedienst

Dieser Dienst kann physisch in den Fortführungsmanager integriert sein oder gesondert realisiert sein. Die Aufgabe des Dienstes ist, Abfragen an die einzelnen zuständigen FIS zu delegieren.

GibInfo(ObjektID, AttRelName):OKSTRA_Objekt

Gibt zum Objekt der Objektidentifikation *ObjektID* den Attributwert oder die Relationspartner für Attribut/Relation *AttRelName* als OKSTRA-konformes Objekt zurück. Die Nutzeranwendung kennt das OKSTRA-Modell und weiß daher, welcher Objektklasse das angelieferte Objekt angehört. Die Operation delegiert die Aufgabe an das zuständige FIS.

Die Operation eignet sich in Umgebungen mit begrenzter Netzwerk-Bandbreite nicht zur häufigen Verwendung. Hier sind polymorphe Varianten vorteilhaft, die Information zu mehreren Attributen/Relationen auf einmal übermitteln können.

Abfrage(AbfrageObjekt:Abfrageobjekt):Menge

Führt die Abfrage, die in AbfrageObjekt formal vorliegt aus, und liefert die entsprechende Menge an Objekten zurück. Delegiert zunächst die Abfrage weiter an die zuständigen FIS. Dort werden die Bedingungen aus der Abfrage, die das FIS nicht selbst auswerten kann, ignoriert. Die Operation muss nach Vorliegen aller Teilergebnisse die von den FIS gelieferten Mengen verrechnen.

B.2.4.3.8 Abfrageobjekt

Einziges Attribut ist:

Abfrage: Eine in einer Abfragesprache (z.B. SQL, OGC Filter Encoding, XQuery) abgefasster Satz.

B.2.4.3.9 Workflowobjekt

Einziges Attribut ist:

Ablauf: Eine in einer Ablaufbeschreibungssprache (z.B. BPEL4WS) abgefasster Satz.

B.2.4.3.10 Fachinformationssystem

Persistenzoperationen:

Erzeuge(Klasse)

Erzeugt ein Objekt der gewünschten Klasse. Andere polymorphe Varianten sind denkbar, z.B. mit Übergabe eines Prototyp-Objektes.

Ändere(Objekt_ID, Objekt)

Veranlasst die Änderung des durch *Objekt_ID* gegebenen Objektes. *Objekt* ist ein Prototyp ohne eigene Identität, der die zu ändernde Information übermittelt.

FrierEin(Objekt_ID)

versetzt das durch *Objekt_ID* gegebene Objekt in den eingefrorenen Zustand. Die enthaltene Information wird nicht entfernt, das Objekt ist aber nicht mehr auffindbar, sondern nur noch durch die TaueAuf-Operation ansprechbar. Eine echte Löschung wird als Operation nicht angeboten, jedoch kann die FrierEin-Operation so implementiert werden, dass ein späteres Auftauen ausgeschlossen ist.

TaueAuf(Objekt_ID)

Macht ein eingefrorenes Objekt wieder sichtbar.

Teile(Objekt_ID):Menge

Erzeugt zwei neue Objekte aus dem gegebenen Objekt. Weitere polymorphe Varianten möglich. Siehe Abbildung 42.

Vereinige(Objekt1_ID,Objekt2_ID):OKSTRA_ID

Erzeugt ein neues Objekt aus den zwei gegebenen. Weitere polymorphe Varianten möglich. Siehe Abbildung 43.

Operationen zur Kooperation mit dem Fortführungsmanager:

ErzeugeObjekt(Objekt_ID)

Teilt dem FIS mit, eine Sicht für ein neues Objekt anzulegen.

FortsetzeSichtFortführung(Objekt_ID)

Teilt dem FIS mit, die Fortführung seiner Sicht durchzuführen (z.B. interaktive Erfassung dafür zu ermöglichen).

Veröffentliche(Objekt_ID)

Teilt dem FIS mit, die Änderungen am Objekt dauerhaft verfügbar zu machen. Jedoch wir das Objekt noch nicht aktuell, so dass das Originalobjekt immer noch zugreifbar bleiben muss,

SetzeObjektAktuell(Objekt_ID)

Teilt dem FIS mit, dass das fortgeführte Objekt von nun an den aktuellen Zustand darstellt.

Storniere(Objekt_ID)

Teilt dem FIS mit, die Änderungen am Objekt in der eigenen Sicht zu verwerfen.

VersucheNeu(Objekt_ID)

Teilt dem FIS mit, die Fortführung der Sicht des Objektes zu wiederholen.

