Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Wirtschaftswissenschaften
des Fachbereichs Wirtschaftswissenschaften
der Universität Osnabrück

Konzeption und prototypische Modellierung einer objektorientierten Architektur für Management Support Systeme (MSS)

vorgelegt von

Dipl.-Kfm. Dietmar Krüger
23. Mai 2001

Dekan: Prof. Dr. L. Knüppel
Referent: Prof. Dr. B. Rieger
Korreferent: Prof. Dr. Th. Witte

Tag der mündlichen Prüfung: 10. Dezember 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ASCII	American Standard Code for Information Interchange
HTML	HyperText Markup Language
ISO	International Standardization Organization
GUI	Graphical User Interface
MSS	Management Support System
OLAP	OnLine Analytical Processing
ooMSS	objektorientiertes Management Support System
SW	Software

Vorwort

Diese Arbeit entstand im Rahmen meiner Lehr- und Forschungstätigkeit am Fachgebiet BWL/Wirtschaftsinformatik II des Instituts für Informationsmanagement und Unternehmensführung (IMU) im Fachbereich Wirtschaftswissenschaften der Universität Osnabrück.

Vor allen anderen danke ich meinem Doktorvater Herrn Prof. Dr. Bodo Rieger für die geduldige und weitreichende Unterstützung. Herrn Prof. Dr. Thomas Witte bin ich für die Übernahme des Korreferats zu besonderem Dank verpflichtet. Herrn Prof. Dr. Bernd Meyer danke ich für die Bereitschaft, den Vorsitz des Promotionsausschusses zu übernehmen und Herrn Prof. Dr. Axel Schreiner für die Nebenfachprüfung.

Weiterhin möchte ich mich bei meinen Kollegen am Fachgebiet Dipl-Kfm. Karsten Brodmann, Dipl.-Kffr. Heike Dalinghaus, Dr. Eitel von Maur, Dipl.-Kffr. Anja Mentrup, Dr. Stephan Postert, Frau Jutta Stelter und Frau Ursula Bertels für die sehr gute Zusammenarbeit bedanken.

Für die fortlaufend moralische Unterstützung danke ich meiner Freundin Ilona Schürmann und meinen Eltern. Zusätzlich danke ich meinen Eltern dafür, daß sie mir meine akademische Ausbildung ermöglicht haben.

Dietmar Krüger, im Dezember 2001

1. Einleitung

1.1 Problemstellung und Zielsetzung

Diese Arbeit befaßt sich mit den Potentialen für eine adäquate Unterstützung der spezifischen Anforderungen von Management Support Systemen (MSS) durch konsequente Anwendung von objektorientierten Konzepten.

Im Gegensatz zu operativen Systemen, die Daten der operativen Abläufe im Rahmen von Administrations- und Dispositionssystemen erfassen und verwalten, besitzen MSS ein analytisches Aufgabenspektrum, durch das betriebliche Fach- und Führungskräfte entscheidungsgerecht mit Informationen, Modellen und Methoden versorgt werden.

MSS als Oberbegriff werden traditionell in drei Bereiche eingeteilt. Diese sind die Versorgung mit Informationen (data-support), die Versorgung mit Entscheidungsalternativen (decision-support) und der Kommunikationsbereich [ChGl99 S.9]. Heutige MSS-Werkzeuge als Träger der Funktionen sind, trotz beobachtbarer Tendenz zur Multifunktionalität, nach einem dieser Einsatzschwerpunkte ausgerichtet. Die MSS-Charakteristika machen es jedoch notwendig, die gesamte Funktionalität integriert abzudecken. Weil die einzelnen Werkzeuge für sich alleine nur im Ansatz bereichsübergreifend unterstützen, ist es notwendig, die Werkzeuge in einer integrierten Umgebung kombiniert einzusetzen. Aufgrund ihres monolithischen Aufbaus fällt dies schwer. Die Werkzeuge werden i.d.R. isoliert angewendet, da sie sich nur schwer kombinieren lassen.

Die Arbeit orientiert sich deshalb nicht an dieser funktionalen Aufteilung. Die Funktionalitäten der einzelnen Bereiche sollen im MSS übergreifend in einer homogenen Umgebung und aus dieser heraus zur Verfügung gestellt werden.

Als Basistechnik für die homogene Umgebung wurde das objektorientierte Paradigma zur Software-Systementwicklung betrachtet. Charakteristisch für die Objektorientierung ist, daß die Daten und die Funktionen als Einheiten auf die Software-Bausteine, die Objekte, aufgeteilt werden. Die objektorientierten Systeme bestehen durchgängig aus einer Menge von kooperierenden Objekten. Dadurch bieten sie wesentliche Potentiale, alle Elemente eines MSS in einer homogenen Umgebung zu integrieren.
Im Rahmen der Arbeit wird ein „Experiment“ durchgeführt, mit dem gezeigt wird, wie die Objektorientierung konsequent auf den MSS-Bereich angewendet werden kann. Dazu ist die Arbeit wie folgt aufgebaut.

1.2 Aufbau der Arbeit

Für die Bewertung des Experiments wird in Kapitel 2 ein Kriterienkatalog erstellt, der sich an den spezifischen integrativen Anforderungen an ein MSS orientiert. Das Kriterienspektrum des Katalogs umfaßt drei Bereiche: die allgemeinen Software-Qualitätskriterien, die Natürlichkeitskriterien und die speziellen MSS-Kriterien.

Das Fundament des Kriterienkatalogs wird aus den grundlegenden Software-Qualitätskriterien abgeleitet. Alle weitergehenden und spezielleren MSS-Kriterien bauen auf dieser Grundlage auf.
Die einzelnen Qualitätskriterien werden beschrieben und ihr Bezug zu dem MSS-Bereich mit Beispielen dokumentiert. Die im Rahmen dieser Arbeit bereitgestellte MSS-Funktionalität wird nach diesen qualitativen Maßstäben ausgerichtet.

Der zentrale Kriterienbereich des Katalogs und damit das Hauptentwurfsziel des MSS ist der Bereich des Natürlichkeitskriteriums. Es beschreibt, wie ein MSS seine Konzepte und Elemente an den Anwender vermittelt. Für die Operationalisierung des Natürlichkeitskriteriums wird zwischen inhaltlichen und bedienerischen Aspekten unterschieden. Es wird festgelegt, nach welchen spezifischen Kriterien die Aspekte des MSS zu bewerten sind, damit sie vom Anwender als natürlich empfunden werden. Sowohl die vom MSS bereitgestellten Inhalte, als auch der Umgang mit dem MSS sollen für den Anwender intuitiv sein. Zusätzlich wirken sich die nach den Natürlichkeitskriterien gestalteten MSS-Aspekte positiv auf die Software-Qualitätskriterien aus.

Die Bedeutung des an Natürlichkeit orientierten MSS-Design wird deutlich an den speziellen MSS-Kriterien, die die Anwendung und Anpassung des MSS auf einer fachlichen Ebene unterstützen. Es werden dazu im Rahmen der Arbeit die drei Kriterien

- Restriktion,
- Führung und
- Anpassung

unterschieden. Die Restriktion beschreibt die im Rahmen des MSS technisch möglichen und fachlich sinnvollen Informationsbeziehungen. Die Führung dokumentiert Kombinationen von fachlichen Informationen in Form von Handlungsalternativen. Ziel ist es, daß durch diese beiden Kriterien die mit der Unterstützungssituation verbundene Komplexität begrenzt und damit der Anwender entlastet wird. Durch das Anpassungskriterium wird die Dynamik behandelt, mit der der Anwender eines MSS konfrontiert ist.

Nach der Operationalisierung der obigen Kriterien werden in Kapitel 3 die Elemente des objektorientierten Paradigmas beschrieben und bzgl. ihrer Anwendbarkeit analysiert. Als die für das objektorientierte MSS-Konzept relevanten Elemente werden Konstrukte zur Komplexitätsbewältigung, die natürliche Modellierungssichtweise und Entwurfsmuster identifiziert. Die Elemente werden hier bzgl. ihres potentiellen Beitrags zu den Kriterien beschrieben und dienen so als Grundlage für das spätere Entwurfskonzept der objektorientierten MSS-Architektur.

Durch die Objektorientierung werden schwerpunktmäßig die Software-Qualitätskriterien des MSS-Kriterienkatalogs adressiert. Dabei wird mittels der objektorientierten Konzepte im MSS die Komplexität bewältigt und eine passende Abstraktionsebene bereitgestellt. Dies ist die Basis für die natürliche Sichtweise der objektorientierten Modellierung, durch die das Natürlichkeitskriterium unterstützt wird.
Der Entwurf von Systemen, die die grundlegenden Software-Qualitätskriterien erfüllen, wird durch Muster unterstützt. Durch die Verwendung von objektorientierten Design Patterns werden MSS flexibler, einfacher und wiederverwendbarer gestaltet, was an dem Stichwort: “designed for change” erkennbar ist.

Im Kernbereich der Arbeit wird in den Kapiteln 4 bis 6 das entwickelte MSS-Konzept in Form eines objektorientierten MSS umgesetzt. Die einzelnen Modellbereiche des objektorientierten MSS und die Beziehungen zwischen den Bereichen werden beschrieben. So können aufgrund dieser Basis auch die Abläufe bei der Anwendung des objektorientierten MSS-Konzepts erläutert werden.
Das Zusammenwirken der zuvor identifizierten ausgewählten objektorientierten Konzepte wird bzgl. der Software-Qualitätskriterien, des Natürlichkeitskriteriums und der MSS-Kriterien bewertet. Die Konzeptentwicklung gliedert sich dazu in die drei Teile Informationsmodell, Interaktionsmodell und MSS-Modell.

Das Informationsmodell, als Grundlage des objektorientierten MSS, wird in Kapitel 4 dargestellt. Es setzt sich aus dem operativen Domänenmodell und dem analytischen Werkzeugmodell zusammen.
Das Domänenmodell bildet die operative Basis des objektorientierten MSS. Als rekonzilierte Informationsbasis benutzt sie die externen operativen Systeme als Informationsquellen [Devl97 S.19, 69].
Das Werkzeugmodell bildet mit seinen Bausteinen das grundlegende Instrumentarium, durch das die analytischen Beziehungen und analytischen Auswertungen von Domänenobjekten im MSS modelliert werden.

In Kapitel 5 wird das Interaktionsmodell als Schnittstelle zwischen dem Informationsmodell und dem Anwender behandelt. Es ermöglicht dem Anwender, auf die für ihn relevanten Ausschnitte des Informationsmodells direkt zuzugreifen. Über das Interaktionsmodell richtet er sich dazu individuell eine Arbeitsumgebung ein, die als Bestandteil alle für die Unterstützungssituation notwendigen Informationsobjekte enthält.

Auf der Basis des Informationsmodells und unter Beteiligung des Interaktionsmodells werden in Kapitel 6 mit dem MSS-Modell die speziellen MSS-Kriterien Restriktion, Führung und Anpassung erfüllt.

Das MSS-Modell wird auf einer Meta-Ebene realisiert. Es beschreibt technische Aspekte in Form von Beschränkungen und fachliche Aspekte durch die elementare Anwendung von Werkzeugobjekten auf das Domänenmodell. Darauf aufbauend wird der Anwender mit Auswertungen und Bemerkungen informativ geführt. Die technischen und fachlichen Aspekte sind die Basis für die Anpassung.

In Kapitel 7 wird abschließend in Form von Anwendungsszenarien die konkrete Anwendung des objektorientierten MSS und das Zusammenwirken aller Systemkomponenten betrachtet.
Die Anwendung wird als Anpassung des integrierten homogenen objektorientierten MSS verstanden. Die verschiedenen Anpassungsebenen für Anwender werden differenziert dargestellt. Der Schwerpunkt liegt bei der Beschreibung der fachlichen Anwenderebenen. Die Erstellung und Verwaltung von Auswertungen und Bemerkungen werden durch Beispiele dokumentiert. Die zugrundeliegenden Abläufe werden durch das Zusammenspiel der einzelnen objektorientierten MSS-Bereiche an Beispielen beschrieben.

Im Kapitel 8 werden zunächst die Ergebnisse der Arbeit zusammengefaßt und bewertet. Es folgt abschließend der Ausblick, in dem auf konzeptionelle und die dafür notwendigen technischen Ergänzungen eingegangen wird.

2. Aufstellung eines Kriterienkatalogs für ein MSS

Ziel der Diskussion der Kriterien in diesem Abschnitt ist es, einen Katalog aufzustellen, der in drei Bereichen mit einem zunehmenden Spezialisierungsgrad Kriterien für die Bewertung und Gestaltung eines MSS beschreibt und operationalisiert. Der Kriterienkatalog dient als Basis für den Entwurf des in dieser Arbeit konzipierten MSS. Der erste Bereich des Katalogs beinhaltet dazu die für das MSS notwendige Software-Qualitätsbasis. Die Diskussion dieser allgemeinen Software-Kriterien wird unter dem Blickwinkel der MSS-Relevanz geführt. Alle weitergehenden MSS-Kriterien bauen auf dieser Grundlage auf. Im zweiten Bereich wird die Natürlichkeit mit ihrem Bezug zu MSS als notwendiges Gestaltungskriterium behandelt. Zur differenzierten Unterstützung bei der Anwendung eines MSS werden abschließend spezielle MSS-Kriterien dargestellt.

2.1 Grundlegende Software-Qualitätskriterien

„Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen.“ [Balz98 S. 257 nach DIN ISO 9126]

Diese allgemeine Definition ist für die praktische Anwendung, insbesondere für den MSS-Bereich, nicht ausreichend. Mit Qualitätsmodellen wird deshalb der allgemeine Qualitätsbegriff durch Unterbegriffe in Form von Qualitätskriterien, wie z.B. Zuverlässigkeit, Änderbarkeit und Effizienz, operationalisiert [StHa97 S.332f., Balz98 S.257]. Aufgrund unterschiedlicher Sichtweisen werden in der Literatur die einzelnen Qualitätskriterien nicht einheitlich verwendet. Sie werden mit überlappenden Bedeutungen beschrieben und in abweichenden hierarchischen Ordnungen kategorisiert [vgl. z.B. Balz96, Dunn93, PaSi94, PoBl93].

Bspw. ist bei Pomberger/Blaschek Robustheit ein Unterpunkt von Benutzerfreundlichkeit, während sie bei Pagel/Six neben Benutzerfreundlichkeit ein eigenständiger Hauptpunkt ist und bei Dunn als Unterpunkt der Zuverlässigkeit gilt [PoBl93 S.10-12, PaSi94, Dunn93 S.20-21].

Für Kriterienkataloge wird als gebräuchliches Klassifizierungskriterium die Unterscheidung zwischen inneren und äußeren Kriterien bzw. Faktoren [u.a. PaSi94 S.51, JCJÖ95 S.464, Meye90 S.2, Heue97 S.160] in Verbindung mit den von Pagel/Six vorgeschlagenen Unterkriterien verwendet. Diese Einteilung ist für diese Arbeit adäquat, da sie eine hinreichende Unterscheidung darstellt. Die folgende Abbildung gibt eine Übersicht (siehe Abbildung 2.1):

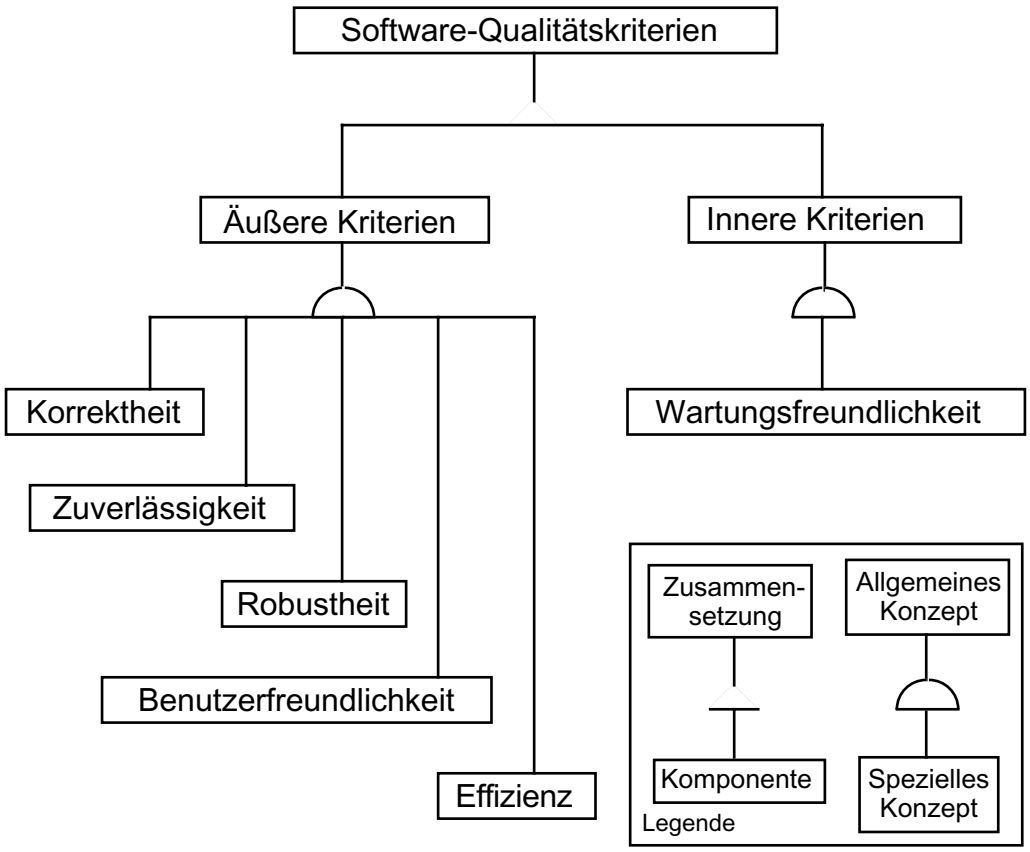

Abbildung 2.1: Software-Qualitätskriterien

Die äußeren Kriterien betreffen die Eigenschaften einer Software, die sie zur Laufzeit zeigen und die damit für den Fachanwender[1] direkt von Interesse sind. Zu den äußeren Kriterien zählen Korrektheit, Zuverlässigkeit, Robustheit, Benutzerfreundlichkeit und Effizienz.

Die inneren Qualitätskriterien spielen für den reinen Fachanwender nur eine indirekte Rolle. Meyer betont in diesem Zusammenhang, daß die inneren Faktoren einzig Mittel zum Zweck der Realisierung der äußeren Kriterien, hierfür allerdings wesentlich bestimmend sind [Meye90 S.2]. Unter dem Oberbegriff Wartbarkeit werden von Pagel/Six die Kriterien, die für den Entwickler von Bedeutung sind, beschrieben.

[1]Fachanwender und Entwickler werden in dieser Arbeit unter dem Oberbegriff des Anwenders zusammengefaßt. Damit sind, wenn der Ausdruck des Anwenders genutzt wird, sowohl Fachanwender als auch Entwickler gemeint.

Auf die einzelnen äußeren Kriterien wird nun eingegangen:

Korrektheit

Das Korrektheitskriterium bezieht sich auf statische und dynamische Eigenschaften des Software-Systems. Die folgende Abbildung veranschaulicht die Komplexität des Korrektheits-Kriteriums (siehe Abbildung 2.2):

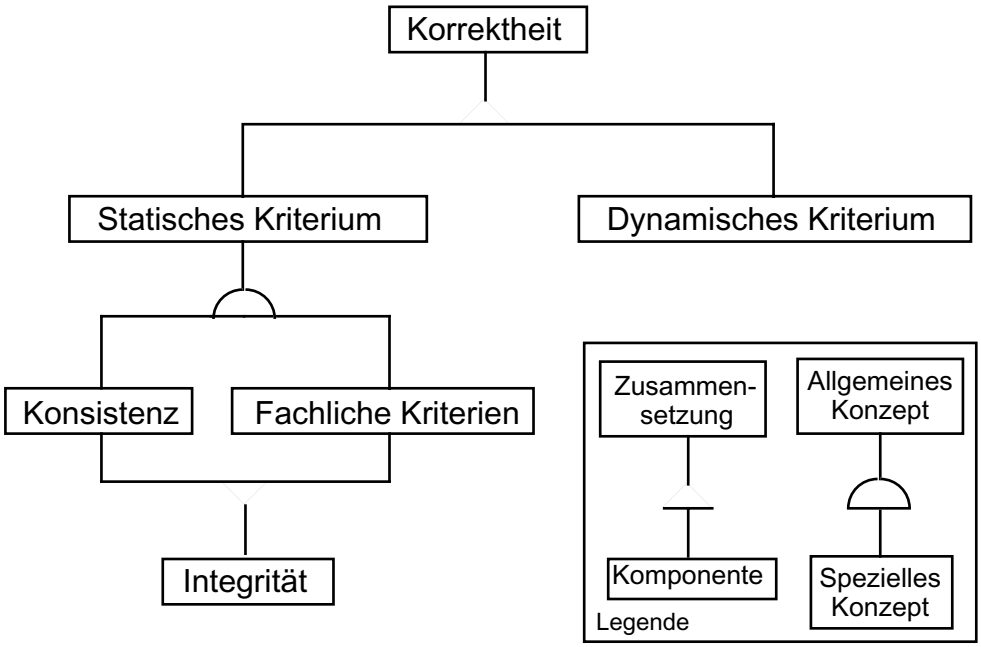

Abbildung 2.2: Korrektheitskriterium

Mit der Erfüllung der statischen Kriterien soll sichergestellt werden, daß der Ausschnitt der Domäne exakt und aktuell repräsentiert wird, was besonders bei MSS wichtig ist. Für den dynamischen Bereich der Korrektheit muß die MSS-Funktionalität mit einem ausreichenden Grad an Fehlerfreiheit auf die MSS-Daten angewendet werden.
Basis für das statische Kriterium ist die Konsistenz. Für den Fall, daß eine konzeptionelle Einheit der Domäne redundant durch mehrere Elemente des Software-Systems verwaltet wird, kann die Konsistenz des Software-Systems verletzt werden.
Z.B. kann ein Kunde versehentlich zweifach gespeichert werden, so daß es zwei gleiche Software-Elemente gibt, die nicht miteinander synchronisiert sind. Bei Zustandsänderungen an einem Kundenelement, z.B. weil er dem Unternehmen einen neuen Auftrag erteilt, kommt es bei Auswertungen der Kundenumsätze zu widersprüchlichen Ergebnissen, je nachdem welcher Kunde berücksichtigt wird, oder zu falschen Ergebnissen, falls die Umsätze eines Kunden mehrfach in die Auswertung einfließen.

Damit ist die Konsistenz notwendige technische Grundlage für die Korrektheit, d.h. wenn ein System inkonsistent ist, dann ist es auch nicht korrekt. Das Gegenteil gilt nicht: Ein konsistentes System muß nicht korrekt sein.
Aus diesem Grund wird die Konsistenz um fachliche Kriterien ergänzt. Durch sie wird der korrekte Zustandsraum des Software-Systems aus einer fachlichen Sicht beschrieben.
Ein Beispiel für ein fachliches Kriterium ist die Beschränkung der Wochenarbeitszeit der Mitarbeiter auf 40 Stunden. Durch dieses Kriterium wird eine Größe von 400 statt 40 Stunden als fachlich falsch erkannt. Durch eine passende Integritätsbedingung, z.B. der Wert liegt unter 100 Stunden, wird die Korrektheit des Systems unterstützt. Eine Auswertung der durchschnittlichen Wochenarbeitszeit der Mitarbeiter kann so nicht durch fehlerhafte Stundenzahlen über 100 verfälscht werden.
Aus der Konsistenz und den fachlichen Kriterien setzt sich das Integritätskriterium zusammen. Durch die Erfüllung der Integrität ist sichergestellt, daß die Handlungen der Anwender die Korrektheit des Software-System nicht gefährden kann [Date95 S.416].

Diese Handlungen bestimmen den dynamischen Aspekt des Korrektheitskriteriums. Zu solchen Handlungen gehören z.B. die Auswertungsfunktionen, mit denen Daten aggregiert werden. Im MSS-Bereich müssen Berechnungen über mehrere Aggregationsstufen fehlerfrei durchführbar sein.
Ein weiteres Beispiel ist die Bestimmung von Abweichungen im Rahmen des Exception Reportings. Die Abweichungen müssen korrekt bestimmt werden, auch wenn sie sich gegenseitig aufheben.
Zu beachten ist, daß die Korrektheit im statischen und dynamischen Bereich voneinander abhängig ist. Ein korrekter Zustand ist die Grundlage dafür, daß das Verhalten korrekte Ergebnisse liefert. Ein fehlerhaftes Verhalten darf nicht die Korrektheit des Zustands verletzen, z.B. durch Widersprüche in den berechneten Auswertungen.

Zuverlässigkeit

Ein Software-System ist zuverlässig, wenn es bei seiner Anwendung über einen bestimmten Zeitraum hinweg das Korrektheitskriterium erfüllt [PoBl93 S.11, StHa97 S.332], d.h. wenn es nicht korrekt ist, dann kann es auch nicht zuverlässig sein.
Durch den zusätzlichen Zeitbezug des Kriteriums besteht die Anforderung darin, daß das Software-System den Anwender über einen bestimmten Zeitraum bei seinen Aufgaben korrekt unterstützt.

Bezogen auf MSS bedeutet dies z.B., daß es als fester Bestandteil des Entscheidungsverhaltens über die gesamte Betriebsdauer hinweg interaktiv nutzbar sein muß, um ad hoc-Abfragen verarbeiten zu können. Eine zeitliche Verzögerung aufgrund eines Systemausfalls stellt den sinnvollen Einsatz für diese Anwendungsform in Frage. Notwendige Wartungsabläufe am MSS, z.B. die Aktualisierung seiner operativen Datenbestände, dürfen deshalb nicht während der produktiven Zeiten zu einem Ausfall des Systems führen.

Robustheit

Ein Software-System ist robust, wenn es beim Auftreten eines extern initiierten Fehlers auf jeden Fall sinnvoll reagiert. Es behandelt dazu den Fehler in kontrollierter Weise und bleibt in einem wohldefinierten Zustand [PaSi94 S.52].
Ein MSS kann mit Fehlern im Rahmen der Kommunikation mit dem Anwender oder einem externen Informationstechnologie-System konfrontiert werden. Jede Fehleingabe des Anwenders muß z.B. abgefangen werden, ohne daß es zu einem Systemabsturz kommt.

Benutzerfreundlichkeit

Ein Software-System ist benutzerfreundlich, wenn der Anwender das System einfach und erfolgreich verwenden kann. Dazu zählt, daß es leicht erlernbar ist und der Anwender eine möglichst kurze Einarbeitungszeit oder Schulung benötigt [vgl. Alte96 S.727].
Die Erfüllung des Kriteriums ist die Grundlage für die Akzeptanz im allgemeinen und im besonderen bei MSS-Fachanwendern, die keine fundierten Informationstechnologie-Kenntnisse besitzen.

Effizienz

Aus einer technischen Sicht heraus nutzt ein effizientes Software-System die Informationstechnologie-Ressourcen nach dem ökonomischen Prinzip für seinen Zweck bestmöglich aus. Dazu zählen u.a. kurze Antwortzeiten für Anfragen und Verarbeitungen, z.B. für eine MSS-Auswertung zur Bestimmung der umsatzstärksten Kunden, und die Beanspruchung von Hardware-Ressourcen, z.B. inwieweit wird die zur Verfügung stehende Rechenzeit für eine Auswertung verbraucht.
Die Festlegung der Effizienz eines MSS beschränkt sich allerdings nicht auf die Bewertung der Ausnutzung von technischen Ressourcen, sondern besitzt zusätzlich einen Anwenderbezug. Deshalb wird dieses Kriterium hier zusätzlich aus einer fachlichen Sicht verwendet.

In diesem Zusammenhang beschreibt es den Aufwand des Anwenders im Umgang mit dem System, um eine MSS-Anfrage an das System zu formulieren und die Ergebnisse zu interpretieren. Die Erfüllung des Kriteriums läßt sich quantitativ mit dem zeitlichen Aufwand bewerten.

Das innere Kriterium der Wartungsfreundlichkeit bezieht sich auf den Entwicklungsprozeß. Es umfasst die Bereiche der Fehlerbehandlung, d.h. die Lokalisierung und Behebung von Fehlern, die Veränderbarkeit, d.h. die Fähigkeit vorhandene Funktionalität zu ändern, und die Erweiterbarkeit, d.h. die Software mit zusätzlicher Funktionalität auszustatten. [Dunn93 S.26].
Die MSS-Domäne verändert sich dynamisch und damit auch die Anforderungen an das MSS. Das MSS muß wartungsfreundlich sein, damit es vom Entwickler entsprechend den Anforderungen flexibel angepaßt werden kann. In diesem Zusammenhang bezeichnen Chamoni/Gluchowski die fehlende Flexibilität als einen speziellen Kritikpunkt von MSS [ChGl99 S.9].
Haslhofer et. al. beschreiben dazu das folgende Beispiel, bei dem das Kundenpotential mit Hilfe eines MSS besser ausgenutzt werden soll. Zunächst werden durch Aktionen Kunden auf Produkte hingewiesen und anschließend die Aktionen n-dimensional nach ihrer Effizienz ausgewertet [Hasl96, CoMM97 S.89]. Notwendig ist dazu, die grundlegenden operativen Elemente und die Aktionen zu ergänzen und die bestehenden Elemente (z.B. die Kunden) anzupassen. Zusätzlich muß eine n-dimensionale Auswertungsfunktionalität zur Verfügung gestellt werden.

Die in diesem Abschnitt beschriebenen äußeren und inneren Software-Qualitätskriterien sind mit ihrem MSS-Bezug eine notwendige, allerdings nicht ausreichende Grundlage für die Konzeption eines MSS. Deshalb wird darauf aufbauend im folgenden Abschnitt mit der Natürlichkeit ein spezielleres Gestaltungskriterium für ein MSS behandelt.

2.2 Natürlichkeit als grundlegendes Gestaltungskriterium

2.2.1 Grundlagen der Natürlichkeit für MSS

Ein Anwender muß ein MSS direkt gestalten können. Die Begründung dafür liegt bei den komplexen und sich dynamisch ändernden Unterstützungssituationen, bei denen der Anwender einen individuellen und von dem situativen Kontext abhängigen Informationsbedarf besitzt.
Standardisierte Auswertungen sind der Ausgangspunkt für detaillierte Analysen. Die dafür notwendigen Auswertungen können wegen der Vielzahl von Auswer-

tungsrichtungen nicht ex-ante komplett aufgestellt werden. Eine ad hoc-Auswertung, bei der Fachanwender schnelle Antworten auf spontane Fragen bekommen, ist notwendig, weil auf jede Antwort neue Fragen entstehen können. Diese Art der Interaktion ist ein charakteristisches MSS-Merkmal [HoWh96 S.143, SpCa82 S.7].

Die indirekte Verwendung eines MSS, unter Beteiligung eines Entwicklers, stellt für die Entscheidungssituation eine zeitliche Verzögerung dar, durch die ein sinnvoller Einsatz in Frage gestellt wird, da nicht schnell genug auf Änderungen der Domäne oder der Anforderungen reagiert werden kann [DeMu88 S.61]. In diesem Zusammenhang wird davon ausgegangen, daß die direkte Anwendung schneller als die indirekte ist [CoMM97 S.3].

Für die direkte Anwendung sind jedoch Voraussetzungen zu schaffen. Der Anwender wird dazu im Rahmen der Arbeit durch das nach dem Kriterium der Natürlichkeit ausgerichtete MSS so unterstützt, daß die dem System zugrundeliegenden technischen Konzepte weitestgehend durch natürliche Konzepte der Realität ersetzt werden. Der Schwerpunkt wird von einer technischen zu einer fachlichen Ausrichtung des MSS verschoben. Die durch das MSS bereitgestellte Sichtweise auf die Unterstützungsituationen wird an die Sichtweise angepaßt, die der Anwender in der Realität von der MSS-Domäne mit ihren Elementen und Eigenschaften besitzt. Damit soll erreicht werden, daß der Anwender im Umgang mit dem MSS nicht seine in der Realität gemachten Erfahrungen in eine technische Welt transformieren muß.
Der Grad der Anpassung zwischen der Sichtweise des Anwenders und des MSS ist an der Distanz zweier Arten von Modellen, die beim Entwurf eines Software-Systems berücksichtigt werden müssen, erkennbar: dem mentalen Modell und dem konzeptionellen Modell.

Mentales Modell

Der Anwender eines Software-Systems verwaltet mentale Modelle über alle Systeme mit denen er konfrontiert wird [Norm87 S.241]. Dazu zählen sowohl Systeme der Realität als auch (virtuelle) Software-System der Informationstechnologie (siehe Abbildung 2.3).

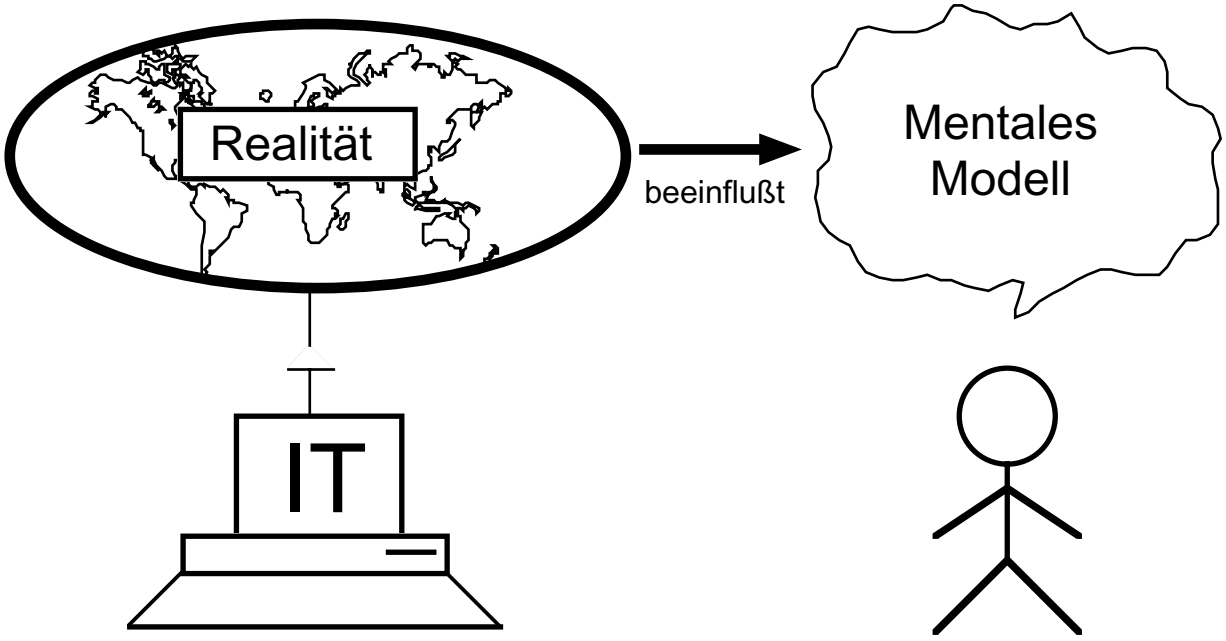

Abbildung 2.3: Mentales Anwendermodell und die Realität

Mentale Modelle sind u.a. wie folgt charakterisiert [Norm87 S.241]:

- Sie sind unvollständig, d.h. sie behandeln nur Ausschnitte der Systeme, mit denen der Anwender umgeht.
- Die Fähigkeit zur Verarbeitung der mentalen Modelle ist (strikt) begrenzt. Komplexe Zusammenhänge werden in mentalen Modellen vereinfacht.
- Sie sind unbeständig. Details des Systems werden vergessen, insbesondere wenn der Anwender mit ihnen länger nicht konfrontiert wurde.
- Die mentalen Modelle eines Anwenders haben untereinander keine festen Grenzen. Ähnliche Sinnbilder und Operationen werden vermischt. Sie können also inkonsistent sein.