Operationen zur Kooperation mit dem Abfragedienst:

GibInfo(Objekt_ID, AttRelName):OKSTRA_Objekt

Gibt zum Objekt der Objektidentifikation *Objekt_ID* den Attributwert oder die Relationspartner für Attribut/Relation *AttRelName* als OKSTRA-konformes Objekt zurück. Achtung: Werte müssen hierfür in OKSTRA_Objekte umgepackt werden!

Abfrage(AbfrageObjekt:Abfrageobjekt):Menge

Liefert zur Abfrage die entsprechende Objektmenge. Nicht in der eigenen Sicht auswertbare Bedingungen werden ignoriert.

B.2.4.3.11 Fortführungsmanager (FFM)

Operationen zur Ablaufdefinition:

RegistriereWorkflow(Klasse, Funktion, WorkflowObjekt:Workflowobjekt)

Die Operation registriert eine formale Beschreibung des Ablaufes, der bei der Fortführung von Objekten der Klasse *Klasse* einzuhalten ist, also im Wesentlichen, in welcher Reihenfolge die FIS angesprochen werden. Sie wird typischerweise von einem interaktiven Konfigurationswerkzeug verwendet, mit dem Ablaufschemata entworfen und dokumentiert werden.

Der Ablauf wird im Objekt *WorkflowObjekt* formal in einer Ablaufsprache ausgedückt. Als Modelle hierfür können z.B. das OMG Workflow-Modell, die Business Process Execution Language BPEL oder die Konstrukte für UML-2.0-Aktivitätsdiagramme dienen. Für eine Funktion dürfen mehrere Abläufe registriert werden, je nach dem, welches FIS die Fortführung anstößt.

Die Operation prüft die Ablaufbeschreibung daraufhin, ob alle vorkommenden FIS registriert sind und wird abgewiesen, wenn dies nicht der Fall ist.

Operationen zur Kooperation mit den Fachinformationssystemen:

SetzeFISaktiv(FIS_ID, aktiv:Bool)

Sobald ein FIS wieder wach geworden ist, sendet es über diese Operation eine entsprechende Mitteilung an den FFM; genauso, wenn es nicht mehr zur Verfügung steht. Der FFM stellt Anfragen an das FIS dementsprechend entweder zu oder hebt sie in einer Warteschlange auf. *aktiv* ist ein zweiwertiger („ja/nein“)-Parameter, der den Zustandübergang kennzeichnet.

ErzeugeObjekt(FIS_ID, Klasse):Objekt_ID

Diese Operation wird vom FIS *FIS_ID* verwendet, um die Erzeugung eines neuen Objektes der Klasse *Klasse* einzuleiten. Der FFM erzeugt eine neue *Objekt_ID* und teilt sie dem FIS als Antwort mit. Der FFM startet den Workflow zum Erzeugen, der das anfragende FIS als Startaktion ausweist und verteilt die Nachricht dann alle beteiligten FIS weiter.

ObjektErzeugt(FIS_ID, Objekt_ID)

Teil dem FFM mit, dass ein FIS seine Sicht eines neuen Objektes erzeugt hat, so dass diese zur weiteren Fortführung bereitsteht. Sind alle Sichten vorhanden, wird der mit dem Objekt verbundene Workflow weitergeschaltet.

BeginneFortführung(FIS_ID, Objekt_ID, Funktion)

Diese Operation wird vom FIS *FIS_ID* verwendet, um die Fortführung eines existierenden Objektes *Objekt_ID* einzuleiten. Der FFM startet den Workflow zu *Funktion*, der das anfragende FIS als Startaktion ausweist und verteilt die Nachricht dann an alle beteiligten FIS weiter. Die gegebene Parametrisierung geht davon aus, dass die Objekt_ID mitteilen kann, welcher Klasse das Objekt angehört. Sieht das Design dies nicht vor, ist die Operation um einen Parameter *Klasse* zu ergänzen. Polymorphe Varianten sind ebenfalls denkbar, z.B. mit zwei Objekt_IDs zum Verschmelzen von Objekten.

SichtFortführungFertig(FIS_ID, Objekt_ID)

Diese Operation teilt dem FFM mit, dass ein FIS mit seinem Teil der Fortführung für Objekt *Objekt_ID* fertig ist. Der FFM schaltet den mit dem Objekt verbundenen Workflow weiter und benachrichtigt die nächsten anstehenden FIS.

Ist der Wokflow beendet, sendet der FFM allen beteiligten FIS eine Veröffentliche-Nachricht..

VeröffentlichungFertig(FIS_ID, Objekt_ID)

Diese Operation teilt dem FFM mit, dass ein FIS mit seinem Teil der Veröffentlichung der Fortführung fertig ist.

Storniere(FIS_ID, Objekt_ID, StornoTyp)

Diese Operation teilt dem FFM mit, dass ein FIS seinen Teil der Fortführung für das Objekt *Objekt_ID* nicht erledigen konnte. StornoTyp sagt, wie hart das Storno ist, z.B. Abbruch der ganzen Fortführung, späterer Neuversuch; Kompensation (d.h. Umgehungsmaßnahmen, Verringerung der Anforderungen usw.). Das Workflowschema entscheidet, auf welche Abbrucharten wie reagiert werden kann. Wird ein Abbruch der Fortführung initiiert, sendet der FFM eine Storniere-Nachricht an alle am Workflow beteiligten FIS.

StornoFertig(FIS_ID, Objekt_ID)

Diese Operation teilt dem FFM mit, dass ein FIS mit seinem Teil des Abbruchs der Fortführung fertig ist.

Operationen zur Kommunikation mit Fachanwendungen:

AbonniereNachricht(FA_ID, Klasse, Funktion)

Registriert eine durch *FA_ID* identifizierte Anwendung zur Benachrichtigung für Fortführungsereignissen an Objekten der *Klasse* gemäß *Funktion*. Sobald der Workflow einer Fortführung erfolgreich beendet ist, werden alle Anwendungen, die ein Abonnement haben, benachrichtigt. Die Nachricht enthält die ObjektID sowie die Funktion. Eine polymorphe Variante für Exemplare (statt *Klasse* dann eine *Objekt_ID*) ist denkbar. Siehe auch

B.2.4.3.12 Weitere Operationen

Der FFM kann die Ermittlung der Objektidentifikation an einen weiteren Dienst delegieren. Dieser muss dann eine Operation

GibNeueObjektId():OKSTRA_ID

anbieten.

Zur Berechnung von Netzknoten-Stationierungen aus Koordinaten und umgekehrt bietet sich ein entsprechender Dienst an. (Ein vereinfachtes Modell dafür wurde bei der Entwicklung des „XML-Prototypen einer Straßeninformationsbank“ spezifiziert.)

GibKoordinateAnStation(AbschnittOderAst,Station):Koordinate

mit der durch den Namen implizierten Semantik.

GibStationZuKoordinate(AbschnittOderAst,Koordinate):Station

mit der durch den Namen implizierten Semantik.

GibAoAZuKoordinate(Koordinate, Abstand):Menge

Gibt eine Menge von Abschnitten oder Ästen, deren Entfernung von der Koordinate durch Abstand begrenzt wird.

B.2.5 Dienste

Als Dienste können nun identifiziert werden:

- der Fortführungsmanager arbeitet intern mit den Fortführungsobjekten und benötigt Workflowobjekte
- der Abfragedienst (es kann mehrere Exemplare geben!) mit den Abfrageobjekten
- der Fachanwendungskatalog mit den Fachanwendungsbeschreibungen
- der Netzknoten-Staionierungsdienst
- der Objekt-ID-Dienst

Die FIS sind selbst nicht unbedingt Dienste in dem Sinne, das sie Leistungen für andere Systeme erbringen. Dennoch müssen mit einer zusätzlichen Schnittstelle ausgerüstet werden, die die Kommunikation mit den Diensten ermöglicht.

Das Objekt-Modell wird in folgenden Klassendiagrammen wiedergegeben. (Die Typen für die Operationsparameter sowie für die Attribute sind aus Platzgründen nicht eingetragen, sie sind den Beschreibungen oben zu entnehmen. Alle Attribute und Operationen sind *public* sichtbar!)