Die mentalen Modelle des Anwenders sind außerhalb des direkten Gestaltungsspielraums eines MSS. Sie bilden sich durch die Erfahrungen, die der Anwender macht. Der Umgang mit Software-Systemen spielt dabei nur eine untergeordente Rolle. Schwerpunktmäßig macht er Erfahrungen in der physikalischen Realität, in der er lebt.

Konzeptionelles Modell

Durch die konzeptionellen Modelle der Informationstechnologie-Systeme wird beschrieben, auf welche Weise die Inhalte und Funktionen des Software-Systems an die Anwender vermittelt werden. Die konzeptionellen Modelle bestehen dazu aus Elementen, Operationen

und den Beziehungen zwischen den Elementen, die von dem Anwender beim Umgang mit dem Software-System von Bedeutung sind. Als Bestandteil des Entwurfsmodells eines MSS sind sie im Gegensatz zu den mentalen Modellen direkt gestaltbar.

Das Gestaltungsziel der Natürlichkeit besteht darin, eine Nähe zwischen mentalen und konzeptionellen Modellen zu schaffen, indem ein geeignetes konzeptionelles Modell entwickelt wird (siehe Abbildung 2.4).

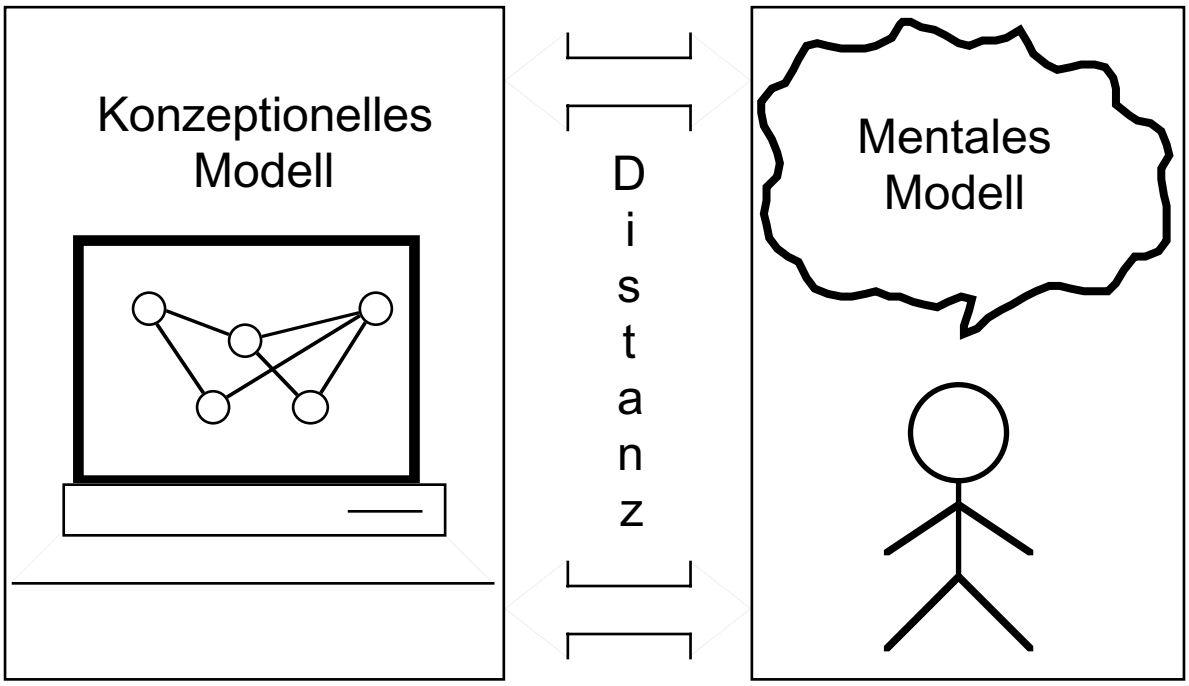

Abbildung 2.4: Konzeptionelles und mentales Modell

Je kleiner die (semantische) Lücke ist, desto geringer ist der mit Verlusten verbundene Umsetzungsaufwand des Anwender zwischen den Modellen.
Die Strategie bei der Natürlichkeit ist, die Distanz zu verringern, indem das konzeptionelle Modell aus bekannten Elementen der Realität, und damit auch Bestandteilen des mentalen Modells über die Realität, aufgebaut wird (siehe Abbildung 2.5).

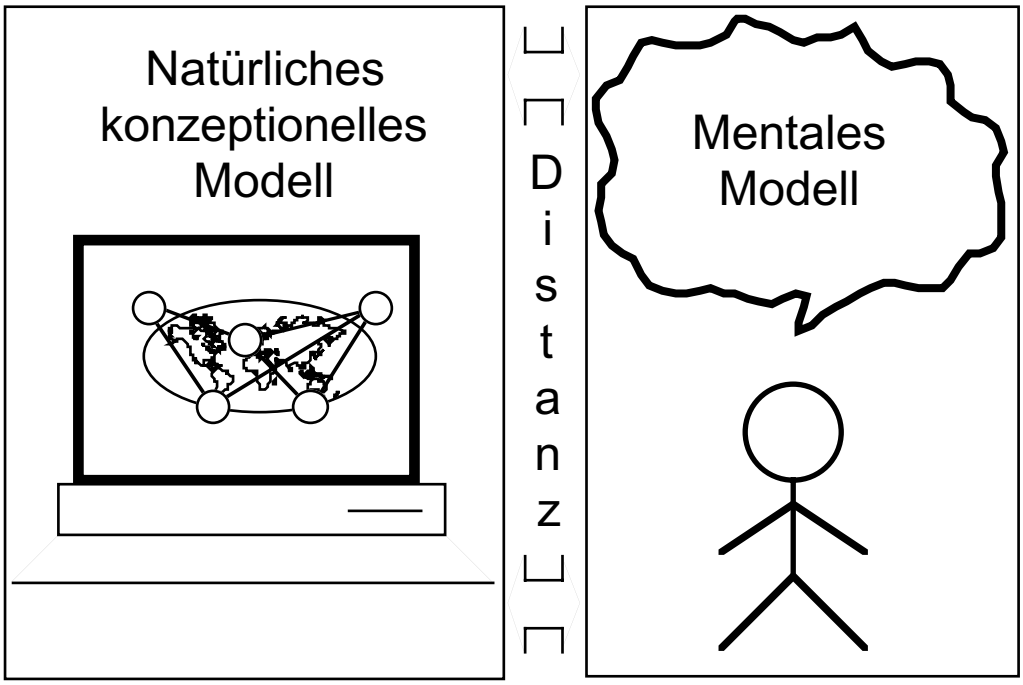

Abbildung 2.5: Verringerte Distanz

Der Anwender soll sich durch seine Kenntnisse über die Realität im Software-System besser zurechtfinden. Allerdings würde eine isomorphe Abbildung der Realität im konzeptionellen Modell das Software-System unnötig beschränken. Die Natürlichkeit wird deshalb so verstanden, daß ausgewählte Charakteristika der Realität um Metaphern ergänzt werden [RoAl90 S.353].
Z.B. besitzt die bei grafischen Benutzeroberflächen verwendete Schreibtisch-Metapher als Element einen Papierkorb, der im Gegensatz zu einem physikalischen Papierkorb, für den Anwender ein unbegrenztes Fassungsvermögen besitzt.
Espen beschreibt dieses erweiterte Gestaltungsziel treffend als: “Create metaphorical Systems in which the user can deduce what to do by refering to what he or she would have done in real life”. [Espe91]

Für die Aufstellung des Kriterienkatalogs eines MSS wird in der Arbeit das Natürlichkeitskriterium konkretisiert und dabei zwischen inhaltlichen und bedienerischen Aspekten des Natürlichkeitskriteriums unterschieden.

Dabei beziehen sich die inhaltlichen Aspekte eines MSS auf die bereitgestellte Infrastruktur. Dazu zählen die Strukturen der Informationen und des Wissens, durch die die Unterstützungssituationen erfaßt werden, sowie die MSS-Funktionen als Basis der MSS-Handlungen.

Durch die bedienerischen Aspekte wird beschrieben, was für die Schnittstelle zu den inhaltlichen Aspekten eines MSS für den Anwender zu beachten ist. Die bedienerischen Aspekte beziehen sich auf die Art und Weise wie die Bestandteile der MSS-Infrastruktur präsentiert und modifiziert werden können.

Das Natürlichkeitskriterium wird im folgenden auf einer konkreteren Ebene beschrieben.

2.2.2 Natürlichkeit durch inhaltliche Aspekte

Die Fortschritte der immer leistungsfähigeren Informationstechnologie-Hardware ermöglichen die Verarbeitung von immer komplexeren Sachverhalten in Informationssystemen. Die Komplexität der inhaltlichen Aspekte, mit denen der Anwender in seiner Domäne konfrontiert wird, können zunehmend in einem MSS verwaltet werden. Das Ziel ist es, die Komplexität so in das MSS zu verschieben, daß sie dort in geeigneten Strukturen effizient organisiert wird. Der Anwender selbst wird so mit einer geringeren Komplexität konfrontiert und damit kognitiv entlastet. Um dies zu erreichen sollte jede Komplexität, die durch den Umgang mit Informationstechnologie-Konzepten zusätzlich entsteht, vermieden werden. Zusammengefaßt wird der Anwender somit durch die komplexere Repräsentation des Unternehmens und seines Umfelds unterstützt [Tayl92 S.12].

Die effiziente Organisation von beliebiger Komplexität ist die notwendige Basis für die Unterstützung des Anwenders mit natürlichen inhaltlichen Aspekten. Die natürlichen Aspekte basieren dabei auf zwei Säulen. Zum einen soll dem Anwender durch das MSS eine passende Abstraktionsebene bereitgestellt werden, die seinem mentalen Modell entspricht. Sie ermöglicht eine direkte Zuordnung zwischen den für den Anwender relevanten Elementen der Domäne und den Elementen des konzeptionellen Modells [ReSu95 S.18]. Zusätzlich unterstützt das natürliche konzeptionelle Modell die Vereinheitlichung der abweichenden individuellen Modelle verschiedener Anwender.
Zum anderen soll der Anwender mit der gesamten Bandbreite von Informationen versorgt werden, die für ihn relevant sind. Der höhere Informationsgehalt unterstützt nach der Theorie des Informationsreichtums die Entscheidungsprozesse des Anwenders [LeDa88].

Der Schwerpunkt der Informationsversorgung bei MSS liegt bei strukturierten, quantitativen, internen Informationen, die aus den operativen Systemen gewonnen werden. Allerdings sind unstrukturierte, qualitative und externe Informationen als wichtige Bestandteile im Rahmen eines MSS zusätzlich bereitzu-

stellen, da sie einen wesentlichen Einfluß auf die Entscheidungsprozesse haben [Devl97 S.50, 134, Mint91]. Mit der Aufhebung der (auch technisch bedingten) Beschränkung des Informationsspektrums sollte das inhaltliche konzeptionelle Modell deshalb so ausgestaltet sein, daß es die gesamte Bandbreite der Informationen aufnehmen kann [RBK+98 S.137]. Der Evolutionspfad von MSS zeigt deshalb, ausgehend von den operativen Systemen, in diese Richtung [SeBa97 S.5, Oppe95 S.188].

Bei der Betrachtung der Eigenschaften des Informationsspektrums beschreibt der Strukturiertheitsgrad, wie stark Informationen in ihrem Aufbau festgelegt sind. Mit der Nutzung des WWW als Informationsquelle ist die Mehrzahl der zur Verfügung stehenden Informationen schwach oder unstrukturiert [Abit97, Bune97 S.117]. Nach Choo äußert sich Unstrukturiertheit in folgenden Informationsmerkmalen:

- Die Informationen enthalten veränderliche Bestandteile.
- Die hinter den Bestandteilen verwalteten Datentypen sind nicht definiert, und ihre Wertebereiche sind nicht eingeschränkt.
- Beziehungen zwischen den Informationen sind nicht festgelegt [Choo98 S.34].

Multi-mediale Informationen, wie Grafiken, Diagramme, Karten, Fotos, Audiodaten oder Videos sind Beispiele für sehr schwach strukturierte Informationen. Textuelle Informationen, wie Notizen, Memoranden, Vorschläge, Berichte, Nachfragen und Nachrichten, besitzen einen mittleren Strukturiertheitsgrad und sind damit semi-strukturiert [Devl97 S.50].
Zu beachten ist für die Integration und Bereitstellung von strukturierten Informationen in einem MSS, daß Informationen mit einem zunehmenden Grad an Heterogenität ähnliche Charakteristika wie unstrukturierte besitzen. Das Methodenspektrum zur Verarbeitung im Rahmen eines MSS muß dies berücksichtigen [RBK+98 S.137].
Als Beispiel sollen zwei Arten von Informationselementen dienen, wie z.B. Kunden und Produkte, die jeweils für sich strukturiert sind. Wenn im Rahmen einer MSS-Analyse Informationen beider Mengen integriert werden, dann besitzt diese Informationsmenge Merkmale von Unstrukturiertheit, weil zwischen den Informationsarten keine direkten Beziehungen festgelegt sind. Als Lösung werden zusätzlich die Informationselemente Aufträge eingeführt, durch die eindeutige Beziehungen zwischen Kunden und Produkten dokumentiert werden. Für das Beispiel ist ein Kunde mit seinen Aufträgen und ein Auftrag ist mit Produkten verbunden. Auf diese Weise können die an der MSS-Analyse beteiligten Informationen strukturiert werden.

Zunehmend müssen externe Informationen als Einflußfaktoren auf das Geschäftsfeld berücksichtigt werden. Es reicht nicht mehr aus, nur die Entwicklung bei Mitbewerbern und Kunden zu beachten. Unternehmen müssen zusätzlich technologische Entwicklungen, politisch-rechtliche Rahmenbedingungen, wirtschaftliche Entwicklungen oder demographische Trends berücksichtigen. Diese Informationen dienen zum Verständnis, wie sich das Umfeld des Unternehmens entwickelt, andere Unternehmen handeln und das Unternehmen im Vergleich steht [Choo98 S.30, Devl97 S.59-61, 134-136].
Das Angebot an externen Informationen ist überwiegend semi-strukturiert [HeHe95 S.135]. Im Zeitablauf sind die externen Informationen größeren Strukturschwankungen unterworfen als die tendenziell stärker strukturierten internen Informationen [vgl. Vets85 S.12].

Neben dem Strukturiertheitsgrad wird häufig zwischen qualitativen Informationen (unterteilt mit einer Nominalskala) und quantitativen Informationen (unterteilt mit einer Kardinalskala) unterschieden. Angestrebt wird für die qualitativen Informationen eine gleichwertige Integration und die Verknüpfung mit quantitativen Informationen [Mint91 S.73, 87]. Mintzberg begründet die Notwendigkeit von qualitativen Zusatzinformationen mit den Grenzen formaler quantitativer Informationen [Mint91 S.85f.]:
Danach sind quantitative Informationen für sich allein in ihrer Aussagekraft zu begrenzt. An ihnen sind keine (qualitativen) Risiken erkennbar, die Qualität einer möglichen Entscheidung ist nicht wahrnehmbar, und sie sind unsensibel gegenüber extremen Situationen. Zusätzlich werden quantitative Informationen durch eine Aggregation zu sehr verallgemeinert, wodurch der Anwender den Bezug zum Einzelfall verliert [Mint91 S.86]. Ihn interessieren z.B. Detailinformationen, warum ein Kunde nicht mehr kauft.

Qualitative weiche Informationen sind somit zur Ergänzung der quantitativen harten Informationen notwendig. Durch die Kombination verstärkt sich die Aussage der quantitativen Informationen und ein höheres Verständnis der Situation, die zu den quantitativen Informationen geführt hat, wird erreicht [WaHR97 S.219, Alte96 S.30].
Beispiele für qualitative Informationen sind Evaluationen, Befragungsergebnisse, Kommentare, strategische Bewertungen des Unternehmens, Bewertungen von abweichenden Kennzahlen und Einschätzungen.
Eine Begründung für abnehmende Verkaufszahlen sind z.B. ein für den Export ungünstiger Wechselkurs oder ein Überangebot auf dem Markt. Ein ertragsschwaches Produkt wird weiterhin produziert, weil es für den Markteintritt in einer Region benötigt wird, obwohl es einen negativen Deckungsbeitrag aufweist.

Der Anwender erkennt durch die im Rahmen des MSS bereitgestellten inhaltlichen Aspekte die physikalischen (z.B. ein Kunde) und konzeptionellen Elemente (z.B. ein Vorgang) seines Anwendungsgebiets auf einer passenden Abstraktionsstufe und der gesamten Bandbreite von Informationen. Durch die Nähe der mentalen und konzeptionellen Modelle verringert sich der kognitive Umsetzungsaufwand, den der Fachanwender für die inhaltlichen Aspekte zu leisten hat. Dies unterstützt die Benutzerfreundlichkeit. Der Umgang mit den Inhalten des Informationstechnologie-System wird als natürlich empfunden, wodurch es intuitiv anwendbar und leichter erlernbar ist.

Der Entwickler versteht die Domäne so, wie sie ihm vom Fachanwender vermittelt wird. Für die Korrektheit und Wartungsfreundlichkeit des Software-Systems ist das Verständnis des Entwicklers von großer Bedeutung [vgl. RoAl90 S.348]. Zusätzlich kann der Entwickler die für den Fachanwender relevanten realen und abstrakten Elemente in der Realität identifizieren, beobachten und direkt im inhaltlichen konzeptionellen Modell umsetzen. Es besteht aus einheitlichen Begriffen, die eine gemeinsame Kommunikationsbasis zwischen Fachanwendern und Entwicklern bilden. Das Vokabular der Domäne wird durch die Benennung und Definition der Modellelemente einheitlich beschrieben. Die Fachanwender und Entwickler erhalten das gleiche Verständnis für die Bedeutung eines Fachbegriffs. Bisher mußte sich der Fachanwender im Umgang mit dem MSS an die Begriffswelt des Entwicklers anpassen. Jetzt paßt der Entwickler das MSS an die Begriffswelt des Fachanwenders an. Die Begriffskonsolidierung ist ein Erfolgsfaktor, der auch außerhalb des Informationssystem-Bereichs im gesamten Unternehmen relevant ist [Lehm99, RaWa95 S.154].

2.2.3 Natürlichkeit durch bedienerische Aspekte

Die Elemente des konzeptionellen Modells für die bedienerischen Aspekte legen fest, wie die Informationen und Funktionen des MSS an den Anwender übertragen werden und wie der Anwender mit dem MSS interagiert. Die MSS-Informationen sollen mit ihren natürlichen inhaltlichen Aspekten unter Beachtung der bedienerischen Aspekte mit natürlichen Konzepten an den Anwender vermittelt werden. Bei der Vermittlung erhält der Anwender einen Eindruck von dem zugrundeliegenden MSS. In diesem Zusammenhang spricht man auch von einem “look & feel” [Togn91 S.129]. Ziel ist es, durch ein konzeptionelles Modell eine durchgängige „virtuelle“ Realität im MSS zu schaffen, in der der Anwender mit allen für eine Unterstützungssituation relevanten Informationen natürlich interagieren kann.

Die Kriterien für die Konzeption der bedienerischen Aspekte sind aus grundlegenden Erfahrungen des Anwenders im Umgang mit der Realität abgeleitet:

1. Das erste und wichtigste Kriterium ist die verlustfreie Übertragung der bei den inhaltlichen Aspekten beschriebenen natürlichen Infrastruktur. Die Informationen sollen dem Anwender mit ihrer Komplexität auf der passenden Abstraktionsebene und mit dem gesamten Informationsspektrum direkt vermittelt werden. Die Aufgabe der Interaktion besteht darin, den Anwender in einer natürliche Weise auf die Inhalte zugreifen zu lassen.

2. Es sollen die bedienerischen Aspekte in dem Sinne homogen sein, daß sie sich vollständig aus gleichartigen konzeptionellen Bedienungselementen zusammensetzen.
   Die Elemente haben die Aufgabe, jede beliebige Information geeignet abzubilden. Sie müssen dazu in der Lage sein, durchgängig jede Detailebene der Information, von einzelnen Eigenschaften eines Informationselements bis zu komplexen Sichten auf eine Menge von Informationen, adäquat zu repräsentieren. Weiterhin muß jede mögliche Interaktion mit einem Informationselement, wie z.B. die Eingabe eines Datums, die Aktivierung einer Funktion oder die Verwaltung von Verbindungen zu anderen Informationselementen, für den Anwender über Bedienungselemente durchführbar sein.

3. Die Homogenität als bedienerischer Aspekt ist auch maßgebend für die Konsistenz. Sie beschreibt als Bedienungsaspekt, daß dem Anwender von dem MSS ein stabiler Eindruck mit festen Gesetzmäßigkeiten vermittelt wird. Sie sind dauerhaft gültig, vergleichbar mit physikalischen Gesetzmäßigkeiten. Bei der Vermittlung von Informationen über Bedienungselemente gefährdet eine widersprüchliche Darstellung von Informationen den stabilen Eindruck, den der Anwender vom MSS besitzt.
   Mit dem Auftreten von Ausnahmeereignissen im MSS darf der Anwender nicht seine vertrauten Bedienungselemente verlieren. Ein Beispiel für eine fehlende Konsistenz ist, daß bei einem Berechnungsfehler die grafisch repräsentierten Bedienungselemente, mit denen der Anwender vertraut ist, vom Bildschirm verschwinden und er mit textuellen Repräsentationsformen konfrontiert wird, die ihm einen Fehler melden. Damit wird der vermittelte Eindruck des Anwender zerstört [vgl. Togn91 S.235-242].
   Das allgemeine Software-Qualitätskriterium Konsistenz gilt auch hier für die Darstellung der Eigenschaften von Informationselementen.

Eine widersprüchliche inkonsistente Darstellung gefährdet den stabilen Eindruck des Anwenders. Z.B. muß die von mehreren Bedienungselementen visualisierte Umsatzhöhe, die mit einem Kunden erzielt wurde, eindeutig sein.
Durch das Einhalten dieser Gesetzmäßigkeiten kann der Anwender Erfahrungen, die er beim Umgang mit dem System macht, wie z.B. beim MSS-Beispiel die Anwendung einer Auswertungsfunktion auf bestimmte Informationen, intuitiv auf andere Teilbereiche des Systems übertragen [vgl. Appl87 S.6].

4. Die Interaktion soll einen möglichst großen Freiraum durch einen wahlfreien Zugriff auf die Informationen und Funktionen des MSS bereitstellen.
Der Anwendungsprozeß besteht nicht aus einer vom System festgelegten Abfolge von Handlungen, durch die dem Anwender fest vorgeschrieben wird, was er im nächsten Bearbeitungsschritt zu tun hat. Sondern der Anwender steuert das System. Er steht im Zentrum des Systems, und es reagiert auf seine Handlungen [Appl87 S.7]. Der Anwender kann sich so spontan und intuitiv entscheiden, was seine nächste Aktion ist.
Die Grundlage dafür ist, daß das System dem Anwender einen Handlungsraum bereitstellt. Innerhalb des Raums erlaubt es ihm alle Handlungen, die nicht die Korrektheit des Systems gefährden.
Durch die Möglichkeit der kurzfristigen Wechsel zwischen den Tätigkeiten unterstützt es eine Arbeitsweise, die wie durch empirische Untersuchungen belegt, für den Arbeitsalltag des Anwenders charakteristisch ist [Hale93 S.14].

Für Fachanwender ist der Umgang mit einem MSS, bei dem die bedienerischen Aspekte nach den natürlichen Kriterien des konzeptionellen Modells ausgerichtet sind, einfach zu erlernen. Durch die Beachtung der natürlichen Bedienungskonzepte wird erreicht, daß sich der Anwender auf seine inhaltlichen Aufgaben konzentrieren kann und sich nicht schwerpunktmäßig mit technischen Aspekten befassen muß. Dies ist traditionell problematisch, weil der Anwender diese Aufgabe zusätzlich im Umgang mit der Informationstechnologie zu leisten hat [Niel93 S.85].

2.3 Spezielle MSS-Kriterien

Die Bereitstellung von inhaltlicher und bedienerischer Natürlichkeit im Rahmen eines MSS unterstützt den Anwender, ist aber nicht ausreichend für das Kriterienprofil eines MSS. Darüber hinaus sind als weitere Kriterien spezifische MSS-Funktionen bereitzustellen. Im Rahmen dieser Arbeit werden die von Silver vorgeschlagenen speziellen MSS-Kriterien Restriktion, Führung und Anpassung als Basis verwendet, um die fachliche Anwendung eines MSS zu beschreiben [Silv91]. Entlang dieser Kategorien wird nun beschrieben, wie sie auf MSS angewendet werden.

2.3.1 Restriktion

Die durch ein MSS dem Anwender zur Verfügung gestellte Unterstützungsumgebung besteht aus einer Menge von MSS-Informationen und MSS-Funktionen. Die Funktionen werden bei Analysen und Auswertungen auf die Informationen angewendet. Die unterstützten Anwendungsalternativen sind dabei beschränkt durch die technisch möglichen und fachlich sinnvollen Kombinationen der zur Verfügung stehenden Informationen und Funktionen (siehe Abbildung 2.6).

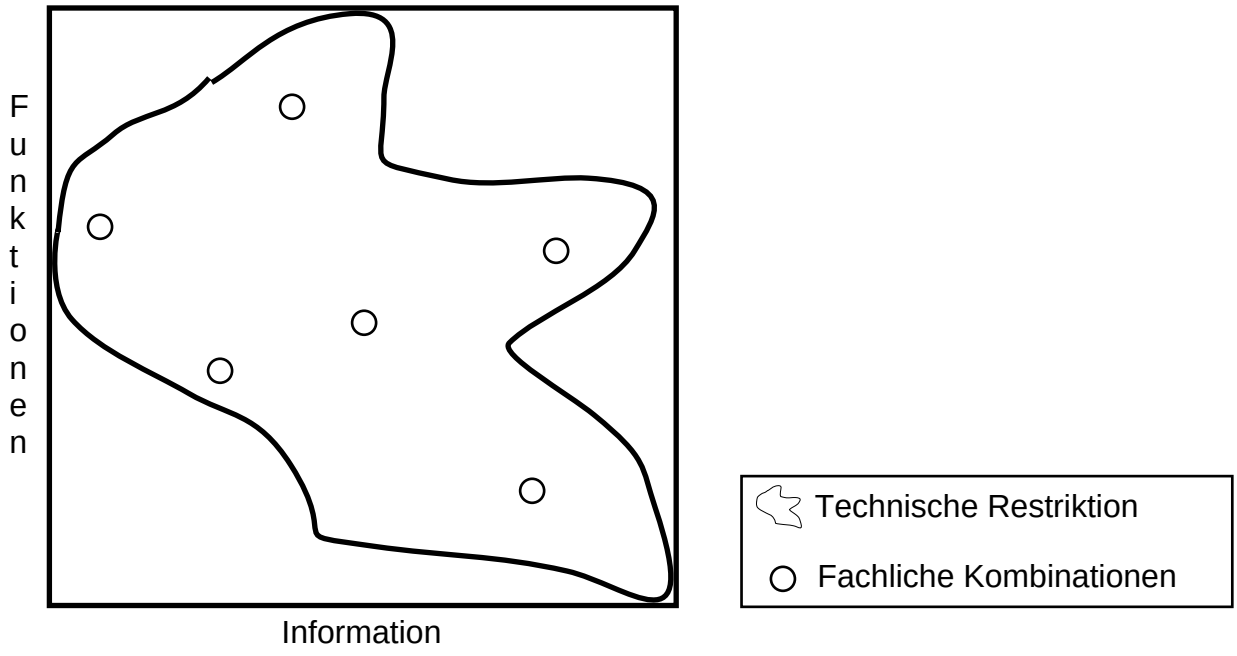

Abbildung 2.6: Restriktion durch Kombinationen von Informationen und Funktionen im Anwendungskontext

Beispiel: Es müssen für eine Aggregation numerische Informationen, z.B Umsatzzahlen von Kunden, und eine Aggregationsfunktion zur Verfügung stehen. Mit der Anwendung der Aggregationsfunktion auf die Kundenumsätze erhält der Anwender eine fachlich bedeutende Auswertung. Die Aggregation von Kundennummern ist dagegen ein Beispiel für eine technisch mögliche, aber fachlich nicht sinnvolle Auswertung.
Das Unterstützungskriterium der Restriktion beinhaltet die Übermittlung dieser Menge von technischen und fachlichen Handlungsalternativen an den Anwender. Das Systemverhalten wird beschrieben durch seine Möglichkeiten und zusätzlich durch seine Einschränkungen. Zum effektiven und effizienten Einsatz ist für den Anwender das Verständnis beider Gebiete notwendig [Silv91 S.114].

2.3.2 Führung

Ergänzt wird die Restriktion um die Führung. In dem vom MSS bereitgestellten Handlungsraum besitzt der Anwender, trotz der Beschränkungen, einen schwer überschaubaren Handlungsfreiraum mit einer Vielzahl von Kombinationsmöglichkeiten. Die Führung eines MSS beeinflußt, ergänzend zu der Beschränkung, die Handlungsfreiheit in der Unterstützungssituation (siehe Abbildung 2.7).

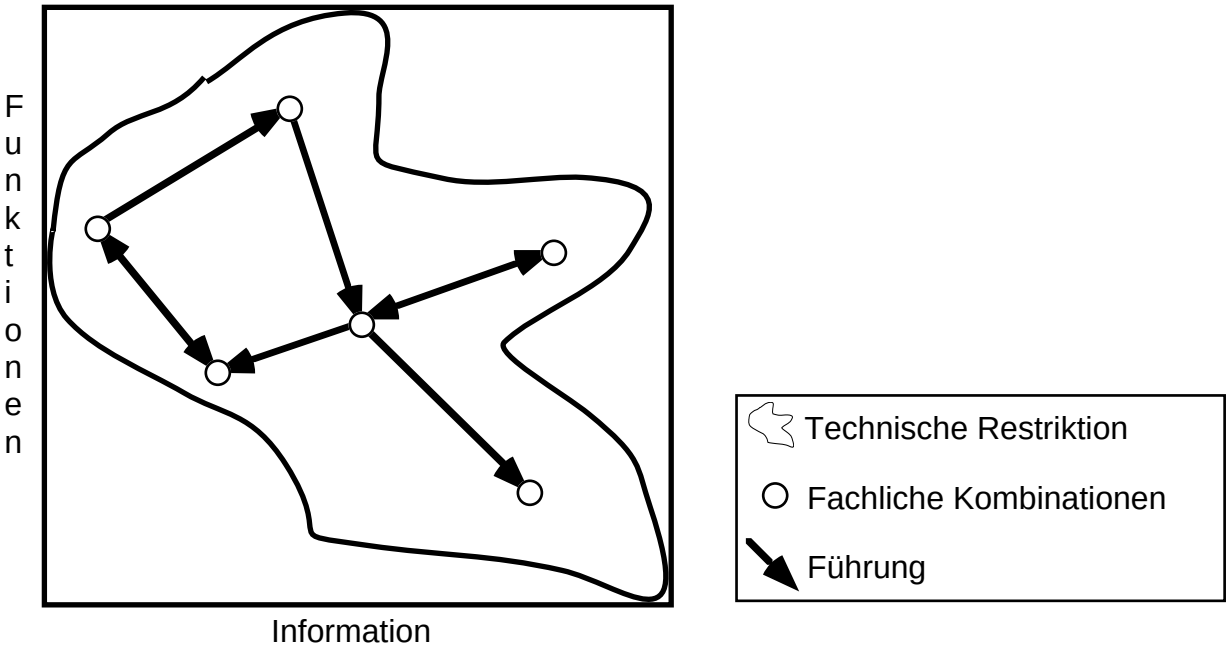

Abbildung 2.7: Führung durch Beschreibung der fachlichen Zusammenhänge

Führung soll als informative Führung verstanden werden. Der Ausgangspunkt der Führung ist die Beschreibung, welche fachlich sinnvollen Kombinationen von MSS-Informationen und MSS-Funktionen existieren und inwiefern sie für einen Anwendungskontext geeignet sind. Darauf aufbauend werden fachliche

Zusammenhänge zwischen den einzelnen Kombinationen aufgezeigt und gegebenfalls bewertet. Die MSS-Analysetätigkeit wird so durch die Dokumentation einer Abfolge von Analyseschritten unterstützt.
Es entstehen im Rahmen der Führung keine suggestiven Vorgaben, wie der Anwender sich zu verhalten hat, sondern er kann auf der Basis der verwalteten Bezüge für sich individuelle Handlungen ableiten.
Bspw. sollen die Auswirkungen einer Produktgruppe auf den Umsatz untersucht werden. Dazu wird der Anwender vom MSS geführt, indem er mit den Auswertungen und Anmerkungen versorgt wird, die mit der Produktgruppe verbunden sind. Die präsentierten Auswertungen kann der Anwender als Ausgangspunkt oder Grundbaustein seiner eigenen Analysen verwenden.

2.3.3 Anpassung

Anpassung ist ein zentrales Konzept für ein erfolgreiches MSS. Es ist eine Eigenschaft des Systems, durch die beschrieben wird, auf welche Art und Weise und bis zu welchem Grad sich die Strukturen und das Verhalten nach der Unterstützungssituation ausrichten können.
Der Bedarf nach Anpassung ergibt sich u.a. aus den folgenden Punkten:

- Durch die Dynamik im externen Umfeld und in internen Bereichen müssen kontinuierlich die Informationen und Funktionen angepaßt werden.
- Die individuellen Informationsbedürfnisse der Anwender variieren untereinander signifikant.
- Mit der Erfüllung von bestehenden Anforderungen entstehen neue. Das entspricht dem Stichwort: “users always want more” [RaWa95 S.151]).

Dazu kommt die Möglichkeit, die Zusammenhänge im System aktiv zu erforschen. Hierfür ist ein höherer Grad an Flexibilität bezüglich Zuständen und Strukturen notwendig, um systemgestaltende Handlungen zu unterstützen.
Bei dem im Abschnitt 2.1 beschriebenen internen Software-Qualitätskriterium der Wartbarkeit wurde das MSS - mit seinen Informationen und MSS-Funktionen - durch den Entwickler angepaßt. Dazu ergänzend soll hier die Anpassung des MSS hauptsächlich auf der Ebene eines Fachanwenders verstanden werden. Mit dem Unterstützungsziel, daß der Fachanwender das MSS selbst anpassen kann, wird berücksichtigt, daß er selbst den größten Beitrag bei der Anpassung leistet [Choo98 S.7]. In der folgenden Abbildung wird die Anpassung der Informations- und Funktionskombinationen und der Zusammenhänge grau dargestellt (siehe Abbildung 2.8).

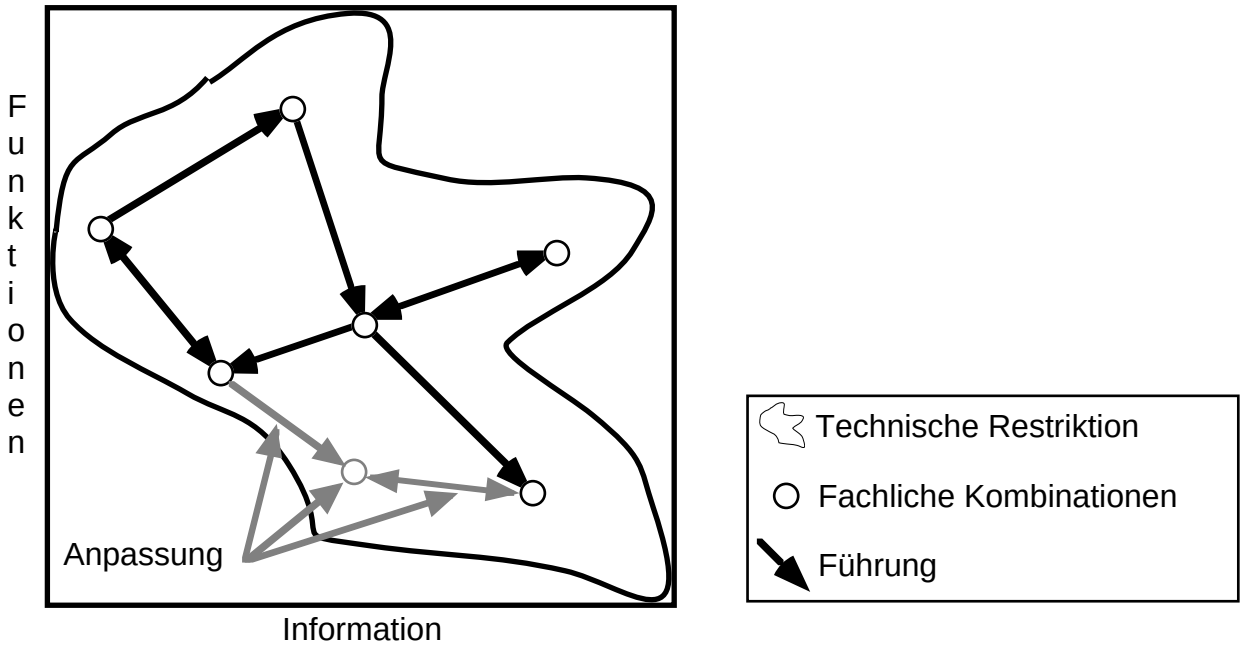

Abbildung 2.8: Anpassung der Kombinationen und Zusammenhänge

Bspw. erstellt der Fachanwender im Rahmen des MSS neue Auswertungen und ändert bestehende.
Die Grundlagen dafür sind aus dem MSS-Bereich die Beschränkung und die Führung. Durch die Beschränkung wird der Fachanwender davor bewahrt, das System so zu modifizieren und zu erweitern, daß die Korrektheit gefährdet wird. Es ist damit ein Bestandteil der Robustheit. Durch die Führung werden sinnvolle Möglichkeiten der Anpassung aufgezeigt. Allgemeine Grundlagen sind die Benutzerfreundlichkeit und die Wartbarkeit.

Nach der Spezifikation der grundlegenden Software-Qualitätskriterien, der Natürlichkeit und der speziellen MSS-Kriterien wird im folgenden Kapitel untersucht, welchen Beitrag die Objektorientierung für ein MSS leisten kann.

3. Beiträge des objektorientierten Paradigmas

In diesem Abschnitt wird betrachtet, welchen Beitrag das objektorientierte Paradigma zur Erfüllung der ermittelten Kriterien leisten kann.

Traditionell ist mit der Objektorientierung in erster Linie eine Effizienz im Sinne einer hohen Produktivität im Bereich der Software-System-Entwicklung verbunden [u.a. Grah93 S. 31f., CoNo91 S.26f.]. Davon ausgehend soll in dieser Arbeit als Schwerpunkt die Begründung für den Einsatz der Objektorientierung mit ihrem Beitrag für ein MSS-Konzept auf zwei Haupteigenschaften basieren, die in den folgenden Abschnitten beschrieben werden. Zunächst werden die objektorientierten Konzepte anhand ihrer Möglichkeiten, Komplexität zu bewältigen, dargestellt. Darauf aufbauend wird auf die natürliche Modellierungssichtweise der Objektorientierung eingegangen.
Da kein einheitliches objektorientiertes Paradigma bezüglich des Entwurfs von Software-Systemen existiert, wird stattdessen jeweils eine Entwurfssicht verwendet, die auf konkreten objektorientierten Konzepten basiert. Diese Sichtweise wird beeinflußt von dem zugrundeliegenden Entwicklungssystem und der dazugehörigen Programmiersprache [AlBW98 S.4]. Auch in diesem Bereich gilt, daß die Baumaterialien die Entwurfstechniken tiefgehend beeinflussen [CoMa96 S.1]. Als Basis für den Entwurf des objektorientierten MSS wird eine konkrete Sichtweise verwendet, die auf der Smalltalk-Entwicklungsumgebung basiert [GoRo89]. Aus dieser Sichtweise heraus werden im folgenden die objektorientierten Konzepte beschrieben und um für das MSS-Anwendungsgebiet relevante Smalltalk-Beispiele ergänzt.

3.1 Konstrukte zur Komplexitätsbewältigung

Das Problem der Komplexitätsbewältigung ist die entscheidende Aufgabenstellung bei der Entwicklung umfangreicher Softwaresysteme [u.a. Kreu90 S.212, Riel96 S.1-2]. Es ist notwendig im Rahmen der MSS Mechanismen für den effizienten Umgang mit komplexen Informationen bereitzustellen. In diesem Abschnitt werden daher die grundlegenden objektorientierten Konzepte auf ihren Beitrag zur Behandlung der Komplexität und der Erfüllung der Software-Qualitätskriterien untersucht.

3.1.1 Modulbildung

Ausgangspunkt der Komplexitätsbewältigung ist die Bildung von Software-Modulen [McCo93 S.774]. Der Problembereich des Software-Systems wird durch die Module in abgeschlossene Teilaufgaben aufgeteilt, um sie möglichst unabhängig voneinander zu lösen.
Bei der Modulbildung werden nach einem festgelegten Prinzip die Daten und Funktionen des Software-Systems aufgeteilt. Das Zerlegungsprinzip der Objektorientierung besteht darin, daß logisch zusammengehörige Daten und Funktionen einem Modul, dem Objekt, zugeordnet werden. Die Zusammengehörigkeit besteht darin, daß die Funktionen nur die Daten des eigenen Moduls verarbeiten können [WiWW90 S.65]. Ein Objekt besitzt durch seine Daten, die Attribute, einen Zustand und durch seine Funktionen, die Methoden, ein Verhalten. Objekte gehören zu einem Objekttypen, wenn sie die gleichen Methoden besitzen.

Eine reine objektorientierte Entwicklungsumgebung wie Smalltalk besteht einheitlich von einfachen Datentypen, wie Zahlen und Zeichen, bis hin zu komplexen Strukturen, wie kompletten Anwendungssystemen, aus Objekten. Dies führt zu sehr homogenen Anwendungssystemen, bei denen die Anwender alle Einheiten mit den gleichen Konzepten erfassen und auf die gleiche Weise behandeln können [RoAl90 S.355].

Die komplexen MSS-Sachverhalte können als Einheit durch Objekte direkt abgebildet werden. Diese semantische Möglichkeit, Objekte beliebig komplex zu strukturieren, basiert darauf, daß durch die Attributwerte der Objekte Bezüge auf beliebige andere Objekte verwaltet werden können. Zwischen den beteiligten Objekten existiert eine Assoziationsbeziehung. Ein MSS-Konzept kann mit allen seinen Daten und Funktionen direkt durch Objekte repräsentiert werden. Eine technisch bedingte Verteilung auf zusätzliche Einheiten, wie z.B. bei der relationalen Technik [Codd70], ist nicht nötig, da die Objektorientierung ausdrucksstark genug ist. Die Abbildung 3.1 veranschaulicht dies anhand eines Beispiels. Sie enthält die an einer Umsatz-Auswertung über Kunden und ihre Aufträge beteiligten Elemente, jeweils für die objektorientierten und die relationalen Strukturen. Bei der relationalen Variante sind gegenüber der objektorientierten zusätzlich als eigenständige Einheiten die Verbindung zwischen Kunden und Aufträgen und die Umsatzfunktion enthalten (siehe Abbildung 3.1). Durch die zusätzlichen Einheiten erhöht sich die Komplexität der modellierten MSS-Sachverhalte.

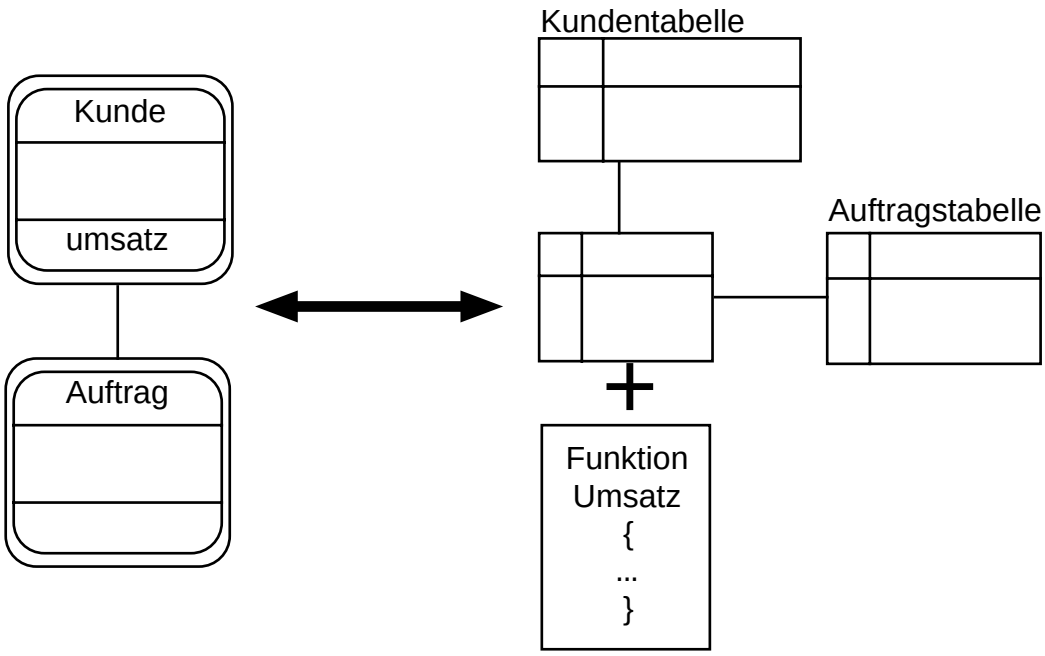

Abbildung 3.1: Objektorientierte vs. relationale Auswertungselemente (Objektnotation nach [CoYo94a])

Mit den Objekten lassen sich zur Laufzeit flexibel Software-Systeme aus Daten und Funktionen zusammenstellen. Wie flexibel ein Objektsystem ist, hängt von dem Grad der Typbindung ab. Bei einer starken Typbindung müssen die hinter den Attributen verwalteten Objekte zu einem bestimmten Objekttypen gehören. Bei einer schwachen Typbindung, wie sie vom Smalltalk-System unterstützt wird, gibt es diesbezüglich keine Einschränkung [AlBW98 S.6-7].
Naturgemäß läßt sich mit einer schwachen Typbindung ein System flexibler zur Laufzeit aus Objekten zusammenstellen als bei einer starken Typbindung. Durch diese Flexibilität wird einerseits die Möglichkeit geschaffen, heterogene Objektstrukturen zu verwalten, andererseits ist damit eine größere Verantwortung verbunden. Es muß explizit sichergestellt werden, daß die hinter einem Attribut verwalteten Objekte passende Eigenschaften besitzen.
Bspw. besteht der Vorteil einer schwachen Typbindung darin, daß Auswertungsobjekte prinzipiell auf beliebige Informationsobjekte angewendet werden können. Allerdings muß für jedes Informationsobjekt sichergestellt sein, daß es die für die Auswertung notwendigen Eigenschaften besitzt. Soll z.B. eine Informationsobjektmenge nach numerischen Eigenschaften aggregiert werden, dann ist dies nur möglich, wenn die Informationsobjekte zumindestens eine solche sinnvoll zu aggregierende Eigenschaft besitzen.
Die Flexibilität mit ihrer Möglichkeit, heterogene Objekte zu verwalten, ist damit die Basis für die Erfüllung des MSS-Kriteriums, einen variablen Strukturierungsgrad bereitzustellen und damit auch die notwendige Grundlage für die Integration von externen Informationen.

Die integrierte Verwaltung der Attribute, der dazugehörigen Methoden und der Abhängigkeiten zwischen Attributen und Methoden unterstützt die Wartbarkeit. Es können effizient Fehler lokalisiert werden, da sich einzelne Objekte testen lassen, und es ist einfacher im Rahmen eines Objekts Änderungen durchzuführen, ohne andere Bereiche des Software-Systems zwangsläufig zu beeinflussen.
Z.B. kann alternativ die Berechnungsvorschrift einer Auswertung als ein eigenständiges Objekt modelliert werden (siehe Abbildung 3.2). In diesem Fall enthält das Auswertungsobjekt zusätzlich als Attribut ein Aggregationsobjekt, und in der Methode wird nicht selbst die Berechnung durchgeführt, sondern an das Aggregationsobjekt die Aufgabe deligiert. Zusätzliche Anpassungen außerhalb des Auswertungsobjekts sind nicht nötig.

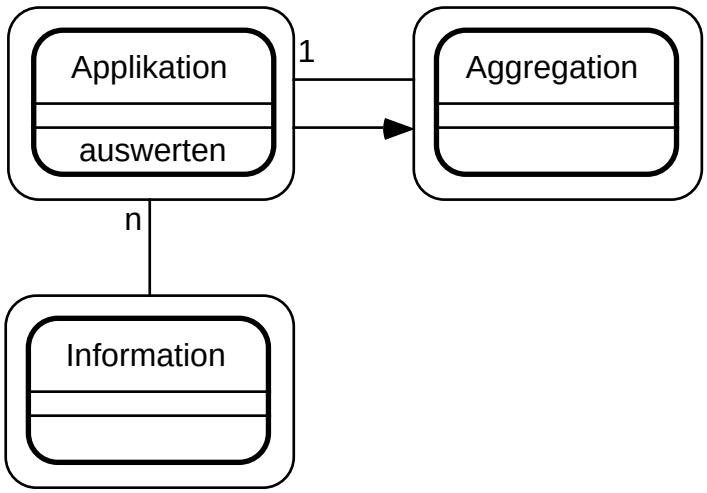

Abbildung 3.2: Auswertungsobjekt und Aggregationsobjekt (Objektnotation nach [CoYo94a])

3.1.2 Abstraktion

In der Objektorientierung repräsentieren die Objekte, aus denen sich das Software-System zusammensetzt, abstrakte Einheiten. Die Objekte sind untereinander nur durch ihr Verhalten, das sie zeigen können, und nicht dadurch, wie sie die mit dem Verhalten verbundenen Aufgaben konkret erfüllen, beschrieben.
Durch die Anwendung des Geheimnisprinzips nach Parnas erhält jedes Objekt eines Software-Systems von den übrigen Objekten nur so viel Informationen wie nötig, aber so wenig wie möglich [Parn72].
Die nötigen Informationen, um die Eigenschaften eines Objekts nutzen zu können, werden in Form einer Schnittstelle bekannt gegeben. Eine Schnittstelle besteht aus einer Menge von Protokollen, die jeweils durch eine Methode realisiert sind. Ein Protokoll besitzt in Smalltalk einen Methodenselektor, also die Bezeichnung der Methode, und die als Parameter übergebenen Objekte. Die Schnittstelle eines Objekts beschreibt, welche Methoden wie aktiviert werden können.

Die Komplexität des inneren Aufbaus eines Objekt, durch die die Eigenschaften realisiert wurden, wird durch die Schnittstelle verborgen. Attributstruktur, Attributwerte und Methodenaufbau sind außerhalb des Objekts nicht bekannt. Es kann darauf kein Bezug genommen werden.
Im Smalltalk-System bilden alle Methoden eines Objekts seine Schnittstelle, die systemübergreifend von allen anderen Objekten genutzt werden kann. Dies entspricht einer öffentlichen Schnittstelle. Mechanismen zur Unterstützung von privaten Schnittstellen von Objekttypen, die nur von Objekten des Typs genutzt werden können, sind nur rudimentär und proprietär, abhängig von der konkreten Entwicklungsumgebung, vorhanden [AlBW98 S.95].

Durch die Vermeidung der direkten Zugriffe auf die Eigenschaften des Objekts wird die Korrektheit, die Zuverlässigkeit und die Robustheit unterstützt.
Bspw. kann ein Auswertungsobjekt in seiner Berechnungsmethode zunächst überprüfen, ob die notwendige Grundlage der Berechnung - ein passender Modellausschnitt und eine Berechnungsvorschrift - spezifiziert ist. Trifft dies zu, dann kann im zweiten Schritt die eigentliche Berechnung durchgeführt werden.
Weil für einen Zugriff auf Attributwerte explizit Zugriffsmethoden eingerichtet werden müssen, kann über diese der Zugriff kontrolliert werden.
Sollen z.B. die Aggregationsfunktionen bzw. Kennzahlen in einer Auswertung auswechselbar sein, dann überprüft die das Attribut betreffende Zugriffsmethode, ob die neu zugeordneten Berechnungsobjekte eine für die Aufgabe passende Schnittstelle besitzen.

Für die Wartungsfreundlichkeit eines Software-Systems ist es von Bedeutung, daß lokale Änderungen an einem Objekt die Zuverlässigkeit des gesamten Software-Systems nicht gefährden, solange das Objekt die von ihm erwarteten Eigenschaften besitzt, die durch seine Schnittstelle beschrieben werden.
Bspw. kann ein Auswertungsobjekt durch die Aktivierung seiner Ergebnis-Methode wahlweise das Ergebnis aktuell berechnen oder mit einem vorberechneten Attributwert antworten. In dem Auswertungsobjekt kann lokal, je nach Anforderungen, zwischen den beiden Alternativen gewechselt werden.

3.1.3 Nachrichten

Die nachrichtenbasierte Kooperation der Objekte eines Software-Systems ist die Basis für die flexible Verteilung der Aufgaben auf die Objekte und die Wartungsfreundlichkeit des Software-Systems.
Die Objekte eines Software-Systems arbeiten zusammen, indem sie untereinander Nachrichten austauschen. Jede Verarbeitung in einem objektorientierten System wird durch das Versenden von Nachrichten initiiert. Der Sender einer Nachricht fordert den Empfänger auf, die mit der Nachricht spezifizierte Aufgabe

zu erfüllen. Der Empfänger ist dabei vollständig selbst verantwortlich festzulegen, wie er auf eine Nachricht reagiert. Die Ausführung der Aufgabe ist davon abhängig, wie er die Nachricht interpretiert und wie sein Zustand ist. Erst zur Laufzeit wird dynamisch festgelegt, welches konkrete Objekt die Nachricht zugesendet bekommt und wie, d.h. mit welcher Methode das Objekt die Aufgabe erfüllt.

Durch die nachrichtenbasierte Kooperation sind die Objekte untereinander unabhängiger verbunden. Dadurch ist es möglich, daß das Objekt, das die mit einer Nachricht verbundene Aufgabe ausführen soll, zur Laufzeit flexibel bestimmt wird. Es kann für eine Aufgabe am geeignetsten sein oder vom Anwender individuell ausgewählt werden.
Z.B. kann in einem MSS zur Laufzeit festgelegt werden, welches Auswertungsobjekt auf ein bestimmtes Objekt der Domäne angewandt wird. Abhängig von der Auswertungsart und speziellen Eigenschaften des Domänenobjekts wird im ersten Schritt ein passendes Auswertungsobjekt bestimmt und erst im zweiten Schritt die eigentliche Berechnung durchgeführt. Besitzt ein Domänenobjekt z.B. n-dimensional ausgeprägte Eigenschaften, dann ist eine OLAP-Auswertung über mehrere Dimensionen möglich und wünschenswert [CoCS93].

Die dem Software-System zugrundeliegenden Abläufe sind durch den Nachrichtenaustausch auf einer hohen Abstraktionsebene transparent. Damit ist es zusätzlich wartungsfreundlich. Für den Fall, daß ein Objekt nicht die passende Methode besitzt, reagiert es in Smalltalk dennoch auf kontrollierte Weise. Es sendet sich selbst eine Nachricht zu, die darauf hinweist, daß es die letzte Nachricht nicht verstanden hat (`#doesNotUnderstand:`). Der Entwickler kann für diesen Fall die vollständigen Nachrichtenabläufe zurückverfolgen, die zu diesem Ausnahmezustand geführt haben.

Die Möglichkeit, daß ein Objekt für sich alleine entscheidet, wie es auf eine Nachricht reagiert, ist eine wichtige Eigenschaft der Objektorientierung. Es kann dieselbe Nachricht an Objekte verschiedener Typen gesendet werden. In Abhängigkeit davon, wie die Methoden in den Objekttypen realisiert sind, zeigen unterschiedliche Objekte auf die gleiche Nachricht nach außen ein gleiches und im Inneren, unter Verwendung ihrer individuellen Eigenschaften, ein (abweichendes) spezialisiertes Verhalten. Dieser Polymorphismus (gr. Vielgestaltigkeit) vereinfacht die Struktur eines objektorientierten Software-Systems, weil die gleiche Methode in unterschiedlichen Objekttypen unterschiedlich realisiert werden kann. Eine Fallunterscheidung in den Methoden muß nicht vorgenommen werden. Dies vereinfacht die Struktur des MSS [Vett95 S.90].
Z.B. soll über eine Menge von Informationsobjekten eine numerische Eigenschaft, die sie alle besitzen, durch eine vom Anwender zur Laufzeit ausgewählte

Weise aggregiert werden. Ohne Polymorphismus hätte jedes Aggregationsobjekt abhängig davon, welche Aggregationsart es anbietet, eine abweichend benannte Methode (z.B. `berechneSumme`, `berechneDurchschnitt`, `berechneMax`, ...), die die Verarbeitung durchführt. Bei der Anwendung müßte in Abhängigkeit vom Aggregationsobjekt unterschieden werden, wie die passende Nachricht lautet. Mit Polymorphismus versteht jedes Aggregationsobjekt die gleiche Nachricht (z.B. `berechne`).

Durch den Polymorphismus wird zusätzlich die Erweiterbarkeit eines MSS unterstützt. Es kann auf einfache Art ein neuer Objektyp, z.B. eine Aggregationsart, hinzugefügt werden. Bestehende Strukturen des MSS müssen nicht geändert werden, das MSS muß nur um die neuen Strukturen ergänzt werden.

3.1.4 Hierarchien

Weitere komplexe und MSS-relevante Eigenschaften betreffen die Möglichkeiten der Objektorientierung, ein Software-System hierarchisch aufzuteilen. Mit der Bildung von Hierarchien werden verschiedene Details auf verschiedenen Ebenen behandelt [McCo93 S.775].
Die zwei grundsätzlichen Möglichkeiten der Objektorientierung ein System hierarchisch zu beschreiben sind:

Komposition
: Durch die Komposition wird zwischen den beteiligten Objekten eine hierarchische Teil-Ganz-Beziehung hergestellt, d.h. zusammengesetzte Objekte (engl. Composites) besitzen als Bestandteile/Komponenten andere Objekte. Komposition unterstützt die Komplexitätsbewältigung, indem durch sie die Übersichtlichkeit und Strukturierung komplexer Architekturen verbessert werden [PaSi94 S.205].
Ein Composite repräsentiert dabei als Einheit seine eigenen Eigenschaften und zusätzlich ausgewählte Eigenschaften seiner Komponenten. Bspw. besitzt ein Auswertungsobjekt als Bestandteile Informationsobjekte, Aggregationsobjekte und Darstellungsobjekte.
In Smalltalk existieren keine gesonderten Sprachkonstrukte für die Formulierung einer Komposition. Die Komposition wird deshalb wie eine Assoziation behandelt. Damit können die Eigenschaften eines Composites zur Laufzeit an die sich dynamisch ändernden Anforderungen angepaßt werden, indem einfach die Komponenten hinzugefügt und entfernt werden.

Vererbung

Durch die Vererbung werden entlang einer Hierarchie Ähnlichkeiten und Unterschiede von Objekten organisiert. Die Vererbung auf der Basis von Klassen ist dabei die verbreitetste Organisationsform[1] [Copl00 S.122]. Es wird zwischen Klassen, als abstrakte Beschreibungen einer Menge von Objekten, und Instanzen oder Exemplaren, als konkrete Objekte, die jeweils zu einer Klasse gehören, unterschieden. Die Beziehung zwischen Instanzen und ihren Klassen wird als „ist-ein“ oder „ist-Instanz-von“ bezeichnet.

Eine Klasse beschreibt die Gemeinsamkeiten ihrer Instanzen, wie den Aufbau der Attributstruktur und die Definitionen der Methoden. Eine Instanz beinhaltet die Unterschiede, die speziellen Werte der Attribute, durch die sie sich von den anderen Instanzen ihrer Klasse unterscheidet.

Weil in Smalltalk alle Elemente Objekte sind, gilt dies auch für Klassen. Sie besitzen als vollwertige Objekte Attribute und Methoden und werden durch Nachrichten aufgefordert, ein bestimmtes Verhalten zu zeigen, wie z.B. eine neue Instanz zu erzeugen.

Jedes Objekt besitzt somit einen Typ und eine Klasse, zu der es gehört. Ein Objekttyp, also die Beschreibung der Schnittstelle eines Objekts, kann dabei auf unterschiedliche Arten implementiert werden. Mehrere Klassen können den gleichen Objekttypen realisieren. Andererseits besitzt eine Instanz durch ihre Klasse als Einheit eine Realisierung und eine Schnittstelle [AlBW98 S.121].

Zwischen den Klassen existieren hierarchische Generalisierungs- und Spezialisierungsbeziehungen. Klassenhierarchien werden erzeugt, indem neue Klassen als Unterklasse einer bereits bestehenden Klasse definiert werden[2]. In einer Unterklasse können die geerbten Eigenschaften der Oberklasse durch zusätzliche Eigenschaften ergänzt und durch die Änderung geerbter Methoden das Verhalten auf Nachrichten verändert werden. Man bezeichnet die Methodenänderungen als Redefinition oder Überschreiben (engl. overriding). Ähnlichkeiten werden so entlang der Hierarchie möglichst weit oben, die Unterschiede möglichst tief beschrieben [Riel96 S.93].

Als Beispiel wird in einer Auswertungsklasse festgelegt, daß ihre Instanzen als Eigenschaften einen Modellausschnitt und eine Berechnungsvorschrift besitzen. Von der Auswertungsklasse wird eine Unterklasse abgeleitet. Sie definiert als zusätzliche Eigenschaft eine

[1]Eine Alternative zu der klassenbasierten Vererbung ist z.B. die Delegation, die in der Programmiersprache Self verwendet wird. Dabei wird entlang einer Hierarchie aus konkreten Objekten die mit einer Nachricht verbundene Aufgabe deligiert [SmMU95 S.56].

[2]Unterschieden wird dabei zwischen einer einfachen Vererbung, wenn eine Klasse nur eine Oberklasse besitzen kann, und einer mehrfachen Vererbung, bei mehreren Oberklassen.

Beschränkung und redefiniert die Berechnungsvorschrift durch die Berücksichtigung der zusätzlichen beschränkenden Eigenschaft.
Durch die Vererbung lassen sich neue Objekttypen, durch die die Elemente des Anwendungsbereich direkt abgebildet werden, erzeugen. Domänenkonzepte, wie z.B. Kunden, Produkte und Aufträge, werden als fester Bestandteil, gleichberechtigt zu den vorgegebenen Basiskonzepten, wie z.B. Zahlen, Zeichen und Listen, in die Entwicklungsumgebung integriert.

Beide Formen der Hierarchie unterstützen die Wiederverwendung von Softwareobjekten auf unterschiedliche Weise:

- Die Wiederverwendung durch Komposition wird als black-box-reuse bezeichnet [GHJV95 S.19]. Dieser Begriff bezieht sich auf die Unsichtbarkeit des inneren Aufbaus der Objekte, die an einer Kompositionshierarchie beteiligt sind. Der innere Aufbau seiner Komponenten bleibt einem Composite verborgen. Unter Einhaltung des Geheimnisprinzips erfolgen die Zugriffe auf die Komponenteneigenschaften ausschließlich über ihre Schnittstellen.
- Im Gegensatz dazu wird eine Vererbungshierarchie als ein white-box-reuse bezeichnet. Für die Unterklassen ist der innere Aufbau ihrer Oberklassen sichtbar. In den überschriebenen Methoden kann z.B. direkt Bezug auf ererbte Attribute genommen werden. Diese Verletzung des Geheimnisprinzips wirkt sich nachteilig auf die Wartungsfreundlichkeit aus, da durch die Änderung der Attributstruktur einer Oberklasse jede Methode der Unterklassen überprüft und gegebenfalls angepaßt werden muß [vgl. Snyd86].

Diese Probleme mit der Vererbung (Stichwort: “inheritance is the GOTO statement of the nineties”) führen zu der allgemeinen Empfehlung, die Komposition der Vererbung vorzuziehen [z.B. bei AlBW98 S.356, GHJV95 S.18-21]. Dennoch kann auf die Vererbung als wichtiger Bestandteil der Objektorientierung nicht verzichtet werden. Für den verantwortungsvollen Umgang mit dem Konzept sind Richtlinien zu beachten [CoMa96 S.59, Riel96 S.75-126].

Die Wiederverwendung ist ein zentraler Vorteil für den Einsatz der Objektorientierung [CoNo91 S.26f., Appe95 S.219-220]. Es können dabei Objekte in Software-Systemen eingesetzt werden, für die sie nicht primär entwickelt wurden. Durch die Wiederverwendung wird für den Kriterienbereich der Wartungsfreundlichkeit die Flexibilität unterstützt [Tayl92 S. 249], indem es möglich ist, vorgegebene Informationselemente einfach anzupassen und so mit einem geringen Aufwand die Funktionalität des Software-Systems zu erweitern.

Bestehende getestete Objekte lassen sich in neuen Zusammenhängen verwenden, was zu einer größeren Produktivität führt. Neu erstellte Objekte können mit einem geringen Aufwand, d.h. ohne weitere Anpassung des restlichen Software-Systems, integriert werden. Der Rückgriff auf Objekte mit bewährtem und getestetem Code erhöht die Korrektheit und die Zuverlässigkeit der Software-Systeme.
Durch die bei der Objektorientierung verwendeten Konstrukte Modulbildung, Abstraktion, Nachrichten und Hierarchien werden die für MSS relevanten Software-Qualitätskriterien in den Bereichen Korrektheit, Zuverlässigkeit, Robustheit und Wartungsfreundlichkeit unterstützt. Der Anwender wird, wie beim Natürlichkeitskriterium gefordert, durch die komplexere Repräsentation des Unternehmens und seines Umfelds unterstützt, indem die Komplexität im objektorientierten Modell effizient organisiert wird [Holu99, Grah93 S.32]. Damit wird der Anwender kognitiv entlastet, da das System für den Anwender die Komplexität verwaltet.

3.2 Objektorientierung als Basis für eine natürliche Modellierungssichtweise

“One of the biggest reasons for moving to the object-oriented paradigm for developing complex applications is that it allows designers more closely to model the real world.” [Riel96 S.2]

Mit den Konzepten zur Komplexitätsbewältigung ist noch nicht beschrieben worden, was effektiv als Objekte modelliert werden soll. Denkbar ist z.B. eine rein technische Sicht durch die Abbildung von Informationstechnologie-Datenstrukturen als Objekte. Vorteilhaft wäre dies lediglich in Bezug auf die effiziente Verwaltung von Abhängigkeiten zwischen den Objekten [Coat96].
Mit der Entwicklung der Objektorientierung ist allerdings eine bestimmte Sichtweise verbunden, durch die die fachliche Basis zur Verfügung gestellt wird. Mit der ersten objektorientierten Programmiersprache Simula wurden Simulationsmodelle erstellt, in denen Strukturen der realen Welt abstrahiert wurden [DaNy66]. Für das Modell relevante physikalische und abstrakte Elemente werden durch Software-Objekte repräsentiert, so daß das Simulationsmodell aus einer Gruppe von interagierenden Objekten besteht [WiGr88 S.196, Clau96 S.36].
Die Vorgehensweise zur Erstellung von Simulationsmodellen wurde auf andere Software-Projekte übertragen [Kreu90 S.213]. Diese Domänenorientierung ist eine semantische Eigenschaft, durch die beschrieben wird, welche Elemente repräsentiert werden [ReSu95 S.17]. Die grundlegende Sichtweise ist, daß Objekte Abstraktionen des Anwendungsgebiets sind, die für den Anwender von Interesse sind. Die folgende Aussage von Taylor unterstreicht dies: “The central activity in working with objects is not so much a matter of programming, as it is

representation...each important real world object or concept is represented by a sw-object and all the information and behaviour associated with that real world object are represented by the sw-object" [zitiert nach Hewi98 S.3].

In diesem Zusammenhang wurde für den Unternehmensbereich der Begriff des Geschäftsobjekts (engl. Businessobject) geprägt [Hewi98 S.3]. Die komplexen Strukturen sind so die Grundlage für die im Kriterienbereich geforderte passende Abstraktionsebene.
Die konsequente Weiterentwicklung des Konzepts der natürlichen Modellierung ist das Convergent Engineering [Tayl95]. Convergent Engineering ist die Synthese von Unternehmensgestaltung und Software-Engineering in einer vereinigten Disziplin. Das Ziel ist eine effiziente und effektive Gestaltung eines Konstrukts, das sowohl für den Anwender als auch für den Entwickler gleichermaßen sinnvoll und verständlich ist [Hube97]. Die Objektorientierung ist mit ihrer Möglichkeit, Domänenelemente als Software-Objekte direkt und natürlich abzubilden, die notwendige Grundlage. Durch das Objektmodell lassen sich die Strukturen und Vorgänge des Geschäftsbereichs sowohl simulieren als auch in der Realität umsetzen.

Mit der natürlichen Modellierungssichtweise erfüllt die Objektorientierung die in Abschnitt 2.2.2 beschriebenen Kriterien an das Informationsmodell. Daraus ergeben sich für die Anwender und Entwickler die dort dargestellten Vorteile aus den Bereichen Korrektheit, Benutzer- und Wartungsfreundlichkeit.

3.3 Design Patterns zur Erhöhung der Flexibilität

Der Einsatz der objektorientierten Konzepte führt nicht zwangsläufig zu „guten" Software-System-Architekturen im Sinne der Software-Qualitätskriterien. Für jede Entwicklungsphase, von Analyse über Design zur Programmierung sind Entscheidungen auf der jeweiligen Abstraktionsstufe zwischen verschiedenen Alternativen zu treffen.
Im Design-Bereich, mit der Überführung der Analyseergebnisse in konkrete Software-System, gab es bis zur Mitte der neunziger Jahre methodisch erhebliche Defizite. Die damals verwendeten Entwurfsmethoden zur objektorientierten Systementwicklung, z.B. von Coad/Yourdon, beschrieben ein sehr vages, allgemeines Vorgehen, sehr dicht am Analyse-Bereich in Verbindung mit Design-Details, die dicht am Programmierungsbereich angesiedelt waren [CoYo94a, CoYo94b, CoNi94, vgl. Schä94 S.158f.].

Der Mittelbau mit der Bildung von konkreten Designstrukturen fehlte. Auch waren die in der Literatur verwendeten Beispiele diesbezüglich kaum hilfreich. Die wenigen verwendeten Beispiele konzentrierten sich auf technische Anwendungsbereiche, wie z.B. Bankautomaten. Wegen dieses begrenzten Anwendungsspektrums ist die Verwendung der beschriebenen Design-Strukturen für eine MSS-Domäne nur in einem sehr geringen Ausmaß möglich.

Für das Problem des qualitativen Design lieferte ein Informationstechnologie-externer Bereich einen Unterstützungsbeitrag. Im Gebiet der Gebäudearchitektur hat der Architekt Christopher Alexander durch Muster (engl. Patterns) die Qualität einer Entscheidung beschrieben [AIS+77 S.x]
Ein Pattern beinhaltet ein wiederholt auftretendes Problem, das eine Entscheidung notwendig macht. Obwohl es für eine Problemklasse wiederholt angewendet wird, sind die Ergebnisse unterschiedlich und nur der Kern der Lösung ist identisch.
Die Patterns werden in einem Katalog verwaltet. Jeder Eintrag besitzt als essentielle Bestandteile einen Namen, das immer wiederkehrende Entwurfsproblem, den Kontext, in dem das Problem auftritt, ein generisches Schema für die Lösung und die daraus entstehenden Konsequenzen [GHJV95 S.3 (im Detail)].
Mit der Erkenntnis, daß auch in dem Bereich des Software-Engineering wiederholt Probleme auftreten, wurden die Muster in diesen Bereich übertragen. Im Bereich der objektorientierten Software-Technik existieren Patterns, angelehnt an die Entwicklungsphasen, auf verschiedenen Abstraktionstufen. Z.B. beschreibt Fowler Analyse Patterns, Gamma, Helm, Johnson und Vlissides Design Patterns und Beck Code Patterns [Fowl97, GHJV95, Beck97].