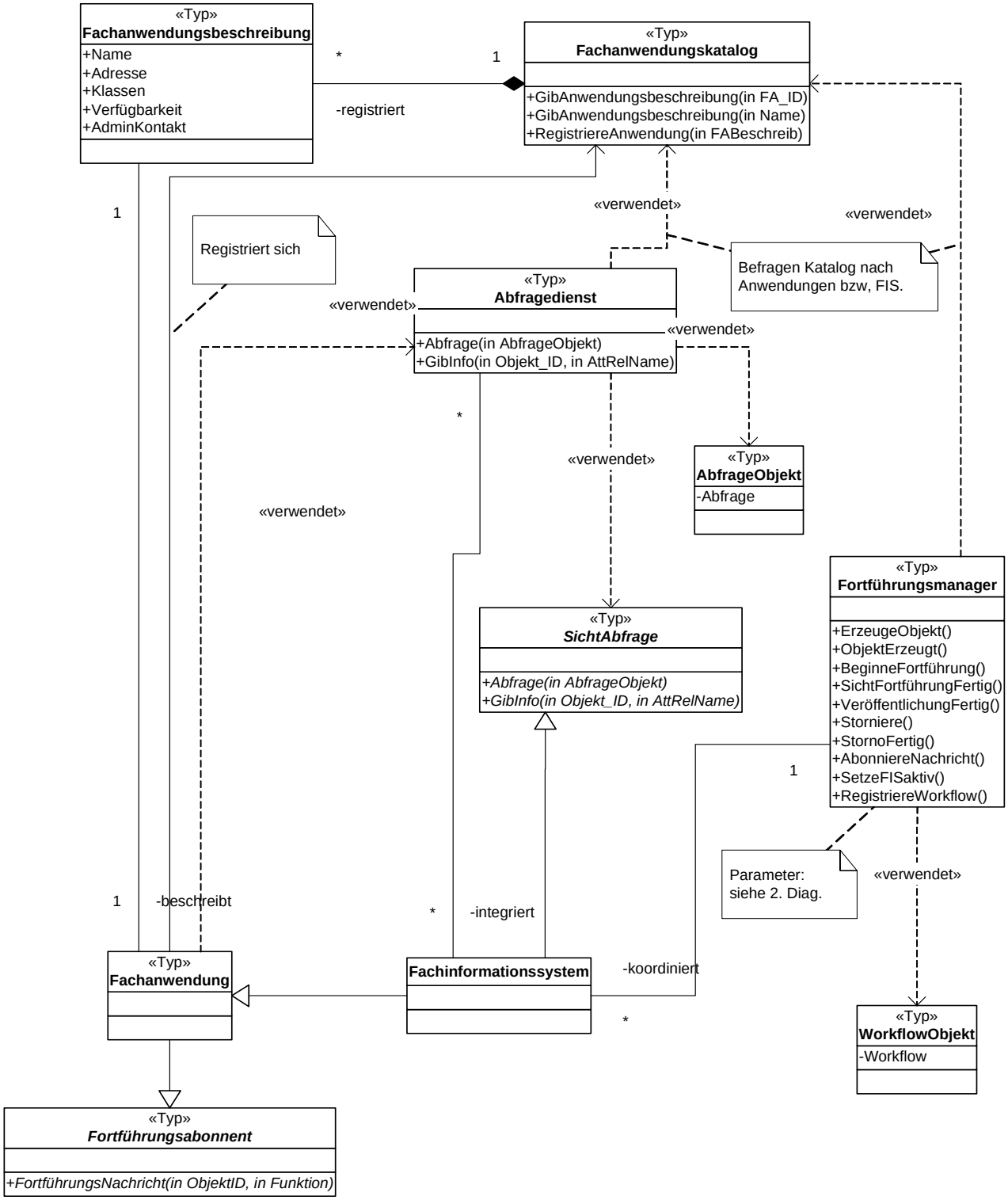


Abbildung 46. Klassendiagramm Abfragen und Regisrtireungen.

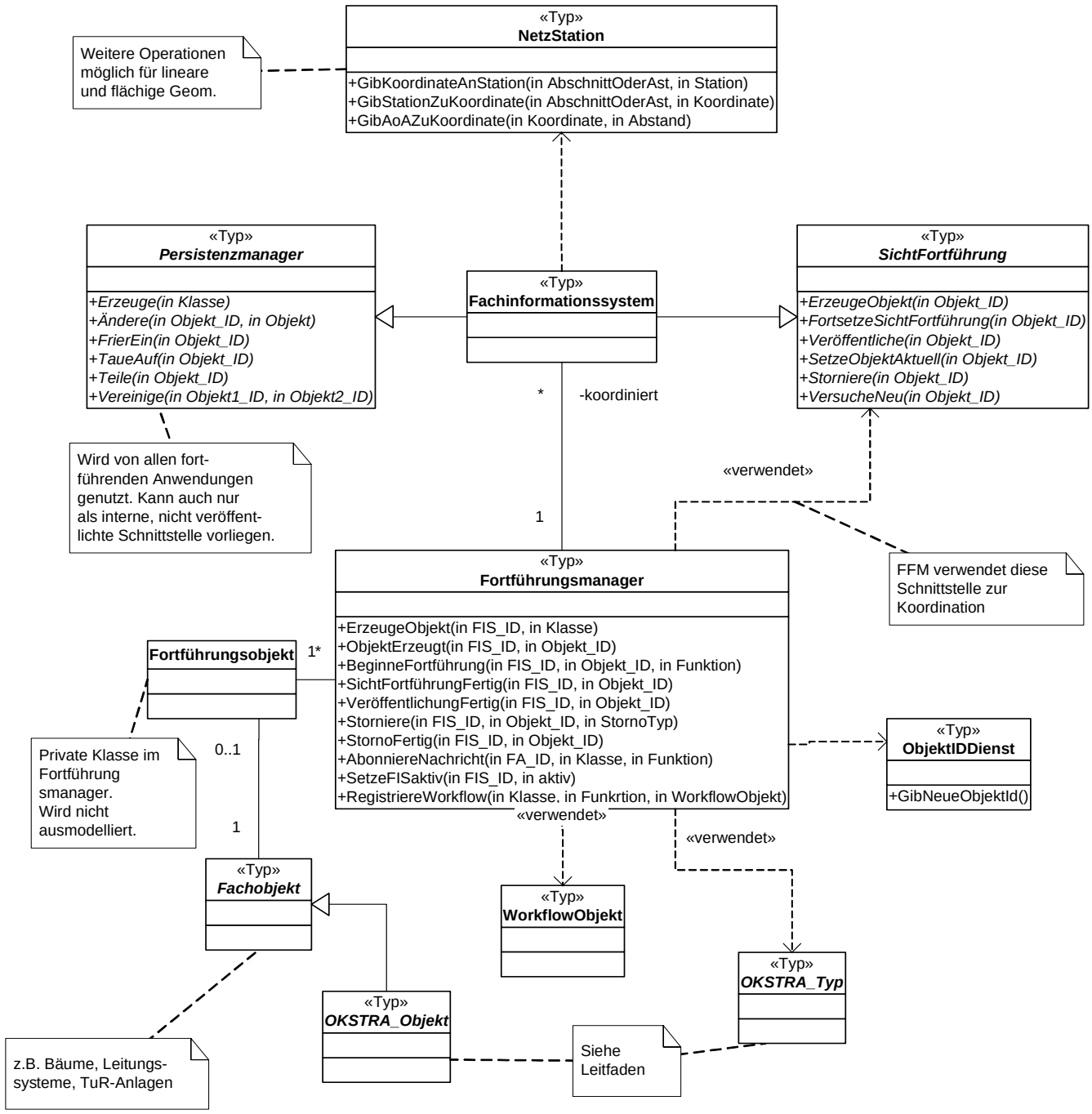


Abbildung 47. Klassendiagramm zur Ablaufkontrolle.

*Anm.: Die Kardinalität der Assoziation zwischen Fortführungsobjekt und Fortführungsmanager ist *:1. Aus technischen Gründen (MS Visio!) im Diagramm nicht korrekt dargestellt.*

B.3 Zusammenfassung

An dieser Stelle würde im weiteren Verlauf der Modellierung der Review des Konzeptionsmodells und die anschließende Aufbereitung in ein Spezifikationsmodell erfolgen.

Als beabsichtigtes Ergebnis der in diesem Anhang vorgestellten Modellierung sollten entstehen:

- Fortgeschriebener Leitfaden. Dies ist das vorliegende Dokument.
- Fortgeführtes Glossar. Die Definitionen für die zentralen Begriffe Fortführung, Bestand, Bestandsobjekt usw. findet sich in B.2.1.2. Eine Übernahme dieser Begriffe in das seinerzeit für den Geschäftsprozesskatalog Neubaumaßnahme aufgestellte Glossar schien sachlich nicht gerechtfertigt – das Glossar bzw. die Erläuterung der Begriffe sollte immer Bestandteil des gerade bearbeiteten Modells sein.
- In den OKSTRA zu übernehmendes Modell, das neben den „statischen“ Objekten auch die Dienste beschreibt, die im Zuge dieses Forschungsauftrages erarbeitet wurden. Dies konnte nicht realisiert werden. Dazu müssten die oben angesprochenen Schritte (externer Review, Aufbereitung, Einbringen in die Pflegestelle) durchgeführt werden.
- Review-Dokumentation zum Leitfaden. Diese wurde in die Fortschreibung des Leitfadens eingebracht.