Für diese Arbeit werden Patterns schwerpunktmäßig auf derjenigen Abstraktionsebene angewandt, wo sie als Vokabular zur Beschreibung von Entwurfsentscheidungen dienen. Die Design Patterns (dt. Entwurfsmuster) beschreiben Probleme und Lösungen, die mit wenigen kooperierenden Klassen realisiert werden können. Sie werden häufig als Mikro-Architekturen bezeichnet, da sie die Bestandteile der Gesamtarchitektur eines Systems sind [Gamm96 S.20].
Die einzelnen Entwurfsmuster werden zur besseren Übersicht in drei grundlegende Anwendungsbereiche unterteilt: Erzeugende Muster (Creational Patterns), strukturelle Muster (Structural Patterns) und Verhaltensmuster (Behavioral Patterns) [GHJV95 S.10]. Jeder Anwendungsbereich wird im folgenden mit einem MSS-relevanten Beispiel vorgestellt.

Erzeugende Muster abstrahieren die Erzeugung von Instanzen. Ein Beispiel für diese Gruppe ist das Singleton-Pattern. Mit ihm wird sichergestellt, daß von einer Klasse genau eine Instanz erzeugt wird und daß auf diese Instanz im gesamten System zugegriffen werden kann.

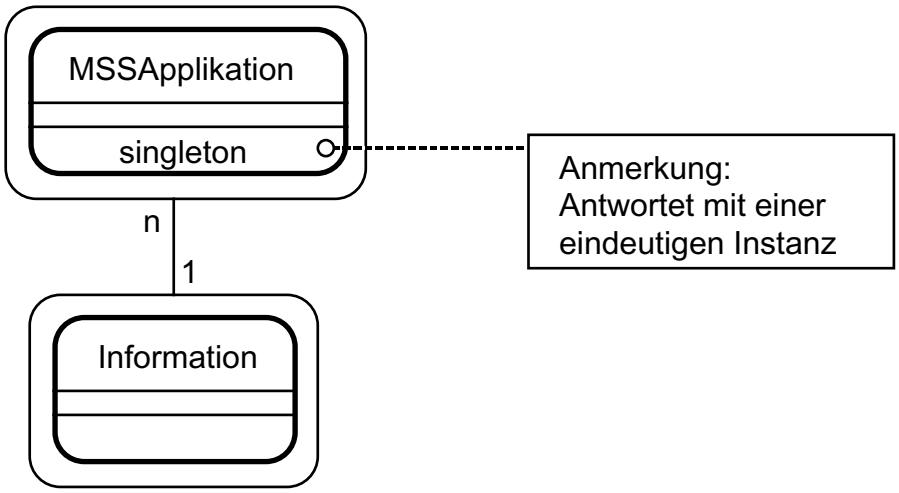

Abbildung 3.3: Beispiel eines Singleton-Pattern
(als Objektmodell [GHJV95 S.127], Notation in Anlehnung an [CoYo94a])

Ein MSS besteht z.B. genau aus einem Applikationsobjekt, durch das alle Informationsobjekte verwaltet werden (siehe Abbildung 3.3). Für die Überprüfung, ob ein bestimmtes Informationsobjekt im MSS existiert, wird mit dem Singleton-Pattern der Zugriff auf das einzelne Applikationsobjekt bereitgestellt, das diese Anfrage bearbeiten kann.

Strukturelle Muster befassen sich damit, wie Klassen und Instanzen kombiniert werden, um größere Strukturen zu bilden. Durch das strukturelle Facade-Pattern wird z.B. eine einheitliche Schnittstelle für ein Subsystem bereitgestellt, durch das das Teilsystem auf einer höheren Abstraktionsebene einfacher verwendet werden kann (siehe Abbildung 3.4).

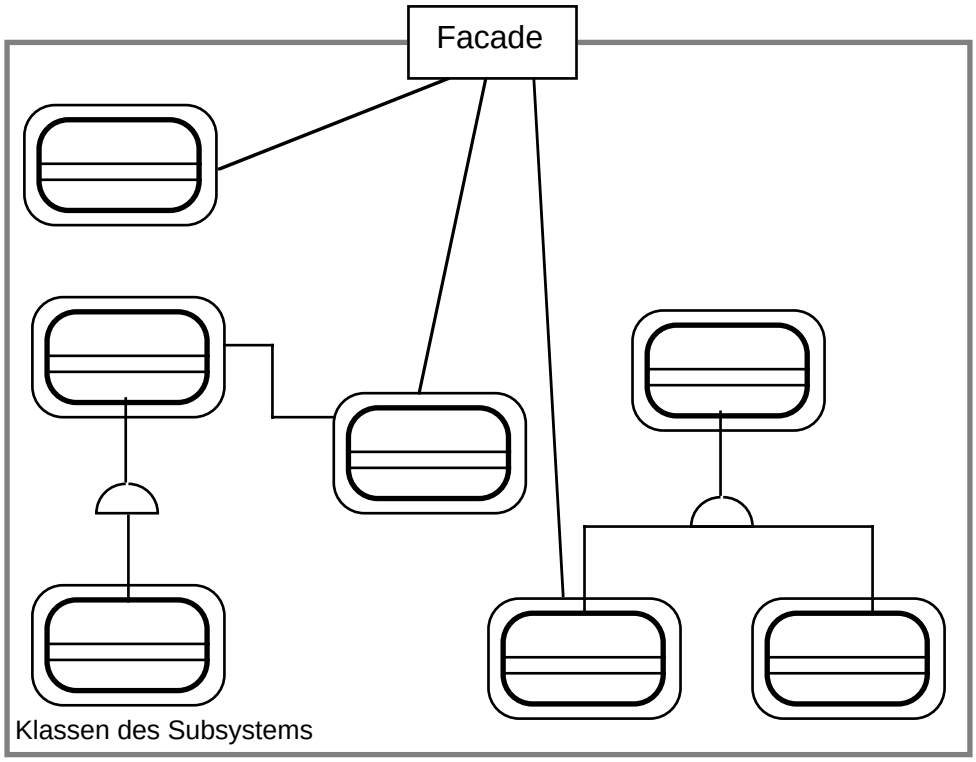

Abbildung 3.4: Beispiel eines Facade-Pattern
(als Schema [GHJV95 S.187], Notation in Anlehnung an [CoYo94a])

Mit dem Facade Pattern lassen sich die für ein MSS notwendigen externen Informationsquellen über eine zentrale Schnittstelle anbinden. Das MSS kann so einheitlich und konsistent über ein Facade die verschiedenen Quellen nutzen.

Zusätzlich werden im Rahmen der Arbeit als strukturelle Pattern Adapter und Composite verwendet.
Ein Adapter-Pattern wird dann eingesetzt, wenn in einem System zwei Objekte als Client und Server zusammenarbeiten sollen, obwohl sie keine zueinander kompatiblen Schnittstellen besitzen. Die beiden Objekte kommunizieren bei diesem Pattern über ein zusätzliches Adapter-Objekt, das die Nachrichten geeignet umsetzt. Die Zusammenarbeit der Client- und Server-Objekte wird so möglich, ohne daß die Objekte direkt angepaßt werden müssen [GHJV95 S.139].
Mit dem Composite-Pattern wird die in Abschnitt 3.1.4 beschriebene Kompositionsbeziehung in Pattern-Form dokumentiert. Zusätzlich wird durch das Composite festgelegt, daß einzelne und Mengen von Objekten gleich behandelt werden können, d.h. die gleichen Protokolle besitzen [GHJV95 S.163].

Verhaltensmuster befassen sich mit Algorithmen und der Zuweisung von Verantwortlichkeiten zwischen den Objekten. Sie beschreiben die Kommunikation zwischen den Objekten zur Laufzeit. Ein Beispiel für diese Gruppe ist das Visitor-Pattern. Ein Visitor repräsentiert eine Operation, die auf beliebigen Elementen einer Objektstruktur angewendet werden kann, ohne daß die Elemente einen direkten Bezug zu der Operation besitzen.
In dem hier verwendeten Beispiel werden über das Visitor-Pattern für die in einer MSS-Anwendung enthaltenen Informationsobjekte verschiedene Speicher- und Repräsentationsformen als Visitor-Klassen, wie z.B. die einfache Speicherung in einer Datei, die Verwaltung in einer relationalen Struktur oder die Repräsentation in einem HTML-Dokument, konzipiert (siehe Abbildung A.1 im Anhang).

Spezifische Eigenschaften, die sich aus der Kombination der Informations- und Speicher-/Repräsentationsform ergeben, werden so effizient berücksichtigt. Z.B. werden bei der Speicherung einer Instanz der Klasse `ToolObject`, durch die ein Bestandteil einer Auswertung abgebildet wird, auf eine vollständige Speicherung der Attributwerte verzichtet, indem der Attributwert für die Ergebnismenge ausgelassen wird. Bei Domänenobjekten könnte z.B. eine Metapher verbunden sein, die bei einer HTML-Präsentation als grafischer Bestandteil verwendet wird.

Um den Nachrichtenablauf bei der Anwendung eines Visitor-Pattern darzustellen, soll für ein konkretes Beispiel der Zustand eines Domänenobjekts in einer ASCII-Datei gespeichert werden. Die beteiligten Klassen und der Nachrichtenfluß werden durch das folgende Interaktionsdiagramm veranschaulicht (siehe Abbildung 3.5).

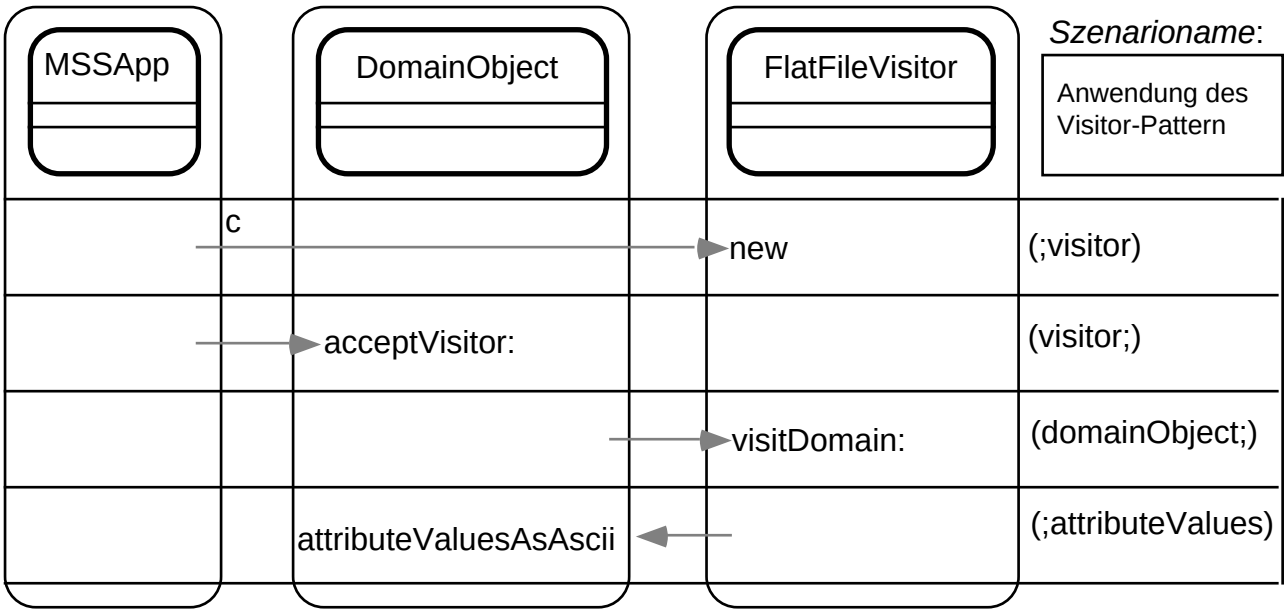

Abbildung 3.5: Szenario für die Anwendung des Visitor-Pattern
(Notation in Anlehnung an [AlBW98 S.377])

Der im Interaktionsdiagramm visualisierte Ablauf beschreibt die folgende Vorgehensweise:

- Im ersten Schritt wird ein passendes Visitor-Objekt erzeugt, das dies leistet (`new`).
- Im zweiten Schritt wird das konkrete Objekt mit der Nachricht `acceptVisitor:` aufgefordert, die Dienstleistung des Visitors in Anspruch zu nehmen.
- Das konkrete Objekt kennt das Protokoll, das alle Visitor verstehen (im Bsp `visitDomain:`). Auf die im dritten Schritt versendete Anfrage `visitDomain:` reagiert der FlatFileVisitor, indem er den Absender im vierten Schritt durch `attributeValuesAsAscii` auffordert, seinen Zustand als Ascii-Zeichenkette zu übermitteln. Diese Zeichenkette benötigt er, um das Domänenobjekt in einer Datei zu speichern.

In den Visitor-Klassen müssen dazu weder eine Fallunterscheidung gemacht werden, noch ist die Speicher- und Repräsentationsfunktionalität über die Hierarchie der Informationsobjektklassen verstreut, sondern zentral in einem Visitor untergebracht. Neue MSS-Speicher- und Repräsentationsformen lassen sich einfach ergänzen, ohne daß die bestehenden Informationsobjektklassen angepaßt werden müssen.

Ein weiteres wichtiges Pattern für MSS ist das in der Arbeit verwendete Interpreter-Pattern, auf dem die einfache Abfragesprache des objektorientierten MSS basiert. Durch dieses Verhaltensmuster wird die Repräsentation und die Ausführung der Abfragen definiert [GHJV95 S.243].

An den behandelten Anwendungsbeispielen ist erkennbar, daß Entwurfsmuster sich auf einer gemeinsamen Abstraktionsstufe befinden, dabei allerdings unterschiedlich komplex sind.
Allgemein läßt sich feststellen, daß es durch die Verwendung der Muster möglich wird, flexiblere und wiederverwendbare Entwürfe zu erstellen. Basis sind zusätzlich eingeführte indirekte Bezüge (Indirektionen) und die Zerlegung von Abstraktionseinheiten in Untereinheiten. Dies kann, übermäßig eingesetzt, zu einem gesteigerten Maß an unnötiger Komplexität führen. Die Richtlinie zum Einsatz von Entwurfsmuster ist deshalb, das System nur so flexibel wie nötig zu gestalten [GHJV95 S.31, Gamm96 S.20].

Ergänzend zur Flexibilität unterstützen die Muster die Wartungsfreundlichkeit durch die auf der Design-Ebene kommunizierbare Systemarchitektur. Mit dem Pattern-Vokabular besitzt der Entwickler eine einfache und dennoch mächtige Möglichkeit, ein System zu dokumentieren [Beck97 S.16].

4. Das objektorientierte MSS-Gesamtkonzept

4.1 Übersicht

Auf der Basis des Kriterienkatalogs und den in MSS Praxisprojekten gemachten Erfahrungen wird die drei-Ebenen-Architektur des objektorientierten Management Support Systems (ooMSS) vorgeschlagen. Das ooMSS besitzt als Kernbereiche ein Informations- und ein Interaktionsmodell. Zusätzlich wird das ooMSS in- und extern um einen Informationsbereich ergänzt. Ein wichtiger Bestandteil des Informationsmodells ist das MSS-Modell, das auf einer Meta-Ebene realisiert ist. Die Meta-Ebene ermöglicht auf einer höheren Ebene die Beschreibung und Verwaltung der speziellen MSS-Eigenschaften, die für das ooMSS bedeutsam sind. Die Bewältigung des mit den MSS-Eigenschaften verbundenen gesteigerten Komplexitätsgrades wird so, im Sinne eines knowledge-levels nach Fowler [Fowl97 S.4], mit einer separaten Meta-Ebene unterstützt (siehe Abbildung 4.1).

Das Informationsmodell steht im Zentrum des ooMSS. Es besteht aus Informationsobjekten, die mit ihren operativen und analytischen Eigenchaften die Basis der gesamten fachlichen Funktionalität des ooMSS bilden.
Das Informationsmodell verwaltet und verarbeitet komplexe Informationsobjekte unter Beachtung der inneren und äußeren Software-Qualitätskriterien. Die Informationsobjekte sind die Basis für die Bereitstellung einer passenden Abstraktionsebene und natürlicher Strukturen mit heterogenen, strukturierten und unstrukturierten, quantitativen und qualitativen Informationsinhalten.
Zur Erfüllung der Kriterien besitzen die Informationsobjekte ergänzend zu der verwendeten objektorientierte Basistechnologie erweiterte Eigenschaften in den Bereichen Selbstbeschreibung, Verwaltung und Zugriff auf die Eigenschaften der Informationsobjekte und Verwaltung von Informationsobjektmengen.
Neben dem operativen Domänenmodell besteht das Informationsmodell zusätzlich aus dem Werkzeugmodell, durch das die analytischen Beziehungen und Auswertungen modelliert werden. Die aus dem Kriterienbereich abgeleiteten Anforderungen an das Werkzeugmodell sind:

- Das Werkzeugmodell soll zur Laufzeit auf eine möglichst einfache benutzerfreundliche Art angewendet werden können.
- Auf die Verwendung einer Programmiersprache bei der Anwendung wird bewußt verzichtet, so daß die Beziehungen und Auswertungen auch durch einen Fachanwender formulierbar sind.
- Das Werkzeugmodell soll in seinem Kernbereich typische analytische Funktionalitäten des MSS-Bereichs enthalten und für einen möglichst großen Deckungsgrad bzgl. MSS-funktionaler Anforderungen wartungsfreundlich erweiterbar sein.

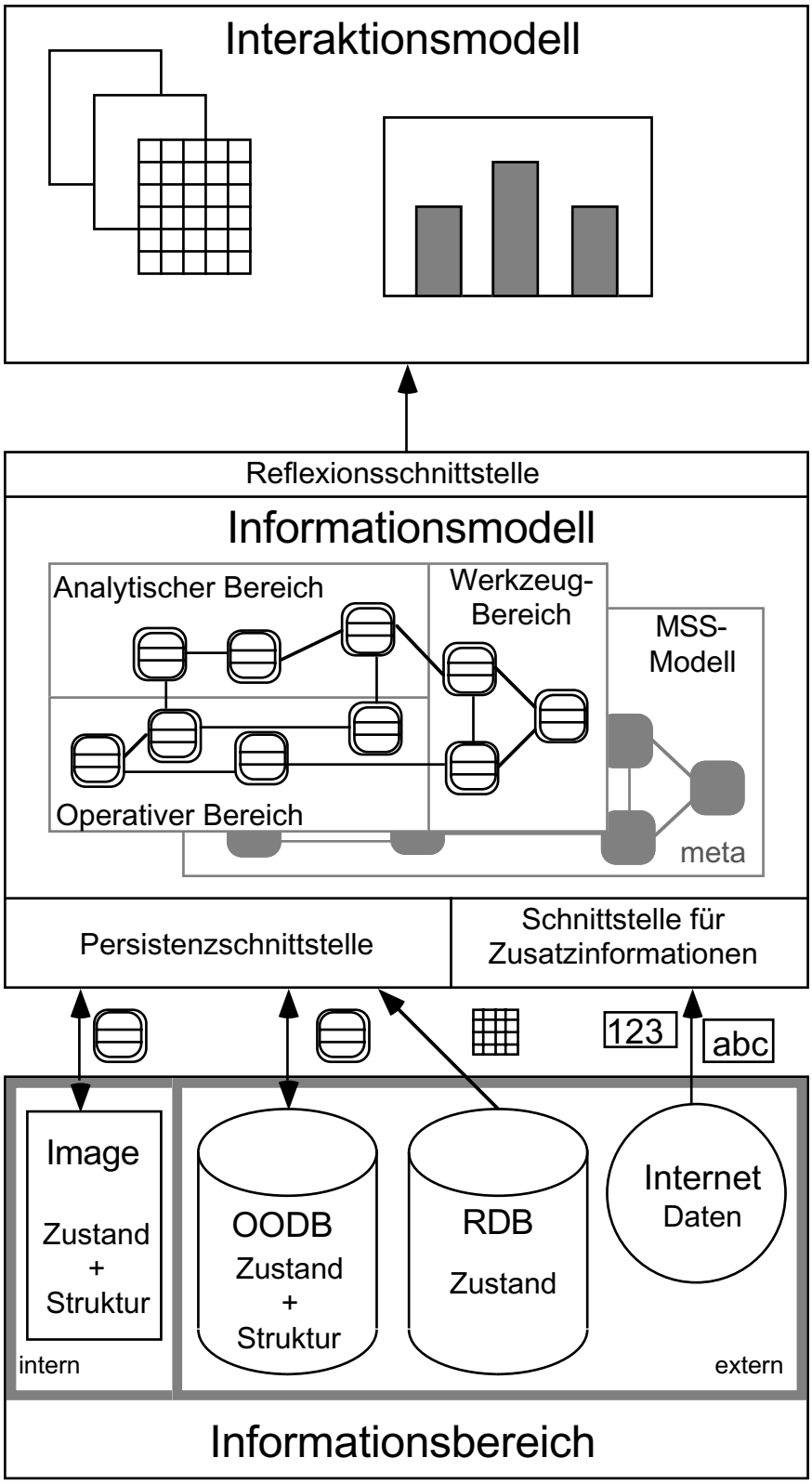

Abbildung 4.1: ooMSS-Gesamtkonzept

Das Informationsmodell deckt durch seinen MSS-Modellbereich die MSS-spezifischen Kriterien

- Restriktion,
- Führung und
- Anpassung

ab. Durch technische Aspekte beschränkt das MSS-Modell den Zustand des operativen Domänenmodells. Mit fachlichen Aspekten werden die Beziehungen zwischen den Elementen des Domänenmodells und des Werkzeugmodells dokumentiert. Darauf aufbauend wird der Anwender durch Kombinationen von fachlichen Aspekten in Form von Auswertungen und durch individuelle Anmerkungen informativ geführt. Die beschränkenden und führenden Aspekte sind die Basis für die Anpassung durch den Anwender. Das MSS-Modell befindet sich dabei auf der Meta-Ebene, durch die anwendungsspezifische Sichten ermöglicht werden.

Das Interaktionsmodell beschreibt die Schnittstelle zwischen dem Anwender und dem Informationsmodell des ooMSS (siehe oberen Bereich von Abbildung 4.1). Es vermittelt dem Anwender adäquat, d.h. homogen, konsistent, wahlfrei und natürlich die aktuellen komplexen Strukturen, Zustände und Funktionen des Informationsmodells.

Für die Anwendung des ooMSS bei einer Unterstützungssituation sind beide Modellbereiche beteiligt. Ein Ausschnitt des Informationsmodells, der sich aus einer Menge von Informationsobjekten und ausgewählter Eigenschaften zusammensetzt, wird dem Anwender in Form einer Umgebung, die aus Interaktionsobjekten besteht, bereitgestellt.

Versorgt wird das ooMSS mit Informationen durch den Informationsbereich. In diesem im unteren Drittel von Abbildung 4.1 dargestellten Bereich können Informationsobjekte auf verschiedene Arten persistent verwaltet werden. Zusätzlich repräsentiert er externe heterogene Informationsquellen, die sich im ooMSS nutzen lassen. Die Versorgung mit Informationen aus relationalen und objektorientierten Datenbanken wird ausführlich von [Maur00] behandelt und ist deshalb nicht im Detail Gegenstand der Arbeit.
Nachfolgend werden wesentliche Entwurfsentscheidungen für dieses Gesamtkonzept begründet:

4.2 Begründung

4.2.1 Trennung der ooMSS Modellbereiche

Ein charakteristisches Entwurfskriterium für das ooMSS ist die weitgehende Entkopplung von Informations- und Interaktionsmodell. Ausgangspunkt des Konzepts ist die Trennung zwischen dem Modellbereich und dem GUI-Bereich bei einer Applikationsarchitektur nach dem klassischen Model-View-Controller-Konzept [KrPo88].
Entkoppelt werden die beiden Applikationsbereiche durch die Verwendung eines Benachrichtigungsprotokolls. Die Modellobjekte, also Instanzen, verwalten die GUI-Objekte, durch die sie repräsentiert werden. Kommt es zu einer Änderung eines Modellobjekts, benachrichtigt es alle seine GUI-Objekte, durch welche es repräsentiert wird. Die GUI-Objekte bekommen damit die Möglichkeit, sich bei ihren Modellobjekten nach den Änderungen zu erkundigen und sich selbst zu aktualisieren.
Die Vorteile dieses Ansatzes einer Applikationsarchitektur liegen im Bereich der Wiederverwendung und Flexibilität.

- Die GUI-Objekte sind dabei nicht auf einen bestimmten Objekttyp spezialisiert, sondern lassen sich durch Parameter anpassen und für unterschiedliche Modelle (wieder-)verwenden. Weiterhin ist es möglich, dasselbe Modellobjekt gleichzeitig durch unterschiedliche Arten von GUI-Objekten zu repräsentieren.
- Die Anwendungen lassen sich flexibel um zusätzliche GUI-Objekte erweitern, ohne daß die bestehenden Modellstrukturen angepaßt werden müssen.
- Die Modelle lassen sich auch unabhängig, d.h. ohne eine GUI-Repräsentation, nutzen.

Nachteilig am Konzept ist die Notwendigkeit, gegenseitige Verweise zu verwalten. Die gesamte Komplexität des Software-Systems steigt wegen dieser Kopplung bei zunehmender Komplexität der beiden Bereiche überproportional. In der Praxis führt dies u.a. dazu, daß beim Model-View-Controller-Konzept wiederholt überflüssige Nachrichten versendet werden und sogar die Gefahr besteht, daß zirkulär Nachrichten versendet werden [Parc95 S.242 f.]. Hinzu kommt, daß die Trennung beim Model-View-Controller-Konzept für die Persistenthaltung der Informationsobjekte nicht konsequent genug ist. Die für die Benachrichtigung bei Modelländerungen notwendige Verbindung von den Modellobjekten zu den GUI-Objekten ist für eine Speicherung der Informationsobjekte ein Nachteil. Da es nicht sinnvoll ist, die GUI-Objekte permanent zu speichern, dürfen konsequenterweise keine Bezüge von den Infomationsobjekten zu ihren GUI-Objekten bestehen [Howa95 S.13-14].

Eine geringere Kopplung und Komplexität wird im Konzept des ooMSS durch eine einseitige Beziehung von den GUI-Objekten zu den Informationsobjekten gelöst. Die Kommunikation der beiden Bereiche wird von der Seite des Interaktionsmodells aus initiert. Das Interaktionsmodell verwendet für den Zugriff auf das Informationsmodell die Reflexionsschnittstelle, über die sich dieses mit seinen Objekten selbst beschreibt.

4.2.2 Persistenz und Anbindung von externen Systemen

Für den konzeptionellen Schwerpunkt des ooMSS, die Verarbeitung und die Repräsentation der Informationsobjekte, ist in einem ersten Schritt der intern im Smalltalk-Entwicklungssystem genutzte Persistenzmechanismus ausreichend. Das Smalltalk-Image ermöglicht auf einfache Art eine lokale Speicherung der kompletten Informationsobjekte mit ihren Strukturen und Zuständen. Im zweiten Schritt ist die Auslagerung der Informationsobjekte in externe Datenbanken wegen der größeren Speicherkapazitäten und der bereitgestellten Datenbank-Funktionalitäten, wie z.B. Mehrbenutzerbetrieb, Transaktionen und Zugriffskontrolle, notwendig [Howa95 S.13, Maur00 S.146-148].
Mit der Verwendung einer objektorientierten Datenbank muß dabei die Objektwelt nicht verlassen werden [Heue97]. Durch sie werden die Informationsobjekte als Einheit von Struktur, Zustand und Verhalten verwaltet. In einer objektorientierten Datenbank können die Objekte auch außerhalb des ooMSS ein Verhalten zeigen. Objektorientierte Datenbanken sind z.B. Bestandteil einer Object-Warehouse-Architektur [Maur00].

Das ooMSS Konzept ermöglicht einen möglichst nahtlosen Wechsel von der ooMSS-internen Verwaltung der Informationsobjekte zu den externen objektorientierten Datenbanken. Dazu muß das ooMSS eine Schnittstelle bereitstellen. Hierüber können alle unterstützten Persistenzmechanismen einheitlich genutzt werden, indem von der konkret genutzten Persistenzalternative abstrahiert wird. Externe Zusatzinformationen werden ebenfalls durch die Verwendung von Schnittstellen in das Informationsmodell integriert. Unterschieden werden dabei zwei grundsätzliche Arten von unterstützten Informationsquellen:

- Relationale Datenbanken als verbreitetste Art von Datenbanken ermöglichen die externe Verwaltung von strukturierten Informationen. Die Informationen können als Zustände von Informationsobjekten im ooMSS verwendet werden [Mösc96].
- Das Internet kann das ooMSS schwerpunktmäßig mit externen, unstrukturierten und qualitativen Informationen versorgen. Im Informationsmodell werden dazu die heterogenen Informationsquellen beschrieben, auf die über das Internet offen zugegriffen werden kann.

Das Konzept des ooMSS wird in den folgenden Abschnitten anhand der drei Bereiche Informationsmodell, Interaktionsmodell und MSS-Modell im Detail behandelt.

5. Informationsmodell

5.1 Grundeigenschaften des Informationsmodells

Als Basisbaustein des Informationsmodells, sowohl des operativen Domänenmodells als auch des analytischen Bereichs, werden die Informationsobjekte (Klasse `InformationObject`) verwendet. Nachfolgend wird die mit jeder Eigenschaft eines Informationsobjekts verbundene Schnittstelle jeweils mit den einzelnen Protokollen durch Anwendungsbeispiele dokumentiert.

5.1.1 Die Selbstbeschreibung der Informationsobjekte

Wie im Abschnitt „Beiträge des objektorientierten Paradigmas“ beschrieben wurde, ermöglicht das dem ooMSS zugrundeliegende Smalltalk-System inhärent den Umgang mit heterogenen Objektstrukturen. Für den Zugriff auf die Informationsobjekte, z.B. im Rahmen einer Interaktion oder einer Verarbeitung, ist die Kenntnis über die konkreten Eigenschaften der beteiligten Objekte notwendig. Zu diesem Zweck nutzen die Informationsobjekte als Eigenschaft die Selbstbeschreibung (engl. Reflexion). Dem Prinzip von Rivard folgend können die Objekte zur Laufzeit zu ihren speziellen Eigenschaften Auskunft geben [Riva96]. Da die strukturellen Eigenschaften für die konkreten Objekte einer Klasse identisch sind, wird die Selbstbeschreibungsschnittstelle standardmäßig auf der Klassenebene bereitgestellt.

Ausgangspunkt der für das Informationsmodell konzipierten Selbstbeschreibungsschnittstelle ist die Dokumentation, welche Eigenschaften ein Informationsobjekt in Form von Attributen und Methoden besitzt. Eine Anwendung des Protokolls ist allgemein und für die Beispielklasse `Kunde`, als Unterklasse von `InformationObject`, wie folgt ausgeprägt:

```
(InformationObject hasAttribut: #attributeName)
=> antwortet mit einem boolschen Wert

(InformationObject hasMethod: #methodName)
=> antwortet mit einem boolschen Wert

(Kunde hasAttribut: #name)
=> antwortet mit einem boolschen Wert

(Kunde hasMethod: #umsatz)
=> antwortet mit einem boolschen Wert
```

Für die Selbstbeschreibungsschnittstelle der Klasse `InformationObject` werden die vom Smalltalk-System standardmäßig vorgegebenen Klasseneigenschaften zur Reflexion genutzt [Riva96]. Das dort beschriebene Protokoll wird unter Verwendung eines Adapter-Pattern [GHJV95 S.139] für die Nutzung im ooMSS vereinfacht und vereinheitlicht. Die Verwendung des Adapters unterstützt die Wartungsfreundlichkeit des ooMSS, da nur an einer Stelle im System Bezug auf die konkreten Eigenschaften des zugrundeliegenden Entwicklungssystems genommen wird, und weil hinter dem Adapter eine effizientere Realisierung der notwendigen Selbstbeschreibungseigenschaften möglich ist, ohne daß das restliche ooMSS angepaßt werden muß.

Der zweite Bestandteil der Selbstbeschreibungsschnittstelle ist die Grundlage für die Bereitstellung der qualitativen Informationen, d.h. in welchen Zusammenhängen alle zu einer Klasse gehörenden Informationsobjekte verwendet werden. Konkret wird diese Eigenschaft der Informationsobjekte in dem Abschnitt „MSS-Modell“ angewendet. Das Protokoll sieht wie folgt aus:

```
(InformationObject allInstances)
=> antwortet mit der Menge aller Informationsobjekte

(Kunde allInstances)
=> antwortet mit der Menge aller Kundenobjekte
```

Dieses Protokoll wurde ebenfalls aus den oben beschriebenen Gründen der Wartungsfreundlichkeit als Adapter-Pattern konzipiert.

5.1.2 Verwaltung und Zugriff auf die Eigenschaften der Informationsobjekte

Im Rahmen des ooMSS soll es nach dem Kriterium der Wartungsfreundlichkeit möglich sein, das Informationsmodell strukturell anzupassen. Grundlage des ooMSS ist die kontrollierte Anpassung der mit Selbstbeschreibung dokumentierten Attribut- und Methodenstrukturen der Informationsobjektklassen. Die entsprechenden Protokolle sind allgemein und für das Beispiel einer Kundenklasse:

```
(InformationObject addAttribut: #attributeName)
=> ein Attribut wird erzeugt

(InformationObject removeAttribut: #attributeName)
=> ein Attribut wird entfernt
```

```
(InformationObject addMethod: #methodName withBody: '^self')
```
=> eine Methode wird erzeugt

```
(InformationObject removeMethod: #methodName)
```
=> eine Methode wird entfernt

```
(Kunde addAttribut: #adresse)
```
=> das Attribut `adresse` wird erzeugt

```
(Kunde removeAttribut: #adresse)
```
=> das Attribut `adresse` wird entfernt

```
(Kunde addMethod: #umsatz withBody: '^self berechneUmsatz')
```
=> die Methode `umsatz` wird erzeugt

```
(Kunde removeMethod: #umsatz)
```
=> die Methode `umsatz` wird entfernt

Es ist zu beachten, daß für die Attribute mit ihrer Definition parallel die passenden Zugriffsmethoden angelegt werden. Für das ooMSS gilt die Konvention, die Attributwerte nur über diese Methoden verändern zu dürfen. Die folgenden zwei Zugriffsmethoden sind damit das Grundgerüst für den geregelten Zugriff:

```
(anInformationObject attributeName: anObject)
```
=> der Attributwert wird geändert

```
(anInformationObject attributeName)
```
=> mit dem Attributwert wird geantwortet

Besitzen bspw. die Instanzen der Klasse `Kunde` das Attribut `Adresse`, dann sieht die Schnittstelle für diese Eigenschaft wie folgt aus:

```
(einKunde adresse: 'Katharinenstr. 3, ...')
```
=> das Attribut `adresse` des Kundenobjekts wird verändert

```
(einKunde adresse)
```
=> das Kundenobjekt antwortet mit seinem hinter dem Attribut `adresse` verwalteten Wert (also z.B. mit der Zeichenkette 'Katharinenstr. 3, ...')

An dieser Stelle der Systembeschreibung ermöglichen die Zugriffsmethoden noch den freien unbeschränkten Zugriff auf die Attributwerte. Durch die Definition dieser Schnittstellen werden weitergehende Beschreibungen der Informationsobjekte möglich, wodurch z.B. Attributwert-Beschränkungen und die Akti-

vierung von Methoden nach Zugriffen umgesetzt werden können. Für die Methoden gilt, daß im Rahmen ihrer Definition als fester Bestandteil der Code des Methodenrumpfs als zusätzlicher Parameter zu übergeben ist.

5.1.3 Verwaltung von Informationsobjektmengen

“Treating multiple business objects as a set is a challenge.” [Hewi98 S.18]

Zur Abbildung von komplexen MSS-Sachverhalten und deren Auswertung ist es notwendig Informationsobjektmengen zu verwalten. Wie in Abschnitt 3.1.4 erläutert wurde, ist die Komposition ein einfaches und leistungsfähiges Konzept zur hierarchischen Aufteilung eines Informationssystems.
Für den effektiven Umgang mit den Informationsobjektmengen müssen einzelne Informationsobjekte und Mengen von Informationsobjekten im ooMSS die gleiche grundlegende Schnittstelle besitzen, damit alle im ooMSS abgebildeten komplexen Informationen soweit wie möglich gleichbehandelt werden können.
Einzelne und Mengen von Informationsobjekten besitzen z.B. einheitlich als grundlegende Eigenschaften eine eindeutige textuelle Repräsentation und eine Zustandsverwaltung. Weitere MSS-spezifische Eigenschaften werden im Abschnitt MSS-Modell beschrieben.

Grundsätzlich läßt sich die Verwaltung von Informationsobjektmengen mit zwei alternativen Konzeptionen erreichen: Es wird eine Unterklasse von `Collection` gebildet, der Klasse, mit der standardmäßig Mengen von Objekten verwaltet werden. Die Unterklasse bekommt zusätzlich das Protokoll von `InformationObject`. Oder eine Unterklasse von `InformationObject` wird gebildet, die um die Collection-Schnittstelle ergänzt wird.

Im Rahmen des ooMSS wurde die zweite Alternative gewählt. Die Klasse `InformationCollection` ist eine Unterklasse von `InformationObject`. Die Begründung dafür ist, daß die Protokolle der Collection-Klasse stabiler sind als die von `InformationObject`. Die häufiger auftretenden Änderungen eines Protokolls von `InformationObject` müssen, im günstigsten Fall, nur an einer Stelle im System, in der Klasse `InformationObject`, durchgeführt werden. Zusätzlich läßt sich durch die gewählte Klassenhierarchie auf einfache Art überprüfen, ob ein Objekt die Schnittstelle eines Informationsobjekts besitzt. Das ist der Fall, wenn es eine Instanz einer Unterklasse von `InformationObject` ist.

Die zusätzlichen Protokolle der Klasse `InformationCollection` sind ein Ausschnitt der Schnittstelle von `Collection`. Zu den realisierten Protokollen zählt u.a. das Hinzufügen (`add:`) und Entfernen (`remove:`) von Elementen, die Überprüfung, ob ein Element enthalten ist (`includes:`) und die verschiedenen Iteratoren über die einzelnen Elemente (z.B. `do:`).

Das Protokoll der Klasse `InformationObject` wird im Rahmen des Umgangs mit Informationsobjektmengen um die Definition einer multiplen Attributeigenschaft ergänzt:

```
(InformationObject addMultipleAttribut: #attributeName)
```
=> das multiple Attribut wird erzeugt

```
(Kunde addMultipleAttribut: #auftraege)
```
=> das Attribut `auftraege` wird erzeugt

Die Selbstbeschreibungsschnittstelle wird um ein passendes Protokoll ergänzt:

```
(InformationObject hasMultipleAttribut: #attributeName)
```
=> antwortet mit einem boolschen Wert, in Abhängigkeit davon, ob das multiple Attribut existiert

```
(Kunde hasMultipleAttribut: #auftraege)
```
=> antwortet mit einem boolschen Wert, in Abhängigkeit davon, ob ein multiples Attribut `auftraege` existiert

Mit der Definition eines multiplen Attributs werden zusätzlich die notwendigen Zugriffsmethoden für die Instanzen generiert:

```
(anInformationObject addAttributeName: anObject)
```
=> dem Attribut wird ein Objekt hinzugefügt

```
(einKunde addAuftraege: einAuftrag)
```
=> dem Attribut `auftraege` wird ein Auftrag hinzugefügt

```
(anInformationObject removeAttributeName: anObject)
```
=> aus dem Attribut wird ein Objekt entfernt

```
(einKunde removeAuftraege: einAuftrag)
```
=> aus dem Attribut `auftraege` wird ein Auftrag entfernt

`(anInformationObject includesAttributeName: anObject)`
=> antwortet mit einem boolschen Wert, in Abhängigkeit davon, ob das Objekt im Attribut enthalten ist

`(einKunde includesAuftraege: einAuftrag)`
=> antwortet mit einem boolschen Wert, in Abhängigkeit davon, ob der Auftrag im Attribut auftraege enthalten ist

Bei der Verwaltung von multiplen Attributen eines Informationsobjekts wird somit das Composition-Pattern angewendet [GHJV95 S.163]. Das Informationsobjekt verwaltet zu diesem Zweck hinter seinem multiplen Attribut eine InformationCollection. Die `InformationCollection` stellt innerhalb des ooMSS die Collection-Funktionalität zur Verfügung, indem sie als Adapter [GHJV95 S.139] die Nachrichten an die Collection, die die Informationsobjekte verwaltet, weiterleitet.
Die Grundeigenschaften des Informationsmodells basieren auf denen der abstrakten Klasse `InformationObject` und der konkreten Klasse `InformationCollection`. Durch `InformationObject` werden die Eigenschaften zur Selbstbeschreibung und Verwaltung der Attribute und Methoden zur Verfügung gestellt. Mit der Klasse `InformationCollection` werden Mengen von Informationsobjekten als Unterklasse von `InformationObject` verwaltet. Das dazu passende Objektmodell dokumentiert dies als Composite-Pattern (siehe Abbildung 5.1).

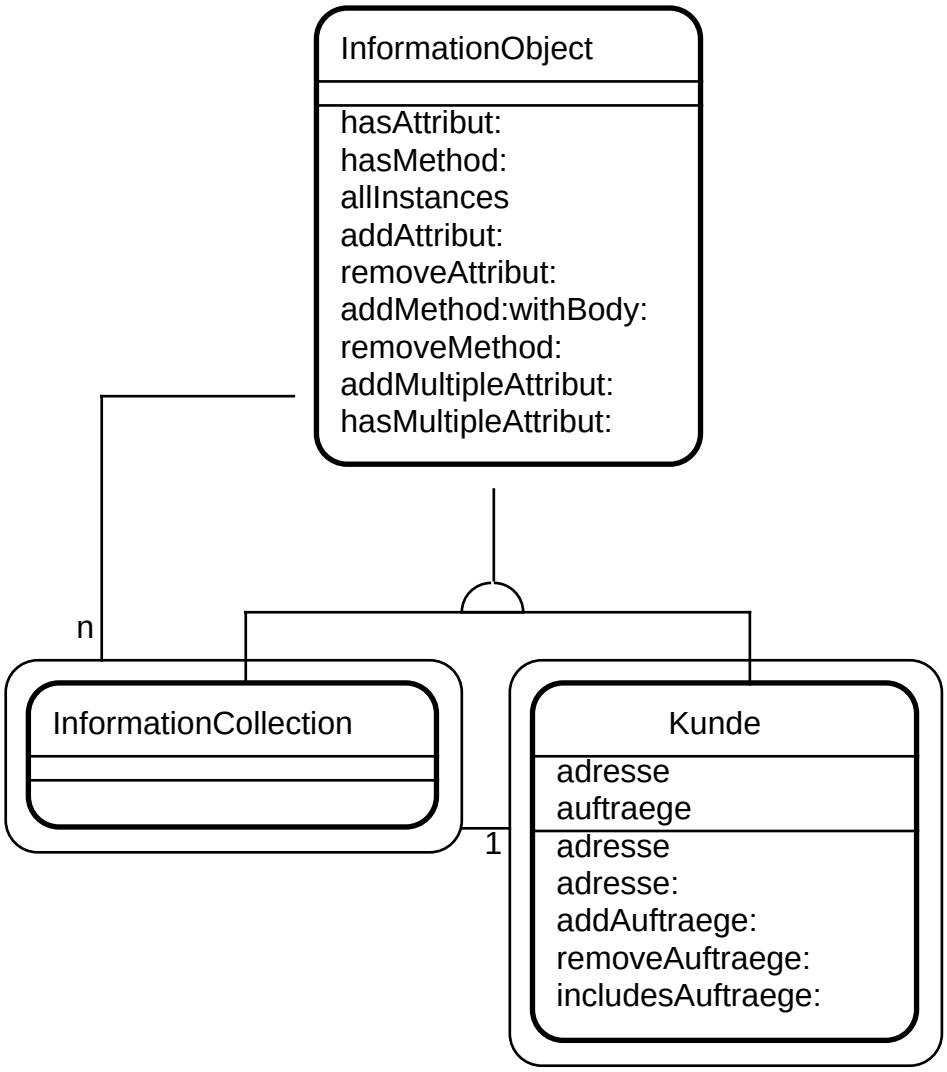

Abbildung 5.1: Objektmodell mit InformationObject, InformationCollection und Kunde (Objektnotation nach [CoYo94a])

5.2 Die Eigenschaften des Werkzeugmodells

5.2.1 Basiskonzept

Zur Erfüllung der an das Werkzeugmodell gestellten Anforderungen wird als konzeptioneller Kern das Interpreter-Pattern [GHJV95 S.243] eingesetzt. Die einzelnen analytischen Inhalte des MSS-Bereichs werden als Ausdrücke in einer einfachen Sprache umgesetzt und mit einem Interpreter ausgeführt. Das grundlegende Ausdruckselement dieser MSS-spezifischen Sprache ist das Werkzeugobjekt, das durch die Klasse `ToolObject` modelliert wird. Beim Einsatz des Werkzeugmodells wird kein Methoden-Code erstellt, sondern eine Menge von Werkzeugobjekten instanziert, miteinander verbunden und durch Parameter angepaßt.

Einheitlich verarbeiten die Werkzeugobjekte einen Ausschnitt des Informationsmodells in Form einer `InformationCollection`. Jedes Werkzeugobjekt besitzt dazu das gleiche Protokoll zur Verarbeitung:

`(aToolObject evaluate: anInformationCollection)`
=> das Werkzeugobjekt antwortet mit einem Ergebnis, für das eine Menge von Informationsobjekten verarbeitet wird

Hinter der mit dem Protokoll verbundenen Methode (`evaluate:`) steht die Interpretermethode, die die als Parameter übergebene Informationsobjektmenge verarbeitet.

Auf einer fachlichen Ebene wird mit der Verwendung einer Werkzeugmetapher dem Anwender vermittelt, daß durch das System eine Menge von einfach bis komplex aufgebauten Werkzeugen zur Verfügung steht. Die Flexibilität, mit der ein Werkzeug eingesetzt werden kann, ist dabei abhängig von seiner Komplexität: Je komplexer das Werkzeug aufgebaut ist, desto spezialisierter ist sein Verwendungszweck.
Bestandteil der Metapher ist weiterhin, daß sich die Werkzeuge miteinander kombinieren lassen (siehe Abbildung 5.2).

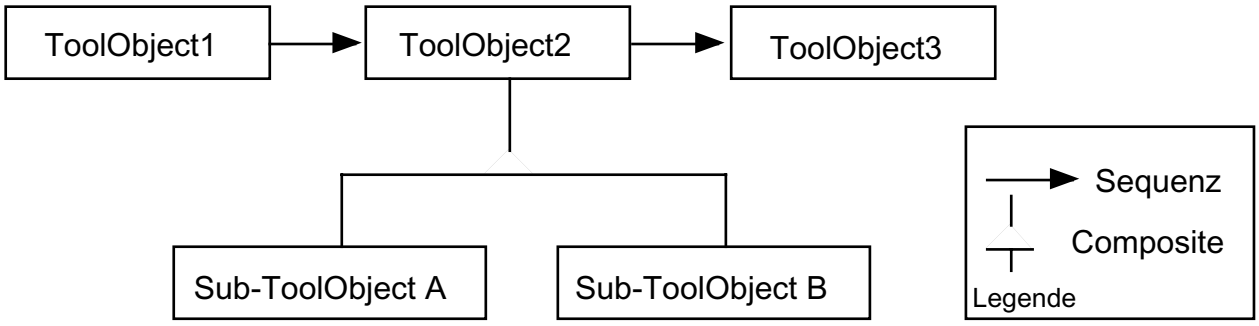

Abbildung 5.2: Kombinationen von Werkzeugobjekten

Ein Teil der Werkzeugobjekte setzt sich hierarchisch aus (Sub-)Tools zusammen, an die es bei der Verarbeitung Teilaufgaben delegiert (Composite-Pattern [GHJV95 S.163]). Durch die Verwendung der Tool-Komponenten wird eine Redundanz bei der Funktionalität der einzelnen Toolklassen vermieden. Der Methoden / Interpreter-Code eines Werkzeugs bleibt überschaubar.
Das Werkzeugmodell erlaubt zusätzlich die Bildung von sequentiellen Bearbeitungsketten. Das Ergebnis eines Werkzeugs wird dabei als Eingabe eines Folgewerkzeugs verwendet. Das Protokoll beschreibt die sequentielle Verarbeitung in Form einer Verschachtelung der Werkzeuge:

```
(secondToolObject evaluate:
    (firstToolObject evaluate: anInformationCollection))
```
=> Eine Informationsobjektmenge wird zunächst durch das erste Werkzeugobjekt verarbeitet. Das (Zwischen-)Ergebnis besteht aus einer Informationsobjektmenge, die durch das zweite Werkzeug weiterverarbeitet wird.

Im folgenden Abschnitt wird das Werkzeugmodell mit den einzelnen MSS-Toolklassen umfassend strukturiert beschrieben, indem auf Funktionalität, zugrundeliegende Konzepte, Einsatzbedingungen, Schnittstellen und Beispiele eingegangen wird. Im Anhang gibt die Abbildung A.2 einen Überblick.

5.2.2 PathTool

Mit dem `PathTool` wird ein Werkzeug bereitgestellt, durch das als Alternative zu einer direkten Referenz ein Teilbereich des Informationsmodells relativ von einem Ausgangsobjekt aus beschrieben wird. Zu diesem Zweck benötigt ein `PathTool` als Parameter eine Informationsobjektmenge, die den Ausgangspunkt der Beschreibung darstellt, und einen relativen Pfad, der festlegt, wie das Informationsmodell entlang der Eigenschaften der beteiligten Informationsobjekte zu durchlaufen ist.

```
((PathTool path: #(#methodSelector1 #methodSelector2))
    evaluate: anInformationCollection)
```
=> Das Werkzeug antwortet mit einer Menge von Informationsobjekten, die durch den Pfad beschrieben wurde.

```
((PathTool path: #(#auftraege #positionen #produkte))
    evaluate: eineKundenMenge)
```
=> Das Werkzeug liefert alle Produkte, die die Kunden gekauft haben.

Das zugrundeliegende Konzept basiert auf der bei [AlBW98 S.117] beschriebenen generischen Lösung, um auf einen beliebigen Aspekt eines Objekts mit einem pfad-basierten Adapter zuzugreifen. Der Pfad besteht dabei aus einer Sequenz von Methodenaufrufen, die traversiert werden müssen, um die Objekte der Ergebnismenge zu erhalten [WoSm95 S.25].
Für den Einsatz in der oben beschriebenen Weise ist ein einfacher Methodenaufruf ungeeignet, um zu dem gewünschten Ergebnis zu kommen. Z.B. besitzen die in einer Informationsmenge enthaltenen Kundenobjekte die Eigenschaft `auftraege` und nicht die Informationsmenge selbst(!). Für die Abarbeitung jedes Pfadbestandteils muß an die Informationsmenge eine Anfrage gestellt werden, durch die sie über die Menge ihrer enthaltenen Informationsobjekte iteriert.

Jedes Informationsobjekt liefert mit seinem Eigenschaftswert, der durch den Pfadbestandteil spezifiziert ist, einen Beitrag zu dem Ergebnis. Die von einer Informationsmenge als Ergebnis zurückgegebene Informationsmenge enthält die gesammelten Eigenschaften seiner Elemente in einer flachen, nicht hierarchischen Form. Die Ergebnismenge enthält z.B. direkt alle Auftragsobjekte und nicht für jeden beteiligten Kunden eine Menge von Auftragsobjekten. Mit dieser Funktionalität wird sichergestellt, daß einheitlich flach auf die Objekte einer Informationsmenge zugegriffen werden kann.

An dieser Stelle des ooMSS-Konzepts wird deshalb die Klasse `Information-Collection` um das Protokoll `nextOnPathElement:` erweitert:

`(anInformationCollection nextOnPathElement: #methodSelector)`
=> Antwortet mit einer Informationsobjektmenge, die jeweils hinter der durch den Methodenselektor beschriebenen Eigenschaft verwaltet wird.

`(eineAuftragsMenge nextOnPathElement: #positionen)`
=> Antwortet mit einer Menge aller Ausftragspositionen der enthaltenen Aufträge.

5.2.3 ValueTool

Mit einem `ValueTool` wird ein atomarer Wert in einer Werkzeugkombination integriert. Die Aufgabe eines `ValueTools` ist es, als Bestandteil anderer Werkzeuge, wie z.B. `FilterTools`, konstante Werte zu repräsentieren. `ValueTools` besitzen dazu ein einfaches Protokoll, durch das der Wert bei der Instanziierung gesetzt wird:

`((ValueTool value: anObject) evaluate: dummyInformationObjects)`
=> Das Werkzeug antwortet mit dem als Parameter übergebenen Objekt. Die hinter dem evaluate-Protokoll übergebenen Informationsobjekte werden von dem Werkzeug nicht benötigt. Bei der Verarbeitung werden sie deshalb ignoriert.

`((ValueTool value: 42) evaluate: dummyInformationObjects)`
=> Das Werkzeug antwortet mit dem Zahlenwert.
`((ValueTool value: 'Schmidt') evaluate: dummyInformationObjects)`
=> Das Werkzeug antwortet mit der Zeichenkette.

Das ValueTool basiert mit seinen Eigenschaften auf dem Ausdruck, mit dem standardmäßig konstante Werte im Rahmen der Anwendung des Interpreter-Pattern repräsentiert werden [AlBW98 S.261].

5.2.4 BooleanTool

Mit dem `BooleanTool` werden boolsche Vergleiche beschrieben.

Ein `BooleanTool` besitzt als Bestandteile zwei weitere Werkzeuginstanzen (`leftTool, rightTool`) vom Typ `BooleanTool`, `PathTool` oder `ValueTool`. Die `BooleanTools` vergleichen die Ergebniswerte ihrer Bestandteile nach einer boolschen Operation, die davon abhängig ist, welcher konkreten Unterklasse das Werkzeugobjekt angehört. Die unterstützten Operationen werden durch die Klassennamen, wie z.B. `AndTool`, `OrTool`, `EqualTool` oder `GreaterThanTool`, vermittelt.
Die BooleanTools werden, wie `ValueTools`, im ooMSS nicht eigenständig verwendet, sondern sind Bestandteil anderer Werkzeugobjekte. Z.B. werden im Rahmen von `FilterTools` durch sie Bedingungen beschrieben.

Eine Beschreibung der Protokolle sieht wie folgt aus...

```
(BooleanTool leftTool: subTool1 rightTool: subTool2)
    evaluate: anInformationCollection)
```
=> Das Werkzeug antwortet mit einer Menge von boolschen Werten, die von der konkreten boolschen Operation und den übergebenen Werkzeugen abhängig sind.

```
(GreaterThanTool
    leftTool: (PathTool path: #(#umsatz))
    rightTool: (ValueTool value: 100000))
    evaluate: eineProduktMenge)
```
=> Das Werkzeug antwortet mit einer Menge von boolschen Werten, die davon abhängig sind, ob die einzelnen Produkte einen größeren Umsatzwert als 100 000 haben oder nicht.

Das Konzept des BooleanTool ist eine Spezialisierung des beim Interpreter-Pattern verwendeten sequentiellen Ausdrucks [GHJV95 S.246].

5.2.5 FilterTool

Mit dem `FilterTool` wird ein Ausschnitt des Informationsmodells durch eine Bedingung beschrieben. Der Unterschied zu der Beschreibung mit dem `PathTool` ist, daß die Ergebnismenge eine Teilmenge der übergebenen Informationsobjektmenge ist. Fester Bestandteil des `FilterTools` ist eine Bedingung in Form von Instanzen der Unterklassen von `BooleanTool`, die von allen Informationsobjekten der Ergebnismenge erfüllt werden.

```
((FilterTool boolean: aBooleanTool) evaluate:
   anInformationCollection)
```
=> Das `FilterTool` antwortet mit allen Informationsobjekten, die die durch das `BooleanTool` festgelegte Bedingung erfüllen.

```
((FilterTool boolean: (EqualTool
   leftTool: (PathTool path: #(#name))
   rightTool: (ValueTool value: 'Müller')))
   evaluate: eineKundenMenge)
```
=> Alle Kunden mit dem Namen Müller werden beschrieben.

5.2.6 AggregationTool

Durch das `AggregationTool` wird eine Informationsobjektmenge zu einer Zahl konsolidiert. `AggregationTools` sind im ooMSS die Basis für die Erstellung von Kennzahlensystemen, die als grundlegende Instrumente für die betriebswirtschaftliche Informationsversorgung dienen.
Fester Bestandteil des `AggregationTool` ist dazu der Parameter, mit dem die Art der Konsolidierung spezifiziert wird. Mögliche Konsolidierungsarten sind u.a. die Anzahl der Objekte (`#count`) und bezogen auf die Objekteigenschaften die Summe (`#sum`), das Maximum (`#max`) oder das Minimum (`#min`). Wird für eine Aggregierung eine Eigenschaft der Objekte verwendet, dann wird zusätzlich als Parameter der Pfad der Eigenschaft angegeben. Um auf die entsprechenden Eigenschaften der Informationsobjekte zuzugreifen, erzeugt und verwendet ein `AggregationTool` dazu ein passendes `PathTool`.

```
((AggregationTool aggType: #count)
   evaluate: anInformationCollection)
```
=> Das Werkzeug antwortet mit der Quantität der Informationsobjektmenge.

```
((AggregationTool aggType: #count)
   evaluate: eineKundenMenge)
```
=> Hiermit wird die Anzahl der Kunden berechnet.

```
((AggregationTool
   path: #(#methodSelector)
   aggType: #aggregationSelector)
   evaluate: anInformationCollection)
```
=> Die Informationsobjektmenge wird nach der durch die Selektoren spezifizierten Eigenschaft und Konsolidierungsart aggregiert.

```
((AggregationTool path: #(#umsatz) aggType: #sum)
   evaluate: eineKundenMenge)
```
=> Hiermit wird die Umsatzsumme der Kunden berechnet.

5.2.7 SortTool

Durch das `SortTool` wird eine Informationsobjektmenge nach einer Eigenschaft sortiert. Als Bestandteil besitzt das Tool den Pfad der Eigenschaft, nach der sortiert wird sowie die Sortierbedingung, durch die die Sortierreihenfolge, z.B. auf- und absteigend, festgelegt wird. Um auf die Eigenschaften der Informationsobjekte zuzugreifen, verwendet ein `SortTool` ebenfalls ein `PathTool`.

```
((SortTool
   path: #(#methodSelector)
   order: #sortOrderSelector)
   evaluate: anInformationCollection)
```
=> Die Informationsobjektmenge wird nach der durch die Selektoren spezifizierten Eigenschaft und Reihenfolge sortiert.

```
((SortTool
   path: #(#preis)
   order: #descending)
   evaluate: eineProduktMenge)
```
=> Alle Produkte der Menge werden nach ihrem Preis absteigend sortiert.

5.2.8 SourceTool

`SourceTool` ist der Bestandteil des Werkzeugmodells, der dem ooMSS den Zugriff auf interne und externe Informationen gibt.

Das SourceTool liefert zwei Arten von Informationen in Form von Objekten:

Durch die erste Anwendungsform werden Informationen, die Domänenelemente und Beziehungen zwischen den Elementen repräsentieren, bereitgestellt. Die Informationen sind als Instanzen in Klassen des Informationsmodells organisiert und deshalb im Rahmen des ooMSS strukturiert. Ein `SourceTool` repräsentiert im ooMSS eine Objektquelle für eine bestimmte Informationsklasse. Als notwendiger Parameter wird für diesen Einsatzzweck die Bezeichnung der Klasse benötigt. Zusätzlich kann mit einer Abfrage in Form eines `BooleanTool` die Menge der zur Verfügung gestellten Instanzen beschränkt werden.

```
(SourceTool onClassNamed: #classNameSelector)
```
=> Antwortet mit allen Instanzen der spezifizierten Klasse.

```
(SourceTool onClassNamed: #Kunde)
```
=> Antwortet mit allen Kundenobjekten.

```
(SourceTool
   onClassNamed: #classNameSelector
   boolean: aBooleanTool)
   evaluate)
```
=> Das Werkzeug antwortet mit allen Instanzen der spezifizierten Klasse, die die Bedingung erfüllen.

```
(SourceTool
   onClassNamed: #Produkt boolean:
   (GreaterThanTool
        leftTool: (PathTool path: #(#umsatz))
        rightTool: (ValueTool value: 100000))
   evaluate)
```
=> Hiermit werden alle Produkte angesprochen, deren Umsatzzahl größer als 100.000 ist.

Die als Parameter übergebene Abfrage setzt das `SourceTool` von der Abfragesyntax des ooMSS in eine für die konkrete Objektquelle passende Form um, z.B. für das Smalltalk-Image oder für eine objektorientierte Datenbank. Die Objektquelle führt die Abfrage aus und liefert als Ergebnis die Informationsobjekte.

In der zweiten Anwendungsform eines `SourceTools` werden die Informationen, deren Ursprung ooMSS-extern ist, in das Informationsmodell integriert. Die Informationen bestehen z.B. aus relational strukturierten Informationen oder alphanumerischen und multimedialen Daten, die aus der Sicht des ooMSS unstrukturiert sind. Durch das `SourceTool` werden die internen Informations-

objekte durch die externen Informationen ergänzt.
Der notwendige Parameter ist in dieser Anwendungsform die Zeichenkette, die die Adresse der externen Information beschreibt.
Als Beispiel für den externen Bereich soll das Internet dienen. Es können Internet-Dokumente, wie HTML-Seiten, als Informationselemente im ooMSS vollständig repräsentiert werden. Das SourceTool verwaltet dabei sowohl die Adresse als auch den letzten aktuellen Inhalt, der sich hinter ihr befand.

```
((SourceTool onUrl: anUrlAdress) evaluate)
```
=> Antwortet mit der durch die Internetadresse direkt spezifizierten Information.

```
((SourceTool onUrl:
    'www.grosshaendler.de/dvd-sparte/bericht1299.html') evaluate)
```
=> Antwortet mit dem als HTML-Seite abgelegten Bericht.

Ein weiteres Anwendungsbeispiel ist die Verwaltung von externen Informationsbezügen in Informationsobjekteigenschaften, z.B.:

```
(Kunde homepage:
    (SourceTool onUrl: 'www.oec.uni-osnabrueck.de/wi2/dkrueger')
    evaluate)
```
=> Als Bestandteil des Kunden wird seine Homepage verwaltet.

```
(aPerformanceIndicatorTool addComment:
    (SourceTool onUrl:
         'www.cnn.com/19.11.1999/topnews.html')
    evaluate)
```
=> Eine Kennzahl (engl. Performance Indicator) wird um einen textuelle Anmerkung in Form einer HTML-Seite ergänzt.

Eine Adresse muß nicht fest in Form einer Zeichenkette definiert, sondern kann als Abfrage verwaltet werden. Hierdurch können vom ooMSS aus Internetdienste, z.B. Suchmaschinen, aktiv genutzt werden. In diesem Fall wird als Parameter die Informationsquelle und ein durch Zeichenketten beschriebener MSS-Kontext spezifiziert. Aus den Schlüsselwörtern des Kontext wird durch das `SourceTool` eine für die Informationsquelle passende Abfrage generiert. Die Abfrage wird an die Informationsquelle übergeben und das Ergebnis vom `SourceTool` als eine Menge von Verweisen auf HTML-Seiten präsentiert.

```
((SourceTool onUrl: anUrlAdress withContext: aTool) evaluate)
```
=> Antwortet mit den über den Internetdienst abrufbaren Informationen, die einen Bezug zu den durch das Werkzeug angegebenen Suchbegriffen besitzen.

```
((SourceTool
    onUrl: 'www.altavista.com'
    withContext: (AndTool
          leftTool: (ValueTool value: 'DVD-Player')
          rightTool: (ValueTool value: 'Deutschland')))
    evaluate)
```
=> Antwortet mit den über die Internetsuchmaschine “Altavista” abrufbaren Informationen, die einen Bezug zu den spezifizierten Suchbegriffen besitzen.

Bei Ehrmann/Engler ist beschrieben, wie eine Software für den Abruf von HTML-Dokumenten auf einfache Weise an nicht standardmäßig unterstützte Suchmaschinen angepaßt wird [EhEn00]. In Anlehnung an diese Vorgehensweise werden die SourceTools im ooMSS eingerichtet, um das konkrete Protokoll einer beliebigen Suchmaschine zu unterstützen.

Die Suchbegriffe müssen nicht als feste Zeichenketten vorgegeben sein, sondern können sich aus Eigenschaftswerten von Informationsobjekten bilden. Das folgende Beispiel demonstriert die Anwendungsform. Für eine Menge von Auswertungsobjekten werden Informationen gesucht, die jeweils einen Bezug zu der ausgewerteten Produktgruppe und dem Land besitzen.

```
((SourceTool onUrl: anUrlAdress withContext: aTool)
    evaluate: anInformationCollection)
```
=> Antwortet mit den über den Internetdienst abrufbaren Informationen, die einen Bezug zu den durch das Tool beschriebenen Eigenschaften der Informationsobjekte besitzen.

```
((SourceTool onUrl: 'www.altavista.com' withContext:
    (AndTool
          leftTool: (PathTool path: #(#produkt #gruppe #name))
          rightTool: (PathTool path: #(#region #land #name))))
evaluate: eineAuswertungsmenge)
```
=> Antwortet mit den über die Internetsuchmaschine “Altavista” abrufbaren Informationen, die einen Bezug zu den beschriebenen Eigenschaften der Auswertungsobjekte besitzen.

In beiden Fällen werden die Informationsbezüge als Liste von `SourceTools` in der oben beschriebenen Form zurückgegeben.

Zusammenfassend läßt sich festhalten, daß mit dem `SourceTool` im ooMSS alle internen und externen Informationsquellen repräsentiert werden können. Die Werkzeuge liefern eine einfache einheitliche Schnittstelle, durch die mit den Quellen kommuniziert werden kann. Die Komplexität bleibt bei der Anwendung verborgen. Die Abfragen werden passend für die jeweilige Quelle umgesetzt. Die Protokolle werden dabei durch ein Adapter-Pattern übertragen. Durch ein Facade-Pattern werden Subsysteme, die sich aus einer Menge von Klassen zusammensetzen, einheitlich repräsentiert.

5.2.9 CubeTool

Eine MSS-spezifische Analyseform ist das OnLine Analytical Processing (OLAP). Dabei handelt es sich um die Konsolidierung, Betrachtung und Analyse von mehrdimensionalen Daten [CoCS93]. Für die Analyse, die Vorhersage und die Planung wird für den Anwender die Möglichkeit zur Verfügung gestellt, die Informationen der Domäne aus verschiedenen Blickwinkeln zu betrachten. Würfel (engl. Cubes) sind beim OLAP-Ansatz die grundlegende Modellstruktur, in denen die Daten organisiert sind. Die Anwender erkennen in den Würfeln eine n-dimensionale Abbildung ihres Geschäftsbereichs [CoMM97 S.13]. Trotz der Verwendung der Würfelmetapher können die OLAP-Würfel mit mehr als drei Dimensionen strukturiert werden.
Dimensionen sind für den Anwender grundlegende Möglichkeiten, den Geschäftsbereich zu kategorisieren [CoMM97 S.13]. Eine Dimension besteht aus diskreten Dimensionswerten. Jeder Dimensionswert untergliedert als endliches Intervall seine Dimension und beschreibt einen Würfelausschnitt.
Im Rahmen des ooMSS wird durch das `CubeTool` eine Informationsobjektmenge in eine (aggregierte) n-dimensionale Struktur transformiert. Die komplexen Informationsobjekte können so mit einer Anzahl von Eigenschaften aus verschiedenen Perspektiven ausgewertet werden. Ein `CubeTool` ist das am komplexesten aufgebaute Werkzeug im ooMSS, was sich auch in seiner Schnittstelle widerspiegelt. Die Schnittstellenbeschreibung beschränkt sich deshalb hier auf die für das Konzeptverständnis elementaren Protokolle.

Zum Aufbau einer Würfelstruktur wird eine Informationsobjektmenge als contents-Parameter übergeben. Die Informationen werden passend eingeordnet.

```
(aCube := (CubeTool new).
aCube evaluate: anInformationCollection)
```
=> Erzeugt ein `CubeTool` und antwortet mit allen enthaltenen Informationsobjekten.

```
(einWuerfel := (CubeTool new).
einWuerfel evaluate: eineMengeVonAuftragspositionen)
```
=> Erzeugt ein `CubeTool` und antwortet mit allen enthaltenen Auftragspositionen.

Um das Design des CubeTools so einfach wie möglich zu gestalten, bezieht sich eine Dimension auf die Eigenschaft einer Klasse und besitzt für jeden Eigenschaftswert einen Dimensionswert. Dimensionen werden in Form von PathTools übergeben:

```
(aCube
    addPath: (PathTool path: #(#pathSelector1);
    addPath: (PathTool path: #(#pathSelector2 #pathSelector3))
```
=> Die Informationsobjekte werden nach ihren Eigenschaftsausprägungen in die Würfelstruktur eingeordnet.

```
(einWuerfel
    addPath: (PathTool path: #(#produkt);
    addPath: (PathTool path: #(#auftrag #datum);
    addPath: (PathTool path: #(#auftrag #kunde #kundenGruppe)
```
=> Die Auftragspositionen werden nach ihren Eigenschaftsausprägungen in die Würfelstruktur eingeordnet.

Um auf die in einem Würfelausschnitt verwalteten Informationsobjekte zuzugreifen, wird dieser zusätzlich mit einer Abfrage beschrieben:

```
(aCube atPath: (PathTool path: #(#pathSelector1))
```
=> Der Würfel antwortet mit einer n-dimensionalen Struktur, bei der alle Informationsobjekte nach ihren durch den Pfad spezifizierten Eigenschaften kategorisiert sind.

```
(einWuerfel atPath: (PathTool path: #(#produkt))
```
=> Der Würfel antwortet mit allen Auftragspositionen, die nach ihren Produkten kategorisiert sind.

Für eine Abfrage lassen sich mehrere Kategorien spezifizieren. Dazu stehen zwei alternative Protokolle zur Verfügung:

```
(aCube atPath: aPathTool1 atPath: aPathTool2)
```
=> Der Würfel antwortet mit einer n-dimensionalen Struktur, bei der alle im Würfel enthaltenen Informationsobjekte nach den durch die PathTools beschriebenen Eigenschaften kategorisiert sind.

```
(einWuerfel
    atPath: (PathTool path: #(#produkt)
    atPath: (PathTool path: #(#auftrag #kunde #kundenGruppe))
```
=> Der Würfel antwortet mit einer n-dimensionalen Struktur, bei der die im Würfel enthaltenen Auftragspositionen nach den beteiligten Produkten und der Kundengruppe, zu dem der Kunde des Auftrags gehört, kategorisiert sind.

```
(aCube atAllPaths: aPathCollection)
```
=> Der Würfel antwortet mit einer n-dimensionalen Struktur, bei der alle Informationsobjekte nach den übergebenen `PathTools` kategorisiert sind.

```
(einWuerfel atAllPaths: einePfadMenge)
```
=> Der Würfel antwortet mit einer n-dimensionalen Struktur, bei der alle Auftragspositionen nach den übergebenen `PathTools` kategorisiert sind.

Zur Beschränkung des Würfelraums wird eine Abfrage in Form eines `BooleanTools` verwendet:

```
(aCube setBoolean: aBooleanTool)
```
=> Durch den Würfel werden nur noch die Informationsobjekte, die die durch das `BooleanTool` beschriebene Bedingung erfüllen, repräsentiert.

```
einWuerfel setBoolean:(EqualTool
    leftTool: (PathTool path: #(#produkt #produktGruppe #name)
    rightTool: (ValueTool value: 'DVD-Player')))
```
=> Durch den Würfel werden nur noch die Auftragspositionen, deren Produkte zu der Gruppe der DVD-Player gehören, repräsentiert.

Auf die in einem Würfel enthaltenen Informationsobjekte läßt sich zusätzlich eine Aggregationvorschrift anwenden.

```
(aCube setAggregation:
    anAggregationTool;
    evaluate: anInformationCollection)
```
=> Der Würfel antwortet mit dem aggregierten Wert der Informationsobjekte.

```
(einWuerfel setAggregation:
    (AggregationTool path: #(#wert) aggType: #sum);
    evaluate: eineMengeVonAuftragspositionen)
```
=> Der Würfel antwortet mit dem aufsummierten Wert der Auftragspositionen.

```
(aCube
    setAggregation: aAggregationTool;
    atPath: aPathTool)
```
=> Der Würfel antwortet mit dem aggregierten Wert der Informationsobjekte, die nach der durch das PathTool spezifizierten Dimension gegliedert sind.

```
(einWuerfel
        setAggregation: (AggregationTool path: #(#wert) aggType:
        #sum);
    atPath: (PathTool path: #(#auftrag #datum))
    atPath: (PathTool path: #(#auftrag #kunde #kundenGruppe))
```
=> Der Würfel antwortet mit dem aufsummierten Wert der Auftragspositionen, die nach zwei Dimensionen gegliedert sind. Die Dimensionen sind das Auftragsdatum und die Kundengruppe, zu der der Kunde des Auftrags gehört.

Die Konzeption einer n-dimensionalen Würfelstruktur ist abhängig von einer Reihe von technischen Faktoren, sowie von der Speichereffizienz, der Zugriffsgeschwindigkeit auf die einzelnen Würfelausschnitte und der Flexibilität des Würfels, z.B. bei der Änderung der Dimensionalität durch das Hinzufügen von Pfaden.
Mögliche alternative Konzeptionen sind bei Pendse [Pend98] beschrieben. Zusammengefaßt gilt, je einfacher die Speicherungsform angelegt ist, desto komplexer und aufwendiger ist der Zugriff und die Verarbeitung der Würfel und umgekehrt. An einem Ende einer Skala steht die Alternative, bei der ein einfacher Würfel durch ein großes Array repräsentiert wird. Die Zugriffsmethoden berechnen in diesem Fall den Index, mit dem zugegriffen wird. Am anderen Ende der Skala werden aufwendig komplexe Würfelstrukturen, in Form von Meta-Arrays, verwaltet. Der Zugriff ist vereinfacht. Allerdings muß die Würfelstruktur bei typischen Würfeloperationen, z.B. beim Kippen (engl. dice), aufwendig umstrukturiert werden.

Für das `CubeTool` wird ein Konzept verwendet, dem eine komplexe Würfelstruktur zugrundeliegt. Diese Design-Entscheidung läßt sich damit begründen, daß mit objektorientierten Entwicklungssystemen besonders effizient komplexe Datenstrukturen verwaltet werden können[1]. Die n-dimensionalen Informationsstrukturen werden mit einer Baumstruktur abgebildet.
Jedes Baumelement ist ein Würfel, der eine dimensionierte Sichtweise auf die im Baum enthaltenen Informationen repräsentiert. Ein Würfelelement besitzt dazu als Eigenschaft eine Menge von Dimensionswerten, durch die der Ausschnitt beschrieben wird und eine Menge von Informationsobjekten, die im Ausschnitt enthalten sind, weil sie einen Bezug zu den Dimensionswerten besitzen (siehe Abbildung 5.3).

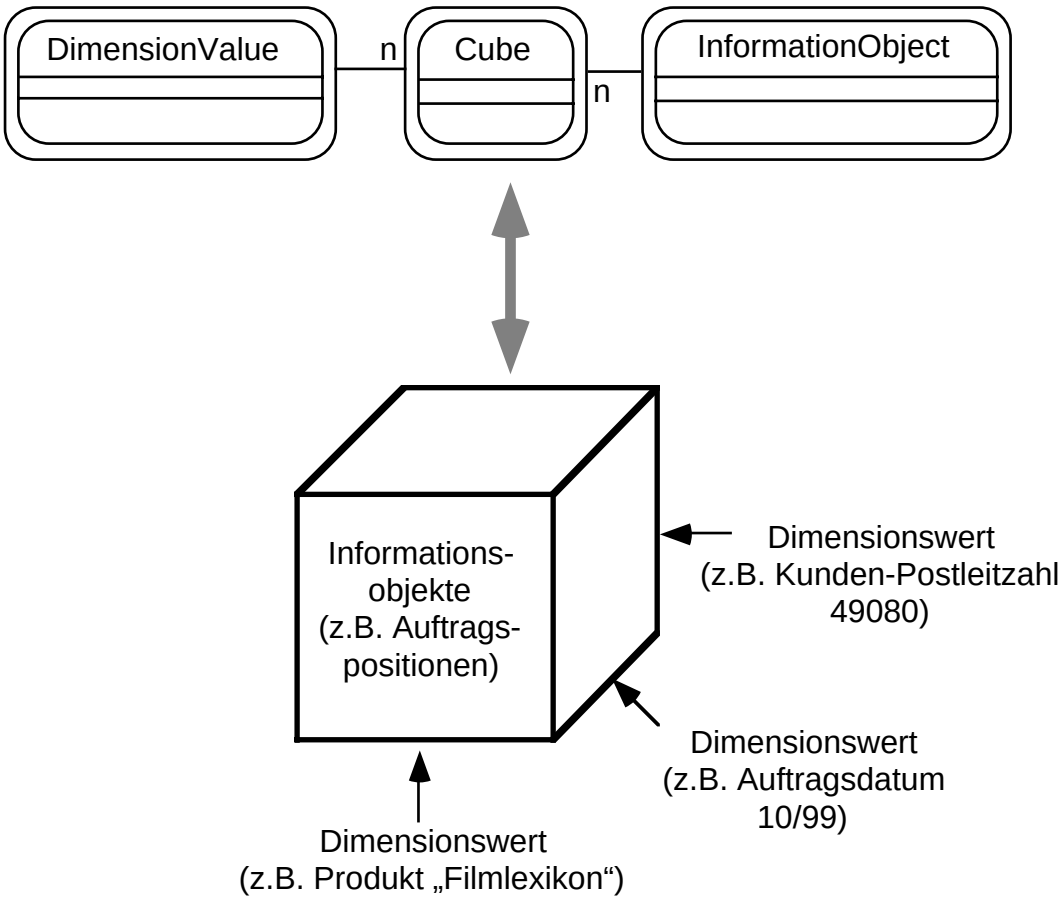

Abbildung 5.3: Würfel mit Dimensionswerten (Objektnotation nach [CoYo94a])

Z.B. enthält ein Würfel alle Auftragspositionen, die durch die Dimensionswerte Produkt „Filmlexikon“, das Auftragsdatum 10/99 und Kunden-Postleitzahl 49080 dimensioniert sind.

[1]Es wurden bei Piraino dazu für eine Aufgabe im Bereich der Matrizenoperationen, die mit dem OLAP-Bereich verwandt ist, die prozedurale C-Programmiersprache mit dem objektorientierten Smalltalk verglichen. Bezüglich der Effizienz beim Zugriff auf die Matrizen konnte Smalltalk seine besten Ergebnisse auf der Basis von komplexen Strukturen erreichen [Pira96 S.11]

Die einzelnen Baumelemente stehen in einer hierarchischen Beziehung zueinander. Je höher ein Element in der Hierarchie steht, desto weniger beschränkende Dimensionswerte besitzt es und desto größer ist seine Menge von Informationsobjekten. In der Hierarchie am höchsten steht das (Baum-) Wurzelelement. Es repräsentiert ohne dimensionale Einschränkungen alle Informationsobjekte eines Baums.
Zur Verwaltung der Hierarchie besitzt ein Baumelement zusätzlich als Eigenschaften seine hierarchisch über- und untergeordneten Elemente. Von jedem Element können Unterelemente abgeleitet werden (`subCubes`). Alle Unterelemente besitzen als Eigenschaft gegenüber ihrem Oberelement einen zusätzlichen Dimensionswert und einen Bezug auf das Oberelement (`parentCube`). Die folgende Abbildung 5.4 veranschaulicht dies:

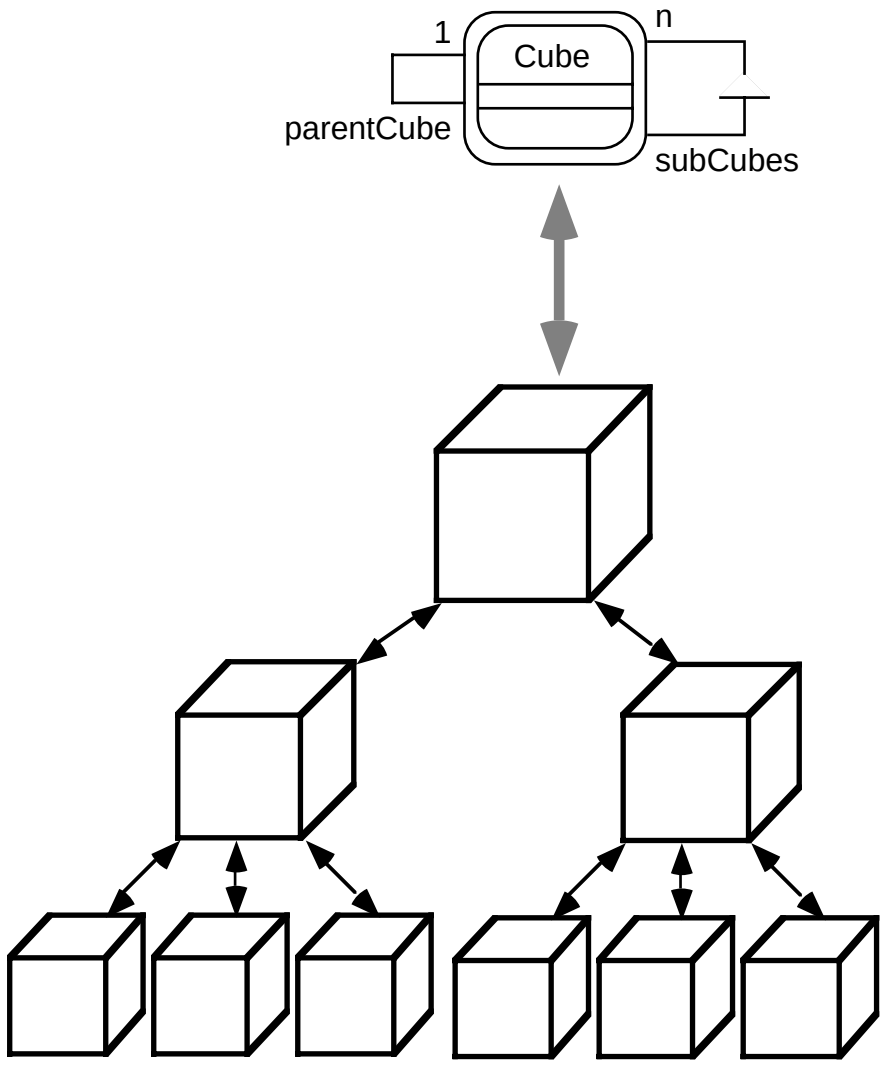

Abbildung 5.4: Würfelhierarchie (Objektnotation nach [CoYo94a])

Beschränkend gilt, daß alle zusätzlichen Dimensionswerte zur selben Dimension gehören, d.h. die Informationsobjekte werden in allen Unterelementen eines Baumelements nach dem gleichen dimensionalen Kriterium aufgeteilt. Die Ele-

mente eines Baums bilden Hierarchiestufen, wobei alle Elemente, die über ihre Dimensionswerte einen Bezug zu den gleichen Dimensionen besitzen, zu einer Hierarchiestufe gehören (siehe Abbildung 5.5).

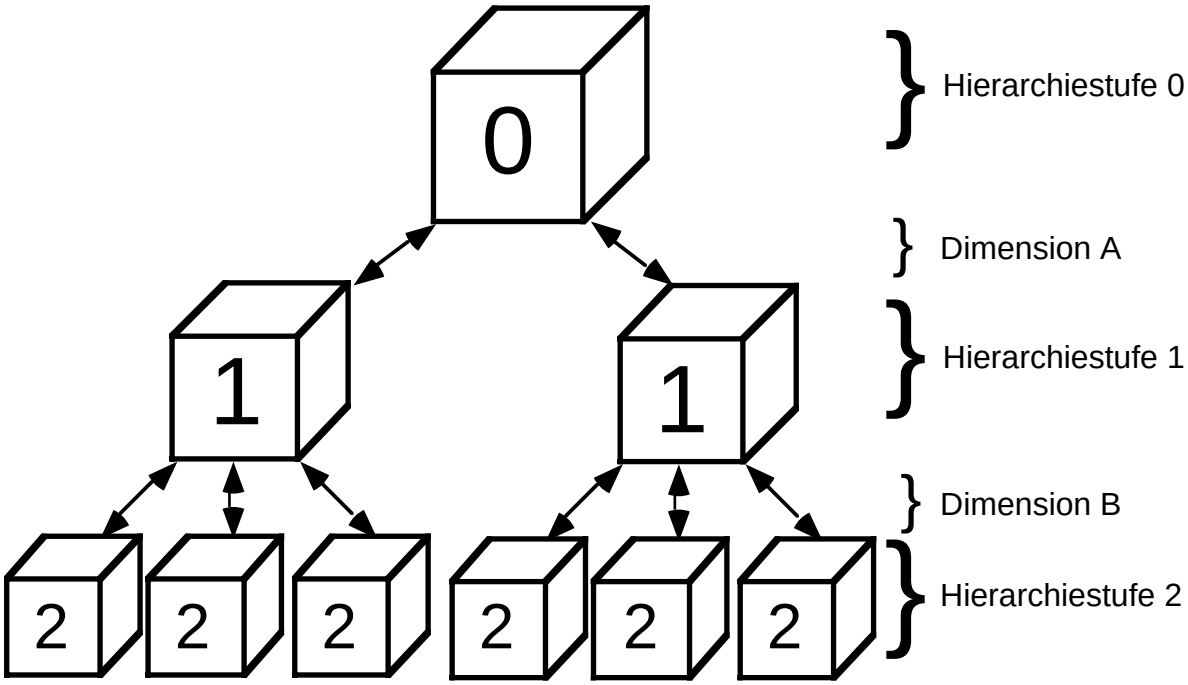

Abbildung 5.5: Dimensionen und Hierarchiestufen

Z.B. sind auf der Hierarchiestufe eins alle Informationsobjekte nach Dimensionswerten der Dimension Auftragsdatum und auf der Hierarchiestufe zwei zusätzlich nach Dimensionswerten der Dimension Produkt organisiert.
Für den Zugriff auf einen dimensionierten Ausschnitt wird ausgehend von dem Wurzelelement top-down entlang des Baums traversiert. Mit jeder tieferen Hierarchiestufe wird die Menge der Informationsobjekte durch die Auswahl eines Elements mit einem zusätzlichen Dimensionswert weiter eingeschränkt. Dieser Vorgang wiederholt sich solange, bis die gewünschte Dimensionierung oder die tiefste Hierarchiestufe erreicht ist. Die Elemente der tiefsten Hierarchiestufe besitzen selbst keine Unterelemente.
Falls der Baum für einen Zugriff nicht passend dimensioniert ist, wird er dynamisch umstrukturiert. Für den Fall, daß der Baum alle für den Zugriff notwendigen Dimensionen enthält, jedoch in einer unpassenden Reihenfolge, werden die Hierarchiestufen umsortiert. Für nicht vorhandene Dimensionen werden zusätzliche Hierarchiestufen gebildet.

Die beschriebene Baumstruktur ist komplex und flexibel. Für die Verwendung im Rahmen des ooMSS stellen die Baumstrukturen eine einfache Schnittstelle zur Verfügung, über die die Konzeptdetails verborgen bleiben. Dies entspricht einem Facade-Pattern, das eine Schnittstelle auf einer hohen Abstraktionsebene zur Verfügung stellt. Hierdurch können Subsysteme einfacher verwendet werden.

6. Interaktionsmodell

In diesem Abschnitt wird das Interaktionsmodell des ooMSS beschrieben. Zu diesem Zweck wird zunächst auf die für das Interaktionsmodell berücksichtigten Graphical-User-Interface-Konzepte (GUI) mit ihrem Bezug zur Objektorientierung und zur Erfüllung der aufgestellten Kriterien eingegangen. Darauf aufbauend wird die Konzeption des Interaktionsmodells durch die Beschreibung des zugrundeliegenden Morph-Elements, die Anwendung des Interaktionsmodells und die Kooperation mit dem Informationsmodell dargestellt.

6.1 Eignung und Historie von Graphical-User-Interface-Konzepten

Das Interaktionsmodell des ooMSS verwendet durchgehend die GUI-Technik. Bei einer GUI wird das Informationsmodell dem Anwender durch eine hochauflösende BitMap-Grafik präsentiert. Diese Basistechnik ermöglicht es, gleichberechtigt Text und Grafik an beliebiger Position auf einem Bildschirm darzustellen. Der Bildschirm ist die „Bühne“ für die Interaktion zwischen dem Anwender und dem Software-System. Der Anwender interagiert mit der GUI vorrangig über Zeigergeräte (i.d.R. Mäuse), durch die sich Bildschirmpositionen direkt ansteuern lassen [Appl87 S.5].

Für die Realisierung einer GUI sind die objektorientierten Konzepte mit ihren Mitteln zur Komplexitätsbewältigung besonders geeignet und zusätzlich können objektorientierte Inhalte über eine GUI besonders gut an die Anwender vermittelt werden[1]. Die Verwendung einer GUI für das ooMSS zur Vermittlung von Objekten an die Anwender ist deshalb vorteilhaft.
Die Objektorientierung und die GUI besitzen dazu eine gemeinsame historische Wurzel. Die grundlegenden Konzepte der heutigen GUI wurden in den siebziger Jahren in Forschungsprojekten des Palo-Alto Research Centers (PARC) entwickelt. Das übergeordnete Thema war die Gestaltung von benutzerzentrierten interaktiven Computersystemen. Der Begriff der Objektorientierung wurde dort von Kay geprägt und als software-technische Basis der Systeme verwendet [Kay93 S.70].

[1] “Contribution of object oriented programming is to make computers think more like people think, GUIs is a good way of allowing people to see their objects and control them.” [GoAl92 S.47].

6.2 Die für das Interaktionsmodell berücksichtigten GUI-Konzepte

6.2.1 Grundlegende GUI-Elemente

Die auf dem Bildschirm einer GUI plazierten GUI-Elemente lassen sich in Anlehnung an [Uhli88, Appl87] grundsätzlich in Fenster und Interaktionselemente (engl. Widgets) unterscheiden.
Die Fenster sind die Arbeitsflächen der Anwendungen. In ihnen werden grafische und alphanumerische Informationen des zugrundeliegenden Informationsmodells visualisiert. Als rechteckige Flächen unterteilen sie den Bildschirm. Da sich die Fenster überlappen können, lassen sich auf dem Bildschirm gleichzeitig eine Vielzahl von Fenstern anordnen, wobei jedes Fenster eine Sichtweise auf das Informationsmodell repräsentiert.
Die Fenster selbst enthalten neben textuellen und grafischen Inhalten eine Menge von Widgets. Alle Interaktionen werden über die Widgets durchgeführt. Sie ermöglichen dem Anwender, Einfluß auf das Modell zu nehmen.

Zu den grundlegenden Widgets zählen:

- Knöpfe (durch ihre Betätigung löst der Anwender eine Aktion aus)
- Menüs (eine vertikal angeordnete Menge von Knöpfen)
- Textfelder (eine rechteckige Fläche, die Text enthält)
- Scroller (durch ihre Positionierung in einem horizontalen oder vertikalen Balken beeinflußt der Anwender, welcher Ausschnitt einer Fläche dargestellt wird)
- Ikonen (Repräsentationen von manipulierbaren Objekten oder Funktionen, die ausgelöst werden können) [Uhli88 S.309]

Zwischen Widgets existieren Kompositionsbeziehungen. Komplex aufgebaute Widgets können sich aus grundlegenden und anderen komplexen Widgets zusammensetzen.

Für die Konzeption des Benutzerparadigmas des ooMSS sind eine Reihe von grundlegenden Prinzipien von GUI relevant, wobei alle einen Bezug zum objektorientierten Paradigma besitzen.

- Eine GUI ist ereignis-gesteuert, d.h. die GUI wird durch Ereignisse, die hauptsächlich der Anwender erzeugt, gesteuert. Der Anwender interagiert mit den GUI-Elementen und löst damit eine Aktion aus. Das System verhält sich dabei reaktiv. Der Anwender initiiert und steuert die Aktionen.

- Die durch den Anwender ausgelösten Vorgänge basieren auf der noun-verb-syntax [Niel93 S.84, Appl87 S.5]. Der Anwender wählt zunächst über ein GUI-Element das Modellelement, auf das er einwirken will und dann die Operation, die das Element ausführen soll. Das System unterstützt den Anwender bei der Auswahl der sinnvollen Operationen, abhängig von der Art und dem Zustand des Elements. Der Anwender besitzt einen großen Freiraum im Umgang mit dem Software-System, bspw. welche Daten wie und in welcher Reihenfolge abgefragt und verarbeitet werden sollen.

- Auf der Basis von Fenstern soll auf einfache Weise zwischen Modi gewechselt werden. Ein Modus ist der Kontext, in dem Aktionen des Anwenders interpretiert werden [Appl87 S.12]. Aus den gleichen Aktionen können sich in verschiedenen Modi verschiedene Reaktionen ergeben. Modi als feste und notwendige Bestandteile einer Anwendung beschreiben den für den Benutzer bereitgestellten Handlungsspielraum [vgl. Appl87 S.12]. Z.B. können in einem Abfragemodus Informationsabfragen an das System formuliert und in einem Löschmodus Informationen gelöscht werden.
  Wenn es dem Anwender möglich ist, auf einfache Weise die Modi zu wechseln, dann stehen ihm scheinbar alle Möglichkeiten des Systems gleichzeitig zur Verfügung und die mit den Modi verbundenen Beschränkungen können weitestgehend vermindert werden. Im Rahmen einer integrierten Umgebung werden deshalb eine Menge von Fenster-Sichten präsentiert, die jeweils einen Modus darstellen. Durch eine einfache Auswahl einer Fenster-Sicht wechselt der Anwender den Modus [Tesl81 S.94].

- Zur Unterstützung der Benutzerfreundlichkeit wird bei den GUI auf die Erfahrungen, die der Anwender mit physikalischen Objekten in der realen Welt gesammelt hat, zurückgegriffen. Durch die Übertragung von ausgewählten Eigenschaften in die GUI soll eine größere Deckung mit dem mentalen Modell des Anwenders erreicht werden.
  Äußerlich erkennbar ist dies an den Metaphern der realen Welt, die bei der Gestaltung der Widgets verwendet werden (Fenster, Knöpfe, Menüs). Zusätzlich kann der Anwender das System direkt steuern und erhält eine Rückkopplung. Z.B. blinken Menüs, Knöpfe erhellen sich, wodurch der Eindruck entsteht, daß sie eingedrückt werden, Text wird beim Selektieren invers dargestellt. Bei längeren Operationen zeigt ein Dialog den Fortschritt an.

Sowohl mit den Metaphern als auch mit der Rückkopplung ist der Anwender vertraut. Bemerkbar ist hier der Einfluß der Objektorientierung mit ihren Ursprüngen im Bereich der Simulation von physikalischen Systemen (siehe Abschnitt 3.2).

6.2.2 Eine objektorientierte GUI

Trotz der Verwendung von objektorientierten Eigenschaften ist eine GUI nicht notwendigerweise objektorientiert[2]. Der fehlende hinreichende Bestandteil ist das Modell, das dem Anwender über die GUI vermittelt wird. Dieses Modell besitzt die größte Bedeutung für die Verwendbarkeit eines Software-Systems. Das durch die GUI bereitgestellte “look and feel” ist weniger wichtig. Es besitzt mehr einen unterstützenden Charakter [RBIM98 S.28, 35].
Besitzt das vermittelte Modell objektorientierte Eigenschaften, dann ist auch die GUI objektorientiert. Ein solches konzeptionelles Modell besteht aus Objekten, Eigenschaften, Verhalten und Beziehungen [RBIM98 S.26]. Ein Objekt wird mit seinen Eigenschaften als Einheit repräsentiert. Zu den Eigenschaften des Objekts zählen alle seine Interaktionsmöglichkeiten, die dem Anwender zur Verfügung gestellt werden. Zusätzlich unterstützt eine objektorientierte GUI die Benutzerfreundlichkeit, da es um Objekte aus dem Domänenbereich des Anwenders zentriert ist [RBIM98 S.11].

Das Gegenteil einer objektorientierten GUI ist eine GUI, die Informationstechnologie-Konzepte visualisiert, die nicht objektorientiert sind. Bspw. wird über die GUI eine Domäne entweder nur in Form von in Relationen abgelegten Daten oder durch eine Auflistung der dem System zugrundeliegenden Funktionen vermittelt[3]. Durch den Verzicht auf die Informationstechnologie-Konzepte muß der Anwender mit weniger Konzepten umgehen und kann sich so auf seine Domänenkonzepte konzentrieren [RBIM98 S.27].
Zu beachten ist, daß das Informationsmodell, welches eine objektorientierte GUI zugrundeliegt, selbst nicht zwangsläufig objektorientiert sein muß. Allerdings bietet sich eine objektorientierte Basis an, da die Informationsobjekte mit ihren Eigenschaften eins zu eins auf dem Bildschirm repräsentiert werden können. Bei einer nicht objektorientierte Basis existiert ein impedance mismatch zwischen dem prozeduralen oder datenorientierten Informationsmodell und der objektorientierten GUI. Dieser Bruch muß mit einer umfangreichen Transformation überwunden werden.

[2] anders siehe [Niel93 S.84]

[3] Für den Verzicht von Informationstechnologie-Konzepten gibt es allerdings die Ausnahme, daß die Domäne selbst aus Informationstechnologie-Konzepten besteht: Im Rahmen von objektorientierten GUI von Betriebssystemen, wie z.B. Windows oder MacOS werden Dateien, Applikationen und Verzeichnisse repräsentiert.

6.2.3 Direct Manipulation Interfaces

Durch die Direct Manipulation Interfaces wird der Ansatz, Erfahrungen des Anwenders mit physikalischen Systemen für die GUI Benutzerparadigma zu nutzen, konsequent weitergeführt. Eine Direct Manipulation Interface ist eine Schnittstelle, die zusätzlich darauf basiert, den Zustand der am Bildschirm präsentierten Objekte direkt, d.h. durch die Aktionen an GUI-Elementen und nicht durch eine textuelle Beschreibung, wie z.B. eine Kommandosprache, zu verändern [Shne83, HuHN86]. Alle Objekte, die für ein Anwendungsgebiet von Interesse sind, werden auch am Bildschirm sichtbar und können durch inkrementelle direkte Aktionen verändert werden. Der Anwender bekommt auf seine Aktionen eine direkte Rückkopplung. Die Aktionen können rückgängig gemacht werden und sind immer, im Gegensatz zu einer in einer Kommandosprache formulierten komplexen Befehlsfolge, syntaktisch korrekt.
In Erweiterung zu den grundlegenden GUI-Konzepten gewinnt der Anwender den Eindruck, mit den zugrundeliegenden Objekten direkt zu interagieren [vgl. HuHN86 S.93f.]. Die Schnittstelle soll vom Anwender als solche nicht wahrgenommen werden. Für den Anwender sind die Repräsentationen die Objekte selbst. Die Schnittstelle wird für ihn damit unsichtbar.

Die grafische Darstellung ist damit eine semantische Repräsentation des zugrundeliegenden Objekts, ein Visueller Formalismus [Hare88]. Das bedeutet, daß die auf dem Bildschirm gezeichneten Elemente mehr als konventionelle Widgets sind. Sie besitzen zusätzlich eine Bedeutung für den Anwendungsbereich. Die Widgets bilden mit dem zugrundeliegenden Domänenmodellausschnitt, dessen aktuellen Zustand sie visualisieren, eine Einheit. Wenn der Anwender mit einer solchen Manipulationseinheit interagiert, wird gleichzeitig das zugrundeliegende Objekt modifiziert.
Die Zuordnung zwischen Repräsentation und Objekt ist dabei eindeutig. Die Repräsentation wird immer mit einem Objekt verbunden. Umgekehrt erscheint die Repräsentation eines Objekts niemals mehrfach in der Umgebung. Das Objekt besitzt mit dem zentralen Ort der Existenz eine ein-eindeutige Identität im System, vergleichbar mit physikalischen Objekten.
Eine weitere Eigenschaft aus der physikalischen Realität ist der Verzicht auf abstrakte Objekte. Da die Objekte nur durch konkrete Einheiten vermittelt werden, vereinfacht sich die Anwendung.
Durch diese Eigenschaften unterstützt eine Direct Manipulation Interface die Bildung eines geeigneten mentalen Modells, womit der Anwender aus der physikalischen Realität vertraut ist.
Ein Beispiel für eine sehr verbreitete Direct Manipulation Interface sind GUI von Betriebssystemen. Dateien werden als Ikonen, Verzeichnisse zusätzlich als Fenster repräsentiert. Ein Verzeichnis-Fenster enthält Verzeichnisse und sonstige

Dateien in Form von Ikonen, die über die Direct Manipulation Interface manipuliert werden können. Der Anwender bewegt dazu die Ikonen zwischen den Fenstern, was auf der zugrundeliegenden technischen Ebene durch Kopiervorgänge oder Verschiebungen umgesetzt wird.

6.2.4 People Places Things-Konzept der Arbeitsplatzumgebung

Eine weitere nützliche Komponente für das ooMSS Interaktionsmodell ist das People Places Things-Konzepts von Taligent. Es handelt sich um ein konzeptionelles Modell, durch das beschrieben wird, wie Domänenobjekte im Rahmen der Interaktion präsentiert werden [CoPo95]. Ursprung des Konzepts ist das Benutzermodell der von IBM entwickelten Richtlinien der einheitlichen Benutzerunterstützung (Common User Access) [ArHS92 S.143f.]. Dem Anwender des Software-Systems wird eine (virtuelle) Arbeitsplatzumgebung vermittelt [ArHS92 S.138].}
Das People Places Things-Konzept löst den applikations- und dokumentenzentrierten Ansatz, der mit dem Desktop-Paradigma verbunden ist, ab, bei dem der Anwender seine Aufgaben durch den Umgang mit Informationstechnologie-Konzepten bewältigt, indem er z.B. Applikationen startet und Dateien öffnet [CoPo95 S.85].
Es wird mit Einheiten umgegangen, die physisch und konzeptuell in der realen Welt des Anwenders im Rahmen seiner Aktivitäten auftreten. Diese Objekte (Things) werden in Umgebungen, den Places, plaziert. Sie stellen eine Unterstützungsumgebung für einen speziellen fachlichen Zusammenhang der Domäne dar. Objekte werden je nach Anwender und Situation angeordnet und organisiert. Die räumliche Anordnung der Objekte reflektiert die semantische Struktur des Systems, ähnlich wie bei visuellen Sprachsystemen [Resu95 S.19]. Es wird für eine Umgebung festgelegt, welches relevante Objekt mit welchen Eigenschaften wie dargestellt wird. Dem Anwender ist es möglich, die Repräsentation nach seinem mentalen Modell von der Domäne auszurichten. Dadurch kann er die mit der Situation verbundenen Objekte besser verstehen und mit ihnen umgehen [Prac90 S.19].
Die Ansammlung von Objekten und ihre Repräsentationen decken die Funktionalität der Applikationen von traditionellen Ansätzen ab. Dort setzen sich Software-Systeme aus Informationstechnologie-Konzepten, wie Applikationen und Dokumenten, zusammen. Daß diese technische Sichtweise verlassen wird, unterstützt das Natürlichkeitskriterium.

6.3 Das Morphic-Framework als Basis des Interaktionsmodells

Das GUI-Framework Morphic [Malo95] wurde als Grundlage des Interaktionsmodells gewählt, weil es die technische Basis der zuvor beschriebenen GUI-Konzepte zu einem hohen Grad abdeckt. Erkennbar ist dies an der durch Maloney beschriebenen Zielsetzung von Morphic: Es ermöglicht die einfache Erstellung und Veränderung von interaktiven grafischen Objekten durch direkte Manipulation und programmierte Steuerung. Mit dieser technischen Basis im Bereich der grundlegenden GUI- und Direct Manipulation Interface-Elemente können auf einfache Weise die inhaltlichen GUI-Konzepte, also die objektorientierte GUI mit dem People Places Things-Konzept, ausgestaltet werden.
Die GUI-Grundbausteine, die Morphic zur Verfügung stellt, sind die grafischen Morph-Elemente (gr. Gestalt, Form, Aussehen). Jeder Morph ist ein vollwertiges Objekt. Die mit Morphic erstellten GUI setzen sich komplett durch Komposition aus Morph-Objekten zusammen. Alle sichtbaren Elemente des Interaktionsmodells bestehen einheitlich aus Morphs.
Jedes Informationsobjekt im ooMSS wird mit seinen gesamten Eigenschaften durch einen speziell entwickelten Morph, dem Informationsmorph, repräsentiert (siehe Abbildung 6.1).

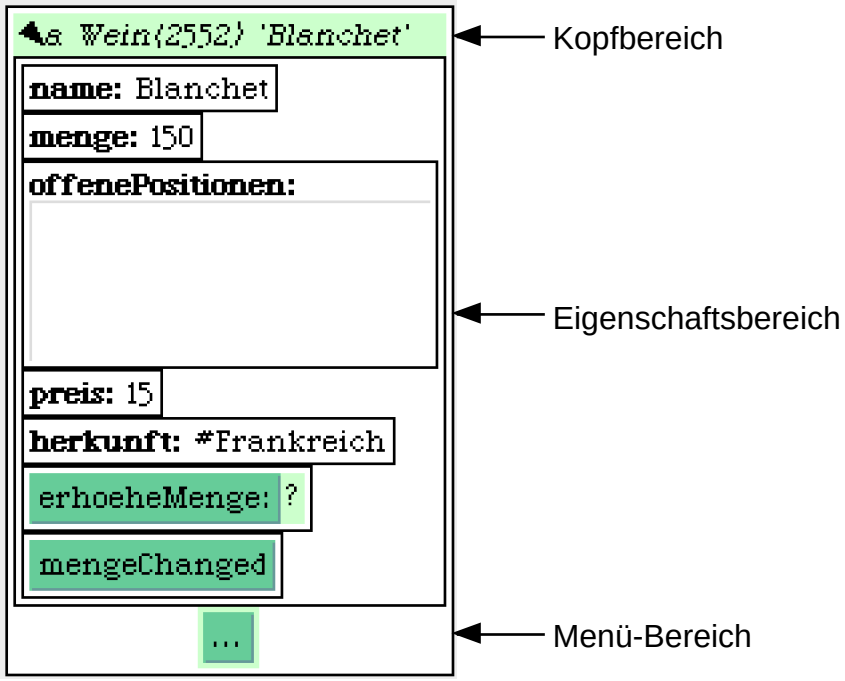

Abbildung 6.1: Beispiel eines Informationsmorph

Ein Informationsmorph besitzt dazu drei Bereiche:

- Der obere Kopfbereich enthält die Bezeichnung seines Informationsobjekts. Die Bezeichnung ist im einfachsten Fall der Klassenname des Domänenobjekts und die eindeutige Objekt-ID der Instanz.
- Im mittleren Eigenschaftsbereich werden die Eigenschaften durch Attribut- und Methoden-Morphs repräsentiert. Die Eigenschaften-Morphs sind standardmäßig vertikal untereinander angeordnet.
- Im unteren Menü-Bereich besitzt ein Informationmorph einen Menü-Knopf. Mit der Betätigung des Knopfs wird ein Menü erzeugt, das einheitlich für alle Informationsobjekte Aktionen zur Verwaltung der Fenster- und Informationsobjekt-Eigenschaften enthält.

Morphs besitzen standardmäßig Direct Manipulation Interface-Eigenschaften und ein physikalisches “look and feel”. Dazu zählt u.a., daß ein Morph vom Anwender über den Mauszeiger aufgehoben und bewegt werden kann, wobei er auf dem Bildschirm einen Schatten wirft, und überlappende Morphs einen Eindruck von räumlicher Tiefe vermitteln (siehe Abbildung 6.2).

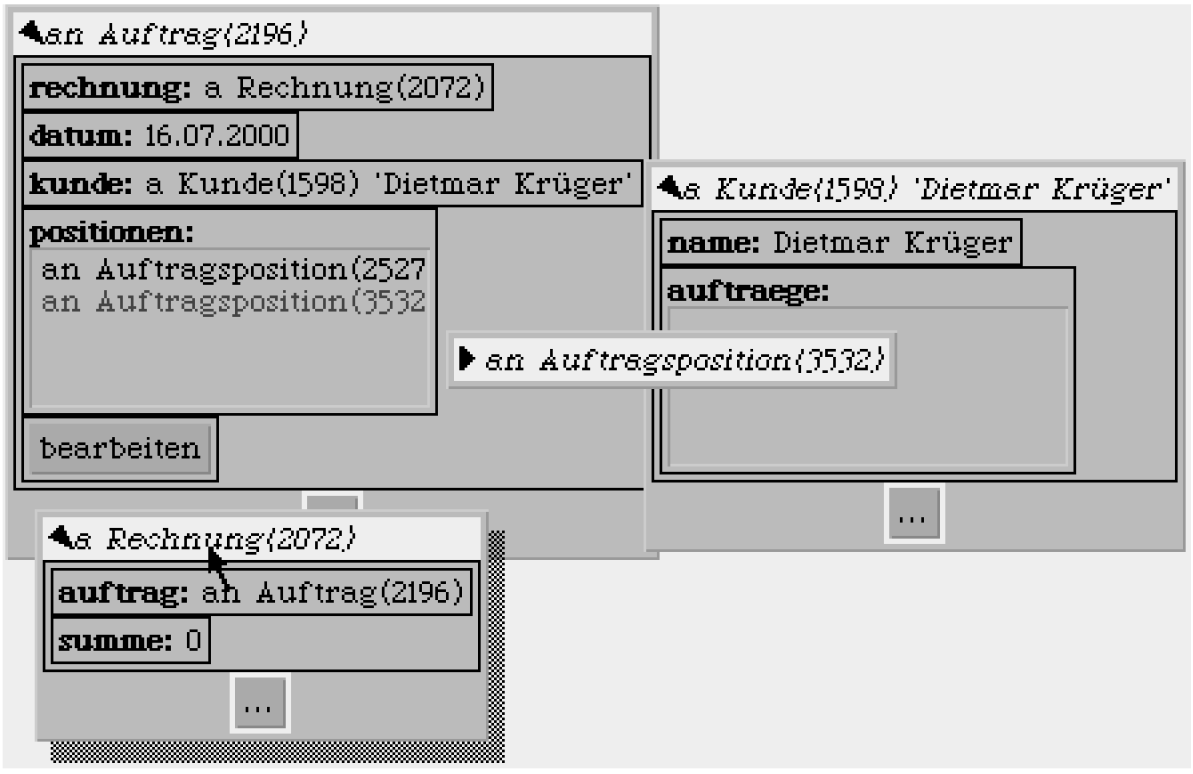

Abbildung 6.2: Physikalisches “look and feel” des Morph-GUI

Jeder Morph ist direkt oder indirekt Bestandteil einer MorphWelt, die nach der People Places Things-Metapher mit ihren enthaltenen Morphs für den Anwender eine Unterstützungsumgebung darstellen.

6.4 Anwendung des Interaktionsmodells

6.4.1 Erzeugung einer Unterstützungsumgebung durch die Einrichtung einer MorphWelt

Das Interaktionsmodell ermöglicht dem Anwender, abhängig von der Unterstützungssituation, selbständig Sichten auf das Informationsmodell einzurichten. Im Rahmen einer Sicht erfährt der Anwender, welche Informationsobjekte existieren, welche Eigenschaften sie besitzen und in welchen Zusammenhängen sie stehen. Die Detaillierungsstufen sind dabei wählbar.
Die für eine Unterstützungssituation relevanten Objekte werden Bestandteil der Sicht. Für jede Sicht legt er fest, aus welchen Informationsobjekten sie bestehen soll, welche Eigenschaften der Informationsobjekte für die Sicht relevant sind und wie die Eigenschaften dargestellt werden sollen. Für jedes Informationsobjekt, das Bestandteil einer Umgebung sein soll, wird ein eigener Informationsmorph erzeugt und in der Umgebung plaziert.

Für die Auswahl der Eigenschaften wählt der Anwender für jeden Informationsobjekttypen bei den Informationsmorphs aus, welche Attribute und Methoden umgebungsspezifisch im mittleren Eigenschaftsbereich visualisiert werden sollen. Dieser Vorgang wird, wie in der Abbildung 6.3 zu sehen ist, über den Menü-Bereich gesteuert.

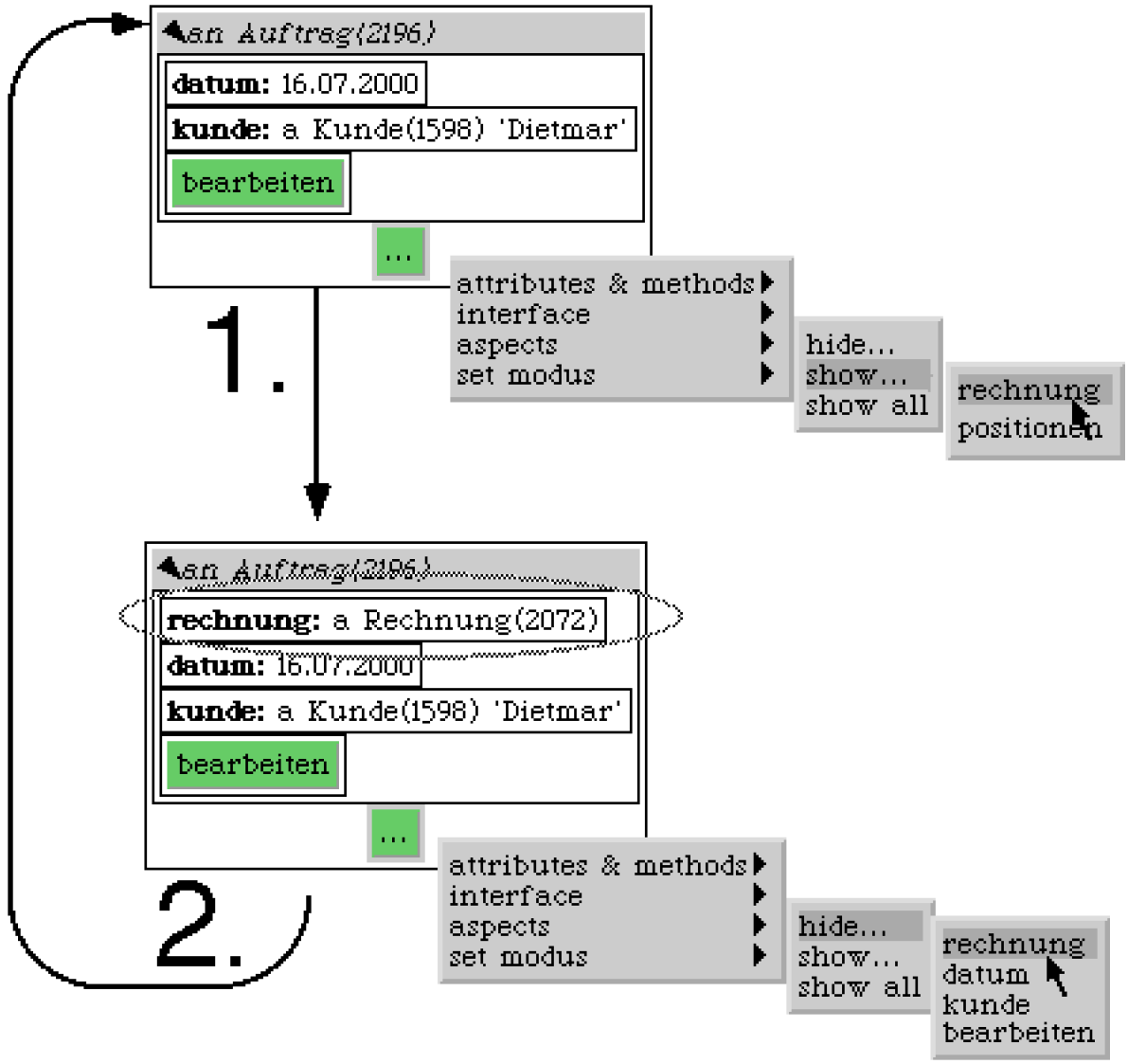

Abbildung 6.3: Menge der visualisierten Eigenschaften ändern

Alternativ zu der Auswahl von einzelnen Eigenschaften kann der Anwender die in einem Fenster angezeigten Eigenschaften durch die Betätigung des Button-Morph, der im Kopfbereich links von der Bezeichnung steht und mit einem Dreieck versehen ist, beeinflussen. Durch die Betätigung des Knopfs wechselt die Darstellung des Fensters zwischen einer platzsparenden Repräsentationsform, bei der nur der Kopfbereich angezeigt wird und der Vollanzeige (siehe Abbildung 6.4).

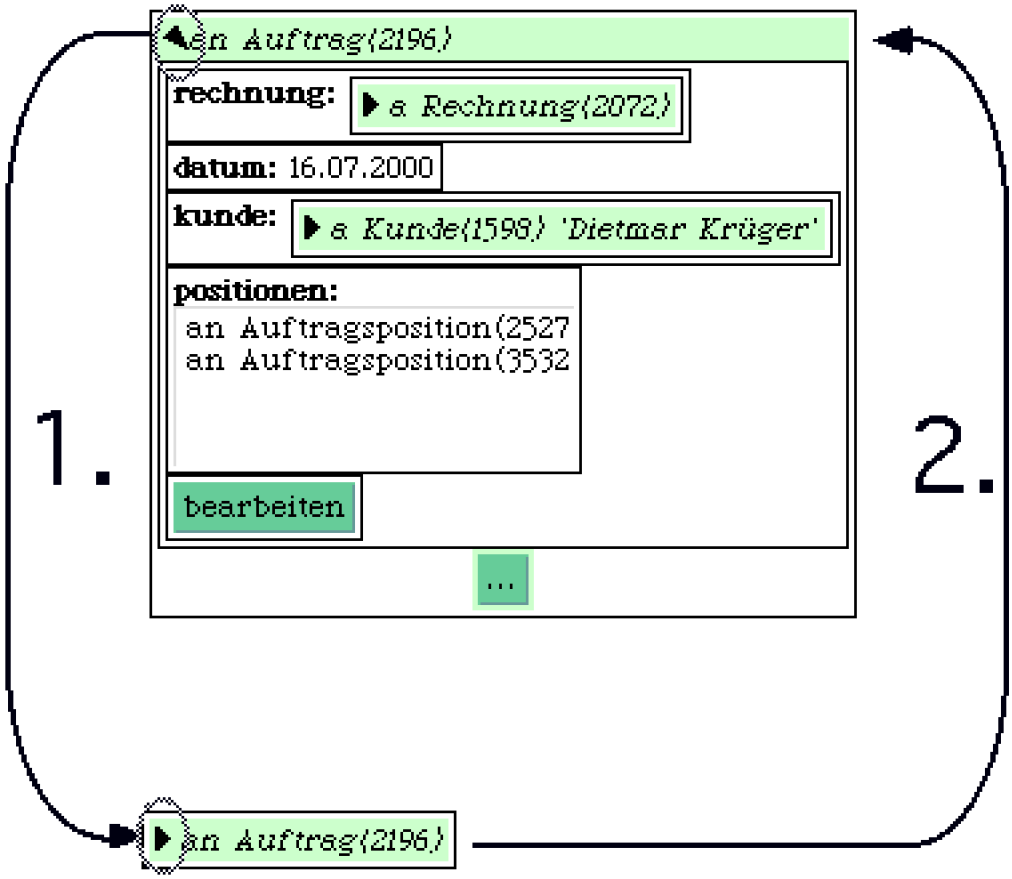

Abbildung 6.4: Darstellungswechsel eines Informationsmorph

Diese Funktionalität, bei der ein Informationsmorph „zusammengeklappt“ wird, unterstützt die Übersicht des Anwenders. Gut designte objektorientierte Systeme bestehen im Sinne der Wiederverwendbarkeit tendenziell aus vielen kleineren statt wenigen großen Objekten. Der Anwender soll nicht mit für ihn unwichtigen Details konfrontiert werden. Er kann sie aber bei Bedarf, wenn er auf ein Objekt einwirken möchte, auf einfache Weise erhalten.

Die an einem Informationsmorph ausgewählten Eigenschaften werden in der Standard-Repräsentationsform vertikal untereinander angeordnet. Ein Informationsmorph besitzt mit dieser Outliner-Anwendungsforms eine einfache generische Darstellungsweise. Die Eigenschaften werden, unabhängig von ihrer fachlichen Bedeutung, gleichberechtigt aufgelistet.
In dieser Anwendungsform ist das Informationsmorph eine Weiterentwicklung des Inspektorwerkzeugs unter Smalltalk, womit ein Objekt mit seiner Attributsstruktur und seinen Attributwerten repräsentiert wird. Zusätzlich enthält es die Methoden, ähnlich wie das Browser-Werkzeug der objektorientierten Self-Entwicklungsumgebung [SmMU95 S.50].

Das Anwendungsparadigma wird im Rahmen der Arbeit um die Möglichkeit ergänzt, abhängig vom Objekttyp eine alternative Darstellungsform zu wählen. Dabei werden die Attribut- und Methoden-Morphs innerhalb des mittleren Bereichs frei angeordnet und mit Texten und Grafiken ergänzt. Der Anwender kann mit dieser zusätzlichen Flexibilität individuell und problembezogen die Sichtweise auf einen Objekttypen anpassen. Durch diese frei gestaltbare Anwendungsform (`FreeForm`) wird die mit einem Objekttypen inhaltlich verbundene Metapher zusätzlich visuell unterstützt.
Z.B. kann ein Kunde durch eine Kundenkartei repräsentiert werden und eine Rechnung durch ein Rechnungsformular.
Das aktuell dargestellte Layout (was und wie dargestellt wird) kann als Standardlayout gespeichert werden[4]. Innerhalb eines Informationsmorph lassen sich für komplexe Attributwerte neben ihrer Position auch alternative Darstellungsformen wählen, wodurch zusätzlich die Komplexitätsbewältigung unterstützt wird [vgl. Born79]. Z.B. können Mengen von Objekten durch Listen und n-dimensionale Objektstrukturen durch Tabellen oder Balkendiagramme visualisiert werden.

6.4.2 Morph-basierte Interaktion mit dem Informationsmodell

Im folgenden Abschnitt wird beschrieben, wie der Anwender mit den durch die Morphs repräsentierten Attributen und Methoden umgeht. Zur Illustration werden Beispiele der operativen Ebene verwendet.
Wie schon beschrieben, wird der selektierte Ausschnitt des Informationsmodells über die Interaktionselemente in seinem aktuellen Zustand angezeigt. Über die Attribut- und Methoden-Morphs interagiert der Anwender mit den Eigenschaften der visualisierten Informationsobjekte. Er kann entlang der Objektbeziehungen navigieren, um das Informationsmodell zu analysieren und es beeinflussen, indem er Attributwerte ändert und Methoden aktiviert.
Ein Attribut-Morph repräsentiert das Attribut des Informationsobjekts durch den Attributnamen und den aktuellen Attributwert. Ist der hinter dem Attribut verwaltete Wert atomar, also z.B. ein alphanumerischer Wert, dann wird der Attributwert durch eine Zeichenkette repräsentiert, die dem Wert entspricht. Für komplexe Attributwerte, die im ooMSS einheitlich als Instanzen einer Unter-

[4]Beeinflußt wurde die Anwendungsform durch die Funktionalität und das Anwendungsparadigma von Hypermedia-Werkzeugumgebungen, wie z.B. HyperCard [Will87]. Bei den dort verwendeten Repräsentationsformen werden auf einfache Art die Schnittstellen-Komponenten, wie Datenfelder, Aktionsknöpfe und Grafiken, erstellt und plaziert. Jederzeit können die entstandenen Anwendungen ausprobiert und gegebenfalls angepaßt werden. Die damit verbundene Vorgehensweise zur Entwicklung eines Software-Systems ist iterativ und inkrementell. Da es eine sehr einfache Möglichkeit ist, Oberflächen zu erzeugen, wird dies im Rahmen des Prototypings auch für den Bereich der objektorientierten Prototypen als einfacher Interface Builder verwendet [CoSh95]. Für die Einrichtung und Anpassung des ooMSS sollte diese Vorgehensweise genutzt werden.

klasse von `InformationObject` angehören, wird die Bezeichnung des verwalteten Informationsobjekts angezeigt. Von der Art des Attributwertes ist auch abhängig, wie dieser durch die Interaktion mit dem Morph geändert wird. Für die Änderung eines atomaren Attributwerts aktiviert der Anwender einfach das Attribut-Morph, woraufhin er über einen Dialog aufgefordert wird, einen neuen Wert einzugeben.
Da ein komplexes Informationsobjekt mit seinen Eigenschaften nicht einfach textuell beschrieben werden kann, besitzt ein Attribut-Morph für einen komplexen Attributwert ein abweichendes Interaktionsverhalten. Durch die Aktivierung des AttributeMorph wird der Informationsmorph von dem als Attributwert verwalteten Informationsobjekt rechts vom Attribut-Morph angezeigt und die Assoziationsbeziehung so verdeutlicht (siehe Abbildung 6.5).

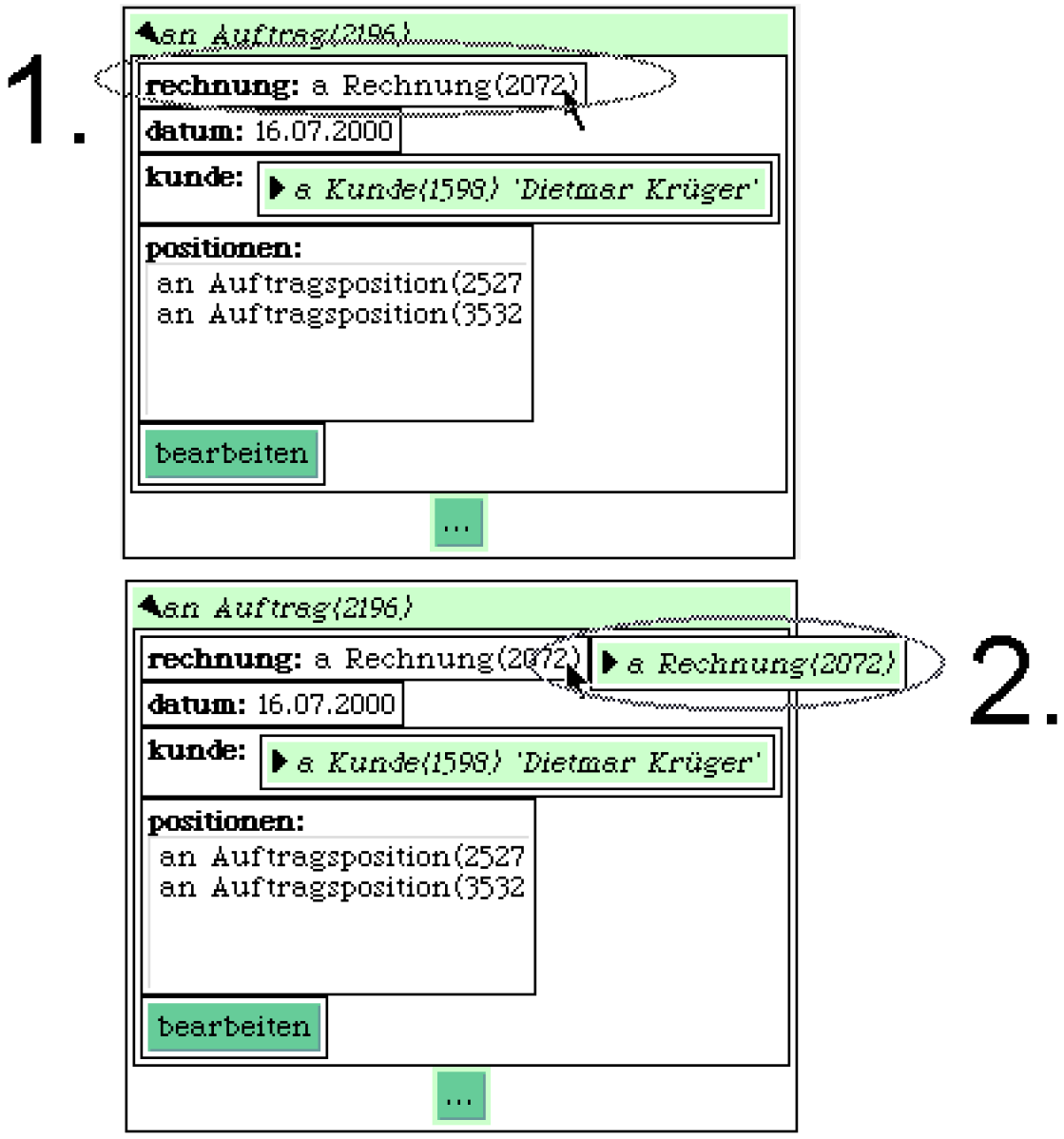

Abbildung 6.5: Zugriff auf einen komplexen Attributwert

Wurde dieses Informationsobjekt noch nicht durch ein Morph-Fenster repräsentiert, wird ein neues Fenster erzeugt. Existierte es schon auf dem Bildschirm, wird lediglich seine Lage verändert. Damit erfüllt sich die Anforderung nach Identität zwischen Repräsentation und Objekt. Der Anwender navigiert durch die komplexen Informationsobjekte.
Er erkennt bei der Navigation z.B., daß zu Aufträgen als Eigenschaft ihre Produkte gehören. Dieser erfaßte Zusammenhang ist für eine Auswertung ausreichend. Genauer ist erkennbar, daß zwischen Aufträgen und Produkten Auftragspositionen stehen, die für jede Verbindung zusätzliche Attribute verwalten. Über das Interaktionsmodell wird unterstützend vermittelt, daß die Produkte von den Aufträgen und ihren Auftragspositionen identisch sind.
Komplexe Attributwertänderungen, also die Verwaltung der Assoziation, erfolgen über die GUI manuell durch drag & drop. Ein Informationsmorph wird dazu auf den Attribut-Morph fallengelassen, dessen Attributwert geändert werden soll. Bei einem singulären Attribut ersetzt der neue komplexe Attributwert den alten. Bei einem multiplen Attribut wird das fallengelassene Informationsobjekt zu der Liste hinzugefügt und so die Kompositionsbeziehung zwischen den Informationsobjekten verwaltet. Aus der Repräsentation der Liste lassen sich zusätzlich einzelne Informationselemente herausgreifen oder löschen.

Ein Methoden-Morph enthält als festen Bestandteil einen Morph-Button. Der Methodenknopf ist mit dem Namen der Methode bezeichnet. Wird der Knopf betätigt, wird die gleichnamige Methode des repräsentierten Objekts aktiviert. Hat die Methode einen Parameter, wird dieser rechts vom Button-Morph angezeigt. Der Anwender interagiert mit ihm wie mit einem Attribut-Morph. Atomare Objekte werden direkt eingegeben. Verweise auf komplexe Objekte werden durch das drag and drop ihrer Morphs hergestellt.

6.4.3 Verwaltung des Informationsmodells über die Informationsmorphs

Dem Entwickler ermöglicht das Interaktionsmodell zusätzlich die Anpassung des Informationsmodells in Bezug auf die Klasseneigenschaften. Er verwaltet die Eigenschaftsstrukturen der bestehenden Klassen und legt neue Klassen an. Dazu werden zusätzlich zu den Informationsobjekten auch ihre Klassen als Informationsmorphs im ooMSS repräsentiert (siehe Abbildung 6.6).

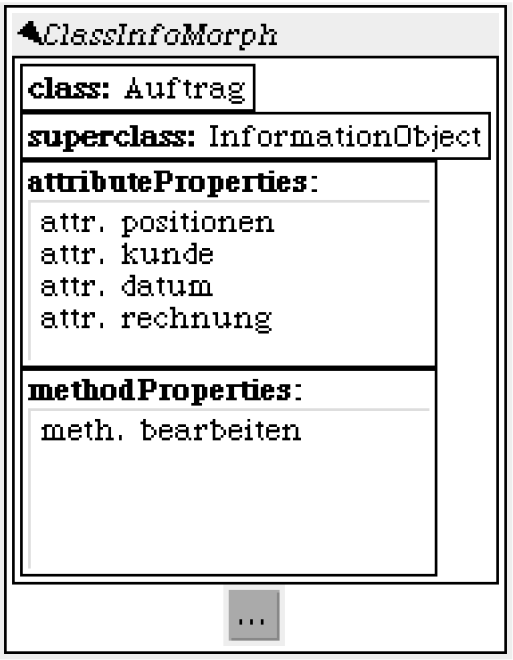

Abbildung 6.6: Aufbau eines Klasseninformationsmorph

Ein Klasseninformationsmorph repräsentiert die Definition einer Informationsklasse durch die Menge aller Attribut- und Methodenbeschreibungen und einen Verweis auf die Oberklasse.
Die Beschreibung der Attribute und Methoden wird jeweils als Objekttyp Eigenschaft (`Property`) abgebildet. Eine Attributeigenschaft besitzt den Namen des Attributs und seine Kardinalität. Eine Methodeneigenschaft verfügt über einen Namen und einen Methodenrumpf[5].
Neue Eigenschaften werden erzeugt, indem der Entwickler ein Eigenschaftsobjekt erzeugt, seine Eigenschaftswerte spezifiziert und es per drag & drop in die Listen des Klasseninformationsmorph bewegt. Nach dem Erzeugen kann die neue Eigenschaft an den Informationsfenstern der Klasse sichtbar gemacht werden.
Bestehende Eigenschaften werden gelöscht, indem sie aus den Attribut- und Methodenlisten entfernt werden. Beim Löschen wird die Konsistenz des Interaktionsmodells gewährleistet, indem aus allen Informationsmorphs, die ein Informationsobjekt der Klasse zusammen mit der Eigenschaft anzeigen, die Eigenschaft entfernt wird. Geändert wird eine Eigenschaft per drag & drop seines modifizierten Eigenschaftsobjekts auf den Klasseninformationsmorph.
Durch die Aktivierung des Oberklassen-Attributs wird der Informationsmorph der Oberklasse in der MorphWelt angezeigt. Die hierarchische Vererbungsbeziehung läßt sich durch den Austausch der Oberklasse verändern. Zwischen den Klasseninformationsmorphs lassen sich Eigenschaften per drag & drop bewegen. Objekteigenschaften lassen sich so zu der passenden Hierarchiestufe zuordnen.

[5]An dieser Stelle der Arbeit wird zunächst nur der Objekttyp beschrieben. Die eigentliche Umsetzung im Rahmen einer Klasse folgt im Abschnitt „MSS-Model“.

Beispiel: Wie in Abschnitt 3.1.4 beschrieben, sind die Ähnlichkeiten von Klassen möglichst weit oben in der Vererbungshierarchie einzuordnen. Die Klassen `Person` und `Produkt` sind beide Unterklassen von `InformationObject` und besitzen jeweils eigenständig das Attribut `name`, in dem eine Zeichenkette verwaltet wird. Es kann jetzt eine neue Klasse `NamedObject` definiert werden, die dieses Attribut enthält und als Oberklasse von `Person` und `Produkt` dient. Soweit wie möglich wird versucht, beim Verschieben der Attribute innerhalb der Klasseninformationsmorphs die Attributwerte der Instanzen der beteiligten Klassen zu erhalten, hier die Namen der Personen und der Produkte.
Neben der Verwaltung von bestehenden Klassen kann der Entwickler durch einen speziellen Morph zusätzlich neue Klassen erzeugen. Der `ClassCreator-Morph` besitzt als visualisierte Eigenschaft den Namen der Oberklasse und den Namen der neuen Klasse. Durch die Aktivierung des create-Knopfs werden die neue Klasse und ihr Informationsmorph erzeugt (siehe Abbildung 6.7).

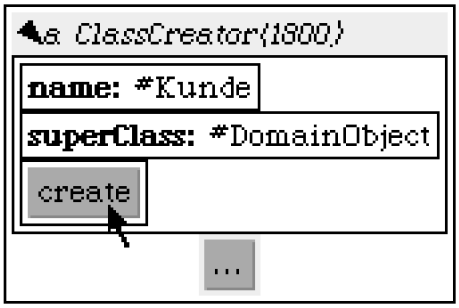

Abbildung 6.7: Aufbau eines ClassCreatorMorph

6.5 Kooperation zwischen Informations- und Interaktionsmodell

Wie in der Architekturübersicht in Abschnitt 4.1 beschrieben wurde, ist die Kooperation der beiden ooMSS-Modellbereiche dadurch geprägt, daß einseitig vom Interaktionsmodell eine Beziehung zu dem Informationsmodell besteht. Damit verwaltet jeder Informationsmorph eine einseitige Assoziation zu dem Informationsobjekt, das er repräsentiert (siehe Abbildung 6.8).

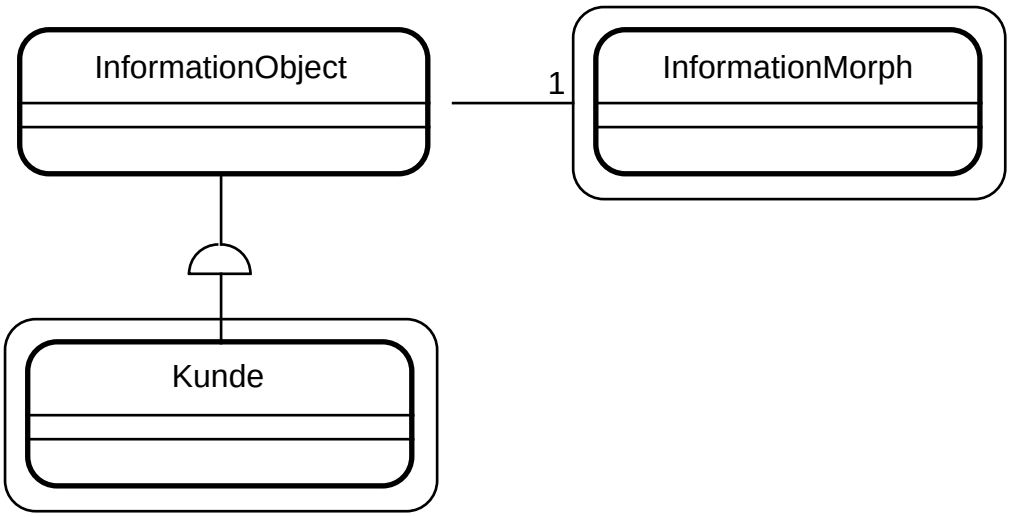

Abbildung 6.8: Objektmodell mit InformationObject und InformationsMorph (Objektnotation nach [CoYo94a])

Bei der Erzeugung eines Morph wird das Informationsobjekt übergeben, welches visualisiert werden soll. Der Morph erzeugt für jede Eigenschaft seines Informationsobjekts einen Eigenschaftsmorph. Der Morph nutzt dazu das Selbstbeschreibungsprotokoll des Informationsobjekts. Es gibt Auskunft über alle Attribute und Methoden.
Nach der Einrichtung muß der Morph seine Darstellung aktualisieren. Aufgrund der einseitigen Beziehung kann der Morph nicht von seinem Informationsobjekt auf Änderungen, die durch Nachrichtenflüsse innerhalb des Informationsmodells entstehen, hingewiesen werden. Der Morph muß selbst aktiv regelmäßig die aktuellen Attributwerte bei seinem Informationsobjekt erfragen. Informationsobjekte antworten mit ihren Attributwerten, als Bestandteil ihres im Informationsmodell festgelegten Standardverhaltens (siehe Informationsmodell).

Ein Benachrichtigungsmechanismus muß dafür sorgen, daß nach der Einrichtung ein Morph die Attributwerte seines Informationsobjekts aktuell darstellt. Weil nur der Morph sein Informationsobjekt kennt und nicht umgekehrt, müssen die Änderungen aktiv vom Morph-Element ausgehen. Das Informationsobjekt besitzt eine passive Rolle. Deshalb wird das Verfahren auch passive Notification oder Polling genannt [CoMa95 S.176].
Der Morph fragt dabei regelmäßig bei seinem Informationsobjekt nach den aktuellen Attributwerten. Das Informationsobjekt gibt über sein Standardprotokoll die Auskunft. Der Morph aktualisiert mit den neuen Werten seine Darstellung. Nachteilig beim Polling-Verfahren ist, im Vergleich zu der Anwendung des Observer-Pattern, wie z.B. beim Model-View-Controller-Konzept [GHJV95 S.325], das große Nachrichtenaufkommen und die unnötige Aktualisierung der Darstellung. Es wurde deshalb im Rahmen der Arbeit ein modifizierter Polling-Mechanismus konzipiert, mit dem Ziel, die Nachteile zu verringern.

Das Nachrichtenaufkommen wird als erstes dadurch reduziert, daß die Morphs nur in bestimmten Zeitintervallen, in einer Größenordnung von wenigen Sekunden nach Änderungen fragen (timer based notification [CoMa95 S.178])[6]. Diese Möglichkeit, den Nachrichtenfluß zu verringern, ist sehr einfach und effizient, allerdings allein nicht ausreichend.
Weiterhin werden nur die in einem Morph angezeigten Eigenschaften bei den Informationsobjekten nachgefragt. Dies geht so weit, daß er sich ruhig verhält, wenn ein Morph keine Eigenschaft visualisert.
Der dritte entscheidende Bestandteil des Pollings ist, daß nur die veränderten Werte aktualisiert werden. Zu diesem Zweck verwalten die Informationsobjekte zusätzlich die Information über Veränderungen ihrer Eigenschaften. Der Morph kann im ersten Schritt nachfragen, ob sich sein Informationsobjekt überhaupt geändert hat. In der Regel ist das nicht der Fall, und der Polling-Zyklus ist beendet. Falls sich etwas geändert hat, fragt der Morph nach den geänderten Eigenschaften. Haben sich Eigenschaftswerte geändert, die im Morph enthalten sind, wird ihre Anzeige aktualisiert. Wurde eine Eigenschaft beim Informationsobjekt entfernt, wird der entsprechende Eigenschaftsmorph entfernt.
Da ein Informationsobjekt in mehreren Sichten in mehreren Morphs enthalten sein kann, reicht es nicht aus, daß nur auf der Seite der Informationsobjekte verwaltet wird, ob sich eine Eigenschaft geändert hat. Für die Verwaltung der Änderungen wird zusätzlich auf der Seite des Informationsmorph für die Eigenschaftswerte der Zeitpunkt der letzten Änderung verwaltet. Über ihr erweitertes Selbstbeschreibungsprotokoll können die Informationsobjekte (ihren Morphs) darüber Auskunft geben:

`(anInformationObject wasModifiedSince: aTimeStamp)`
=> antwortet mit boolschem Wert, durch den beschrieben wird, ob sich der Empfänger seit dem als Parameter übergebenen Zeitpunkt geändert hat

`(anInformationObject modAttributesSince: aTimeStamp)`
=> antwortet mit allen Attributeigenschaften, die sich seit dem als Parameter übergebenen Zeitpunkt geändert haben

`(anInformationObject registerModFor: #attributeName)`
=> verwaltet den Zeitpunkt der Änderung von der Eigenschaft, die durch den Parameter spezifiziert wird

[6]Damit es nicht zu unnötigen Verzögerungen bei den Änderungen des Informationsmodells kommt, die über das Interaktionsmodell ausgelöst wurden, wird bei einer manuellen Tätigkeit des Anwenders ein Polling-Zyklus bei allen beteiligten Morphs ohne Zeitverzögerung angestoßen.

Abschließend sei noch darauf hingewiesen, daß wegen der einseitigen Beziehung das Informationsmodell seine Darstellung nicht direkt beeinflussen kann. Dies ist problematisch, weil dadurch die Informationsobjekte Fehler und andere Meldungen nicht direkt über die GUI ausgeben lassen können.
Zur Lösung des Problems verwaltet ein Informationsobjekt seine Mitteilungen an den Anwender. Ein Informationsmorph fragt zusätzlich im Rahmen eines Polling-Zykluses bei seinem Informationsobjekt nach, ob seit der letzten Nachfrage eine neue Mitteilung erzeugt wurde. Falls dies der Fall ist, fragt der Informationsmorph die neue Mitteilung ab und erzeugt einen passenden Morph, der über die GUI den Anwender informiert.

7. MSS-Modell

Aufgabe des MSS-Modells ist es, für typische MSS-Fragestellungen relevante Ausschnitte des Informationsmodells inklusive Strukturen, Methoden und Wertungen abzubilden. Die dem ooMSS zugrundeliegenden Informationsobjekte werden dazu im MSS-Modell um Eigenschaften ergänzt, auf denen die MSS-Funktionen Restriktion, Führung und Anpassung basieren.

7.1 Grundkonzepte

Die Informationsobjekte verwalten ihre im folgenden beschriebenen Meta-Informationen auf der Meta-Ebene durch Instanzen der Klasse `InformationAspect` (Aspekt: lat. Ansicht, Gesichtspunkt, Anblick) und dokumentieren sie, unabhängig von der Implementierung, im Rahmen einer erweiterten Selbstbeschreibung in Form von Protokollen.
Als Eigenschaft besitzen die Aspektobjekte den inhaltlichen Zusammenhang (`modelContext`) zum Informationsmodell. Für eine Klasse beschreibt der Aspekt eine Eigenschaft aller ihrer Instanzen, alternativ nur eine Eigenschaft eines einzelnen Objekts. Das dazu passende Protokoll sieht folgendermaßen aus:

`(anInformationAspect modelContext)`
=> Der Informationsaspekt antwortet mit dem Modellausschnitt, auf den er sich bezieht.

`(anInformationAspect modelContext: anInformationObject)`
=> Der Modellausschnitt eines Informationsaspekts wird auf ein einzelnes Informationsobjekt gesetzt.

`(anInformationAspect modelContext: InformationObject)`
=> Der Modellausschnitt eines Informationsaspekts wird auf die Menge der zu einer Informationsobjektklasse gehörenden Instanzen gesetzt.

Unterschieden werden im MSS-Modell zwei grundlegende Aspekttypen, durch die die MSS-spezifischen Kriterien unterstützt werden. Mit den Aspekten des Restriktionsbereich werden die technisch möglichen Verwendungen der Informationsobjekte abgelegt. Mit den Aspekten des Führungsbereichs werden die fachlich bedeutenden Verwendungen der Informationsobjekte beschrieben (siehe Abbildung 7.1).

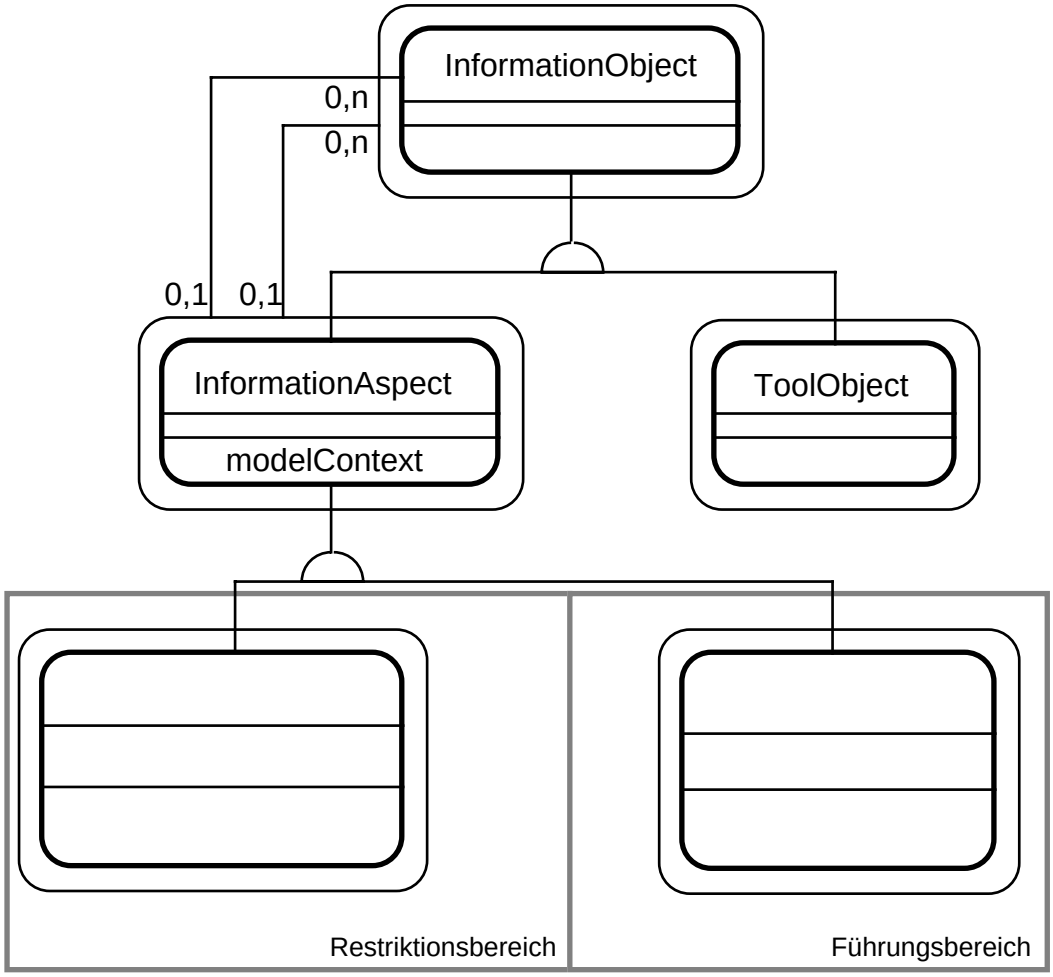

Abbildung 7.1: Objektmodell der Informationsaspekte (Übersicht)
(Objektnotation nach [CoYo94a])

7.2 Restriktionsaspekte

7.2.1 Technische Restriktionsaspekte

Die technischen Restriktionsaspekte entstehen bei der Einrichtung der Strukturen des Informationsmodells. Durch sie wird die Anzahl der möglichen Zustände des Informationsmodells beschränkt, indem durch logische Beschränkungen (engl. Constraints) festgelegt wird, was bei Zugriffen auf die Eigenschaften bei der Anwendung des ooMSS zu berücksichtigen ist. Der Einsatz von Beschränkungen sichert die Konsistenz und damit die Qualität [RiMK99]. Diese Mechanismen bilden damit die technische Grundlage für die Erfüllung der Restriktion des MSS-Kriterienbereichs (siehe Kapitel 2.3.1).
Die technische Grundlage dafür ist der im Informationsmodell enthaltene kontrollierte Zugriff auf die Eigenschaften im Informationsmodell. Der Zugriff wird durch das MSS-Modell um die Berücksichtigung der Beschränkungen erweitert.

Jede Beschränkung ist dazu mit einer Attribut- oder Methodeneigenschaft eines Informationsobjekts verbunden. Auf der Meta-Ebene werden für jede Eigenschaft Beschränkungen durch Instanzen der Klassen `AttributeAspect` oder `MethodAspect` abgelegt. Beim Attribut- und Methodenzugriff werden so die Beschränkungen durch die modifizierten Zugriffsmethoden berücksichtigt (siehe Abbildung 7.2).

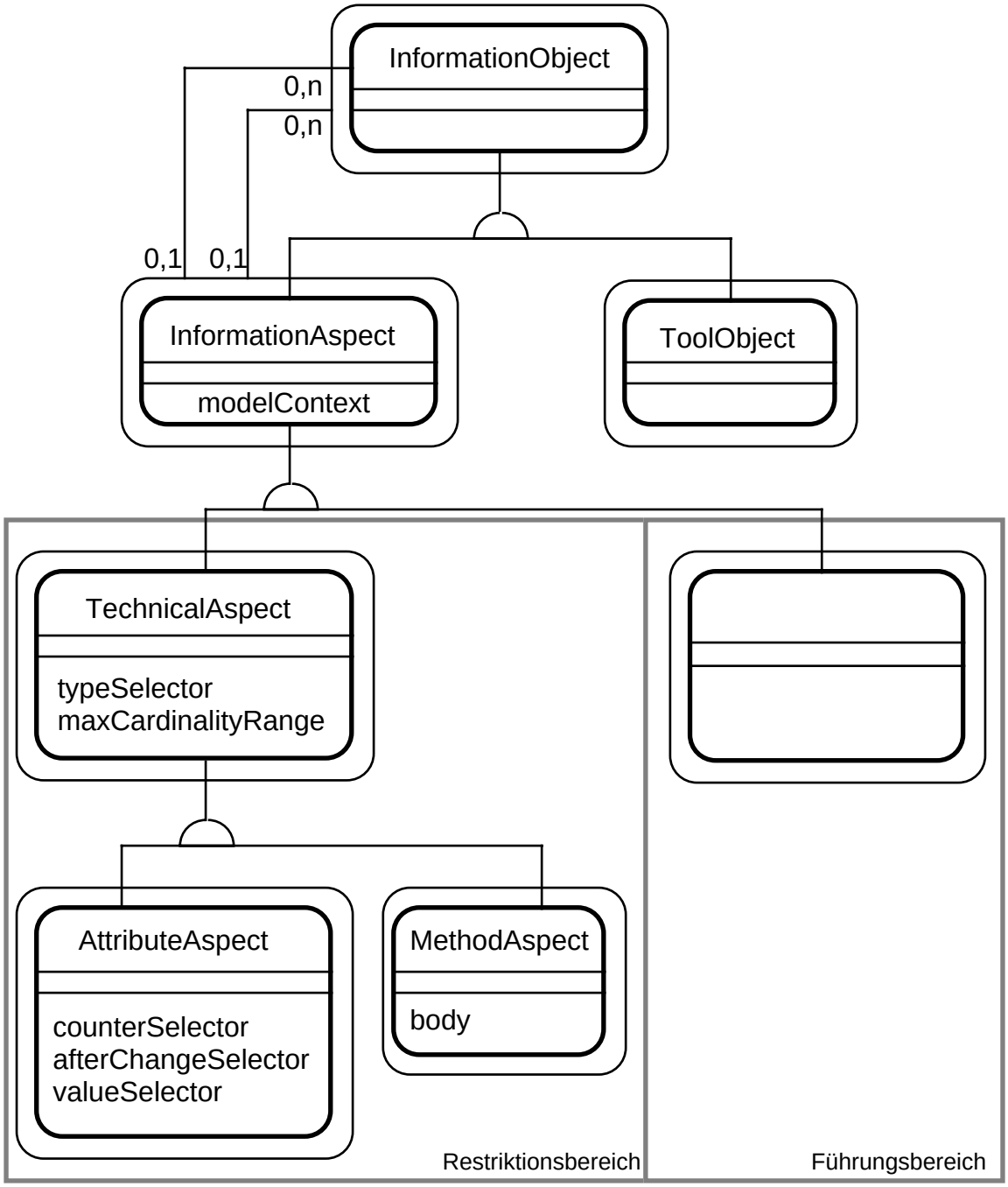

Abbildung 7.2: Objektmodell der Informationsaspekte mit TechnicalAspect (Objektnotation nach [CoYo94a])

Die technischen Restriktionsaspekte besitzen einheitlich als Eigenschaft einen Bezeichner, einen Objekttypen und eine Kardinalität.
Die Bezeichner-Eigenschaft (`selector`) unterstützt die grundlegende Verwaltung der Attribut- und Methodenaspekte. Bei der Erzeugung eines Aspekts wird dazu der Bezeichner als Parameter übergeben. Mit der Erzeugung eines Aspekts ist gleichzeitig, unter Verwendung der grundlegenden Protokolle des Informationsmodells, die Erzeugung der dazugehörigen Eigenschaft verbunden[1]. Auf einen existierenden Aspekt einer Informationsobjektklasse kann durch den Bezeichner auf einfache Weise zugegriffen werden. Ein Bezeichner ist im Rahmen einer Informationsobjektklasse eindeutig. Zwei technische Restriktionsaspekte können keinen gleichen Bezeichner besitzen.

```
(aTechnicalAspect := TechnicalAspect selector: #aspectSelector.
    InformationObject addTechnicalAspect: aTechnicalAspect.
    InformationObject atTechnicalAspect: #aspectSelector)
```
=> Ein technischer Restriktionsaspekt wird erzeugt, zu einer Informationsobjektklasse hinzugefügt (inkl. der dazu passenden Eigenschaft) und über seinen Bezeichner abgerufen.

```
(InformationObject removeTechnicalAspectAt: #aspectSelector)
```
=> Ein technischer Restriktionsaspekt und die damit verbundene Eigenschaft wird anhand seines Aspektbezeichners aus einer Informationsobjektklasse entfernt.

Die folgende Tabelle 7.1 gibt eine Übersicht auf die im ooMSS enthaltenen Constraint-Typen und ihre Berücksichtigung in den technischen Attribut- und Methodenaspekten:

	Attributaspekt	Methodenaspekt
Typisierung	√	√
Kardinalität	√	√
Wertebereich	√	-
Inverse Beziehung	√	-
Trigger	√	-

Tabelle 7.1: Constraint-Typen des ooMSS

[1]Zusätzlich werden die im Abschnitt 6.4.3 beschriebene Objekttypen Attribut- und Methodeneigenschaft hier umgesetzt.

Typisierung

Im Rahmen eines technischen Restriktionsaspekts wird ein Objekttyp (`typeSelector`) in Form einer Klassenbezeichnung beschrieben. Bei einem Attribut dient der Objekttyp zur Typisierung der als Attributwert verwalteten Objekte. Durch die Typisierung wird sichergestellt, daß die Objekte das Protokoll der benannten Klasse besitzen und damit bestimmte Eigenschaften haben. Die Vorteile einer starken Typbindung lassen sich so zusammen mit der Flexibilität einer schwachen Typbindung in dem ooMSS nutzen.
Bei der Änderung des Attributwertes über die Zugriffsmethoden wird überprüft, ob die als Parameter übergebenen Objekte zu der im Attributaspekt bezeichneten Klasse oder einer Unterklasse gehören. Ist dies der Fall, wird die Änderung des Attributwertes durchgeführt, falls nicht, wird die Änderung abgewiesen.
Diese Eigenschaften und grundlegenden Abläufe spiegeln sich in folgenden Protokollen wider:

```
(anAttributeAspect :=
    AttributeAspect selector: #attributeSelector
    typeSelector: #ClassNameSelector.
InformationObject addTechnicalAspect: anAttributeAspect)
```
=> Ein technischer Restriktionsaspekt wird erzeugt und zu einer Informationsobjektklasse hinzugefügt (inkl. des dazu passenden Attributs).

```
(anInformationObject
    attributeSelector: aSecondInformationObject.
anInformationObject attributSelector)
```
=> Auf das durch den Selektor bezeichnete Attribut wird unter Beachtung des Typs zugegriffen.

```
(anAttributeAspect := AttributeAspect
    selector: #kunde typeSelector: #Kunde.
Rechnung addTechnicalAspect: anAttributeAspect)
```
=> Der technische Kundenaspekt wird erzeugt und zu der Klasse `Rechnung` hinzugefügt.

```
(eineRechnung kunde: einKunde.
eineRechnung kunde)
```
=> Hinter der Kundeneigenschaft der Rechnung wird der Kunde verwaltet.

```
(eineRechnung kunde: einAuftrag.
eineRechnung kunde)
```
=> Die Verwaltung eines Auftragsobjekts als Kundeneigenschaft der Rechnung wird abgewiesen. Hinter der Kundeneigenschaft befindet sich unverändert ein Kundenobjekt.

Für eine Methode dient der Objekttyp auf vergleichbare Weise zur Typisierung der als Parameter übergebenen Informationsobjekte. Beim Aufruf einer Methode wird vor der eigentlichen Verarbeitung zunächst die Einhaltung dieser Beschränkung überprüft. Wenn das Informationsobjekt vom passenden Typ ist, dann wird die Methode ausgeführt, anderenfalls wird die Verarbeitung nicht durchgeführt. Die allgemeinen Protokolle sind analog zu denen der Attributaspekte. Ein konkreter Anwendungsfall läßt sich wie folgt beschreiben:

```
(aMethodAspect := MethodAspect
    selector: #berechne
    typeSelector: #Berechnungsvorschrift.
Auswertung addTechnicalAspect: aMethodAspect)
```
=> Der technische Berechnungsaspekt wird erzeugt und zu der Klasse `Auswertung` hinzugefügt.

```
(eineAuswertung berechne: eineBerechnungsvorschrift)
```
=> Die Methode `berechne:` wird erfolgreich aktiviert, wobei die zu verwendende Berechnungsvorschrift mit übergeben wird.

```
(eineAuswertung berechne: einKunde)
```
=> Die Aktivierung der Methode berechne: wird abgewiesen, da als nicht geeignete Berechnungsvorschrift ein Kundenobjekt übergeben wird. Die Auswertung bleibt unverändert in ihrem Zustand.

Kardinalität

Mit der Kardinalitätsbezeichnung (`maxCardinalityRange`) wird bei einem Attribut beschrieben, ob ein einzelnes Objekt oder eine Menge von Objekten durch eine Instanz der Klasse `InformationCollection` hinter einem Attributwert verwaltet wird. Fehlt bei der Erzeugung eines technischen Restriktionsaspekts die Kardinalitätsbezeichnung, wird standardmäßig ein einzelnes Objekt verwaltet. Für ein mehrfaches Attribut, also für eine Kompositionsbeziehung, wird automatisch bei der Erzeugung des Objekts der Attributwert mit einer Liste von Informationsobjekten initialisiert.

Die Protokolle zur Erzeugung und Verwaltung der Kardinalitätsbeziehung ergänzen die bisher beschriebenen auf folgende Art und Weise:

```
(anAttributeAspect := AttributeAspect
    selector: #attributeSelector
    typeSelector: #ClassNameSelector
    maxCardinalityRange: #single.
InformationObject
    addTechnicalAspect: anAttributeAspect.)
```

=> Ein einfacher Attributaspekt wird erzeugt und zu einer Informationsobjektklasse hinzugefügt (inkl. des dazu passenden singulären Attributs).

```
(anAttributeAspect := AttributeAspect
    selector: #attributeSelector
    typeSelector: #ClassNameSelector
    maxCardinalityRange: #multiple.
InformationObject
    addTechnicalAspect: anAttributeAspect.)
```

=> Ein einfacher Attributaspekt wird erzeugt und zu einer Informationsobjektklasse hinzugefügt (inkl. des dazu passenden multiplen Attributs).

```
(anAttributeAspect := AttributeAspect
    selector: #auftraege
    typeSelector: #Auftrag
    maxCardinalityRange: #multiple.
Kunde
    addTechnicalAspect: anAttributeAspect.)
```

=> Der technische multiple Auftragsaspekt wird erzeugt und zu der Klasse Kunde hinzugefügt.

```
(einKunde addAuftraege: einAuftrag)
```

=> Hinter der Auftragseigenschaft des Kunden wird ein zusätzlicher Auftrag verwaltet.

```
(einKunde addAuftraege: eineRechnung)
```

=> Die Verwaltung eines Rechnungsobjekts hinter der Auftragseigenschaft des Kunden wird abgewiesen. Das Kundenobjekt bleibt in seinem Zustand unverändert.

Für den Aspekt einer Methode dient die Kardinalitätsbezeichnung zur Festlegung der Kardinalität der als Parameter übergebenen Informationsobjekte. Ohne Angabe der Kardinalität wird hier ebenfalls ein einzelner Parameter erwartet. Wie bei der Typisierung wird beim Aufruf einer Methode vor der Verarbeitung die passende Kardinalität überprüft.
Beispiele für konkret angewendete Protokolle sind:

```
(aMethodAspect := MethodAspect
    selector: #berechne:
    typeSelector: #Berechnungsvorschrift
    maxCardinalityRange: #single.)
```
=> Der erzeugte technische Berechnungsaspekt verwendet einen einzelnen Parameter. Die weitere Anwendung entspricht den zuletzt beschriebenen Protokollen der Methodenaspekte.

```
(aMethodAspect := MethodAspect
    selector: #visualisiere:
    typeSelector: #DomaenenObjekt
    maxCardinalityRange: #multiple.
Darstellung addTechnicalAspect: aMethodAspect.)
```
=> Der erzeugte Methodenaspekt verwendet eine Menge von Domänenobjekten und wird zu der Klasse `Darstellung` hinzugefügt.

```
(eineDarstellung visualisiere: eineDomaenenobjektMenge)
```
=> Die Methode `visualisiere:` wird erfolgreich aktiviert, wobei die darzustellenden Domänenobjekte mit übergeben werden.

```
(eineDarstellung visualisiere: einKunde)
```
=> Die Aktivierung der Methode `visualisiere:` wird abgewiesen, da als nicht geeigneter Darstellungsinhalt ein einzelnes Domänenobjekt übergeben wird. Die Darstellung bleibt unverändert in ihrem Zustand.

Über die einheitlichen Eigenschaften hinaus besitzen Attributaspekte zusätzlich individuelle Selektoren zur Bestimmung eines Wertebereichs, einer inversen Beziehung und eines Triggers.

Wertebereich

Durch den Wertebereich-Selektor (`valueSelector`) wird ergänzend zu der Typisierung die Menge der hinter einem Attributwert stehenden

Objekte weiter eingeschränkt. Durch den Selektor wird eine Methode eines Objekts beschrieben, die beim Aufruf mit einer Menge von diskreten Attributwerten antwortet. Bei Attributwertänderungen wird der neue Attributwert nur übernommen, wenn er in der Menge enthalten ist. Die dazugehörigen Protokolle sind:

```
(anAttributeAspect := AttributeAspect
    selector: #attributeSelector
    typeSelector: #ClassNameSelector
    valueSelector: #valueSelector.
InformationObject addTechnicalAspect: anAttributeAspect)
```
=> Ein technischer Restriktionsaspekt wird erzeugt, wobei ein Wertebereich-Selektor definiert wird.

```
(InformationObject valuesAtAspect: #attributeSelector)
```
=> Die Informationsklasse antwortet für das durch den Selektor bezeichnete Attribut mit einer Menge von möglichen Attributwerten. Die Klasse aktiviert dazu ihre Methode, die durch den `#attributeSelector` bestimmt wird.

```
(anAttributeAspect := AttributeAspect
    selector: #bewertung
    typeSelector: #Symbol
    valueSelector: #bewertungsMoeglichkeiten.
Notiz addTechnicalAspect: anAttributeAspect.)
```
=> Ein Bewertungsaspekt wird erzeugt und zu der Klasse `Notiz` hinzugefügt. Durch die Bewertung soll der Nutzen einer Notiz dokumentiert werden. Mögliche Ausprägungen, mit denen die Methode `#bewertungsMoeglichkeiten` antwortet, sind `#gut`, `#mittel`, `#schlecht`. Auf dieser Festlegung basieren folgende Protokollabläufe:

```
(eineNotiz bewertung: #gut)
```
=> Eine Notiz wird positiv bewertet.

```
(eineNotiz bewertung: #mittelpraechtig)
```
=> Eine Bewertung der Notiz findet nicht statt, weil sich der Wert nicht in der Menge der möglichen Ausprägungswerte befindet. Die Notiz bleibt unverändert.

Auch das Interaktionsmodell kann auf die Wertebereichseigenschaft des MSS-Modells zurückgreifen. Dazu können dem Anwender bei

einer manuellen Änderung des Attributwertes an einem Informationsmorph die konkreten Werte in einer Auswahlliste präsentiert werden (siehe Abbildung 7.3). Der Anwender wird so zusätzlich geführt.

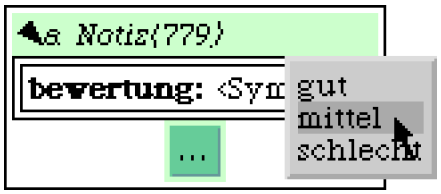

Abbildung 7.3: Auswahlliste

Inverse Beziehung

Mit dem Selektor der inversen Beziehung (`counterSelector`) wird eine zwingend beidseitige Assoziation zwischen zwei Informationsobjekten sichergestellt. Angelegt werden inverse Selektoren im Rahmen der folgenden Protokolle:

```
(anAttributeAspect1 := AttributeAspect
    selector: #attributeSelector2
    typeSelector: #InformationObject2
    counterSelector: #attributSelector1.
InformationObject1
    addTechnicalAspect: anAttributeAspect1)
```

=> Es wird ein Attributaspekt erzeugt und zu der ersten Informationsobjektklasse (InformationObject1) hinzugefügt. Hinter dem Aspekt wird von den Instanzen der ersten Informationsobjektklasse eine Instanz der durch den Selektor `#ClassNameSelector2` spezifizierten zweiten Informationsobjektklasse (`InformationObject2`) verwaltet. Die inverse Beziehung besteht darin, daß wenn eine Instanz der ersten Klasse hinter ihrem Attribut `#attributSelector2` eine Instanz der zweiten Klasse verwaltet, dann auch von der Instanz der zweiten Klasse hinter ihrem Attribut `#attributSelector1` gleichzeitig dieselbe Instanz der ersten Klasse verwaltet wird und umgekehrt. Die folgende Abbildung 7.4 veranschaulicht diese beidseitige Beziehung:

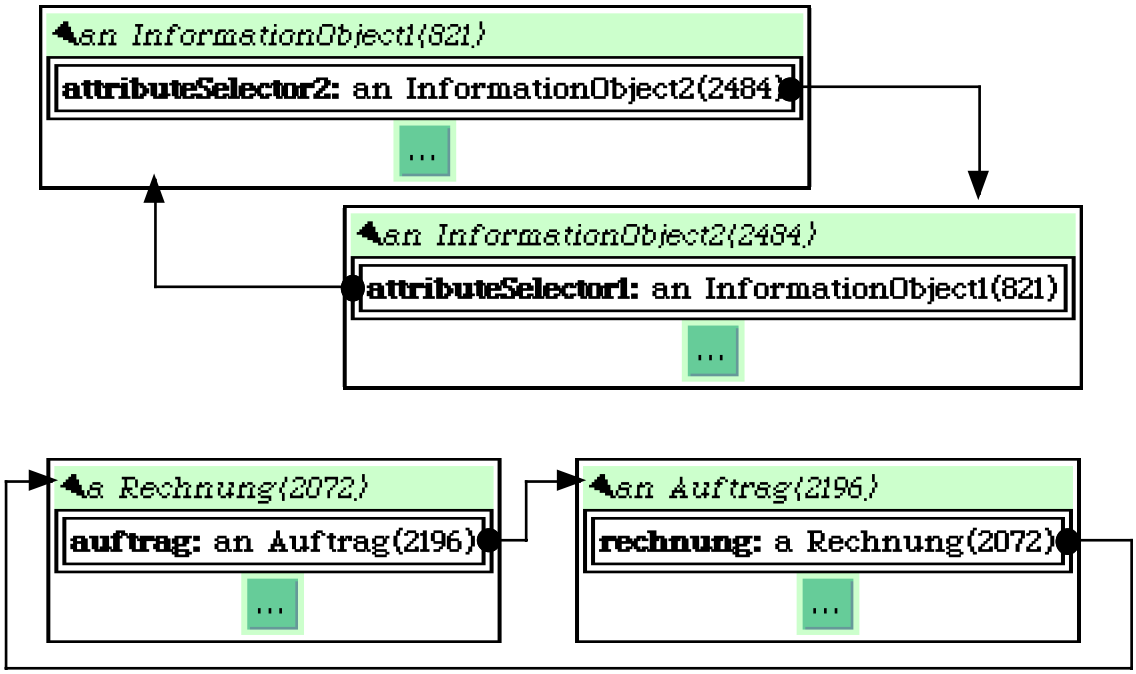

Abbildung 7.4: Inverse Beziehung (allgemein und speziell)

Das Protokoll für den in der Abbildung zusätzlich enthaltenen konkreten Anwendungsfall sieht wie folgt aus:

```
(einAuftragsAspekt := AttributeAspect
    selector: #auftrag
    typeSelector: #Auftrag
    counterSelector: #rechnung.
Rechnung addTechnicalAspect: einAuftragsAspekt.
einRechnungsAspekt := AttributeAspect
    selector: #rechnung
    typeSelector: #Rechnung
    counterSelector: #auftrag.
Auftrag addTechnicalAspect: einRechnungsAspekt.)
```

=> Es wird zunächst ein Auftragsaspekt erzeugt und zu der Klasse `Rechnung` hinzugefügt. Dann wird analog ein Rechnungsaspekt erzeugt und zu der Klasse `Auftrag` hinzugefügt.

```
(eineRechnung := Rechnung new.
einAuftrag := Auftrag new.
eineRechnung auftrag: einAuftrag)
```

=> Das Rechnungsobjekt verwaltet hinter seiner Eigenschaft `#auftrag` das Auftragsobjekt. Gleichzeitig verwaltet dasselbe Auftragsobjekt hinter seinem Attribut `#rechnung` dasselbe Rechnungsobjekt.

```
(einAuftrag rechnung: nil.
eineRechnung auftrag)
```

=> Mit der einseitigen Auflösung wird die Verbindung von der Gegenseite - in diesem Fall das Attribut `#auftrag` der Rechnung - ebenfalls aufgelöst. Die Rechnung antwortet auf die Nachricht `#auftrag` mit der Konstanten `"nil"`, durch die nichts repräsentiert wird.

In der Implementierung sind weitere zusätzliche Elemente berücksichtigt worden, auf die im Detail hier nicht eingegangen werden soll:

- Die zugrundeliegenden Abläufe sind zusätzlich von der Kardinalität der beidseitigen Beziehung abhängig. Mit einer Fallunterscheidung werden 1:1, 1:n und n:n-Assoziationen berücksichtigt.
- Bei der Verwaltung der zwingend beidseitigen Assoziation werden neben den direkt betroffenen Informationsobjekten zusätzlich die indirekt betroffenen Informationsobjekte behandelt. Wenn eine Verbindung zwischen zwei Informationsobjekten neu hergestellt wird, müssen gegebenenfalls auch die zuvor mit den beiden Informationsobjekten verbundenen Informationsobjekte in ihrem Zustand aktualisiert werden.

Trigger

Der letzte technische Restriktionsaspekt ist der Trigger. Durch Definition eines Triggers wird ein Attribut aktiv, d.h., es wird an dem Objekt eine Handlung ausgeführt, wenn sich der Attributwert ändert. Durch ein aktives Attribut kann die Konsistenz und Korrektheit unterstützt werden. Bei der Erstellung einer Assoziation kann so das Informationsobjekt, das als neuer Attributwert verwaltet werden soll, auf seine spezifischen Eigenschaften hin überprüft werden. Mit dem Trigger-Mechanismus lässt sich bspw. im MSS-Bereich ein Exception-Reporting in das Informationsmodell integrieren [Rieg93 S.95ff.]. Bei dieser auf dem Management-by-Exception basierenden Anwendungsform wird die Aufmerksamkeit des Anwenders durch das System aktiv auf signifikante Abweichungen von Planwerten gelenkt.

Die folgenden Protokolle zeigen die Trigger-Anwendung allgemein und an einem Beispiel:

```
(anAttributeAspect := AttributeAspect
    selector: #attributeSelector
    typeSelector: #ClassNameSelector
    afterChangeSelector: #methodSelector:.
InformationObject addTechnicalAspect: anAttributeAspect)
```

=> Ein technischer Restriktionsaspekt wird erzeugt, wobei ein Selektor für einen Trigger definiert wird. Nach der Änderung des Attributs wird die durch den Selektor beschriebene Methode aktiviert.

```
(einInhaltsaspekt := AttributeAspect
    selector: #inhalt
    typeSelector: #InformationObject
    afterChangeSelector: #ueberpruefeInhalte:.
Auswertung addTechnicalAspect: einInhaltsaspekt)
```

=> Bei der Änderung des Auswertungsinhalts werden durch den Aufruf der Methode `#ueberpruefeInhalte:` die neuen Inhalte auf ihre Auswertbarkeit hin überprüft. Die neuen Inhalt sollten z.B. für eine Aggregationsauswertung mindestens eine numerische Eigenschaft besitzen. Haben sie diese nicht, sind sie als Auswertungsinhalt untauglich und werden zurückgewiesen.

```
(einIstWertaspekt := AttributeAspect
    selector: #istWert
    typeSelector: #InformationObject
    afterChangeSelector: #ueberpruefeAbweichung:.
einKennzahl addTechnicalAspect: einIstWertaspekt)
```

=> Bei der Änderung der Kennzahl wird durch den Aufruf der Methode `#ueberpruefeAbweichung:` der neue Ist-Wert überprüft, inwieweit er von dem Planwert abweicht. Ist die Abweichung signifikant, wird sie im Rahmen der Methode `#ueberpruefe-Abweichung:` an den Anwender gemeldet.

Die Verbindung zwischen den technischen Restriktionsaspekten und dem Interaktionsmodell veranschaulicht die folgende Abbildung 7.5:

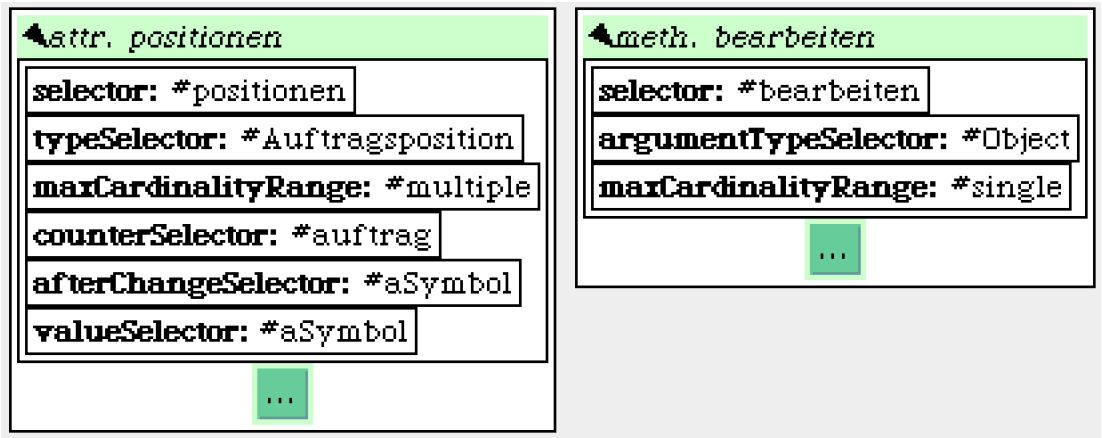

Abbildung 7.5: Attribut- und Methodenaspektmorph

Die Abbildung 7.5 zeigt jeweils einen Informationsmorph für einen Attribut- und einen Methodenaspekt. Auch die Werte dieser Restriktionsaspekte werden analog zu den anderen Elementen des Informationsmodells über die morph-basierte Schnittstelle des ooMSS vom Entwickler spezifiziert.

7.2.2 Fachliche Restriktionsaspekte

Auf der Basis der technischen Beschränkungen wird durch die fachlichen Aspekte zusätzlich die für eine Anwendung des ooMSS notwendige Beziehung zwischen den Werkzeugobjekten und den Domänenobjekten des Informationsmodells dokumentiert. Durch die fachlichen Aspekte, die als Instanzen der Klasse `ToolAspect` abgebildet werden, wird so die Erfüllung des Restriktionskriteriums vervollständigt (siehe Abbildung 7.6).

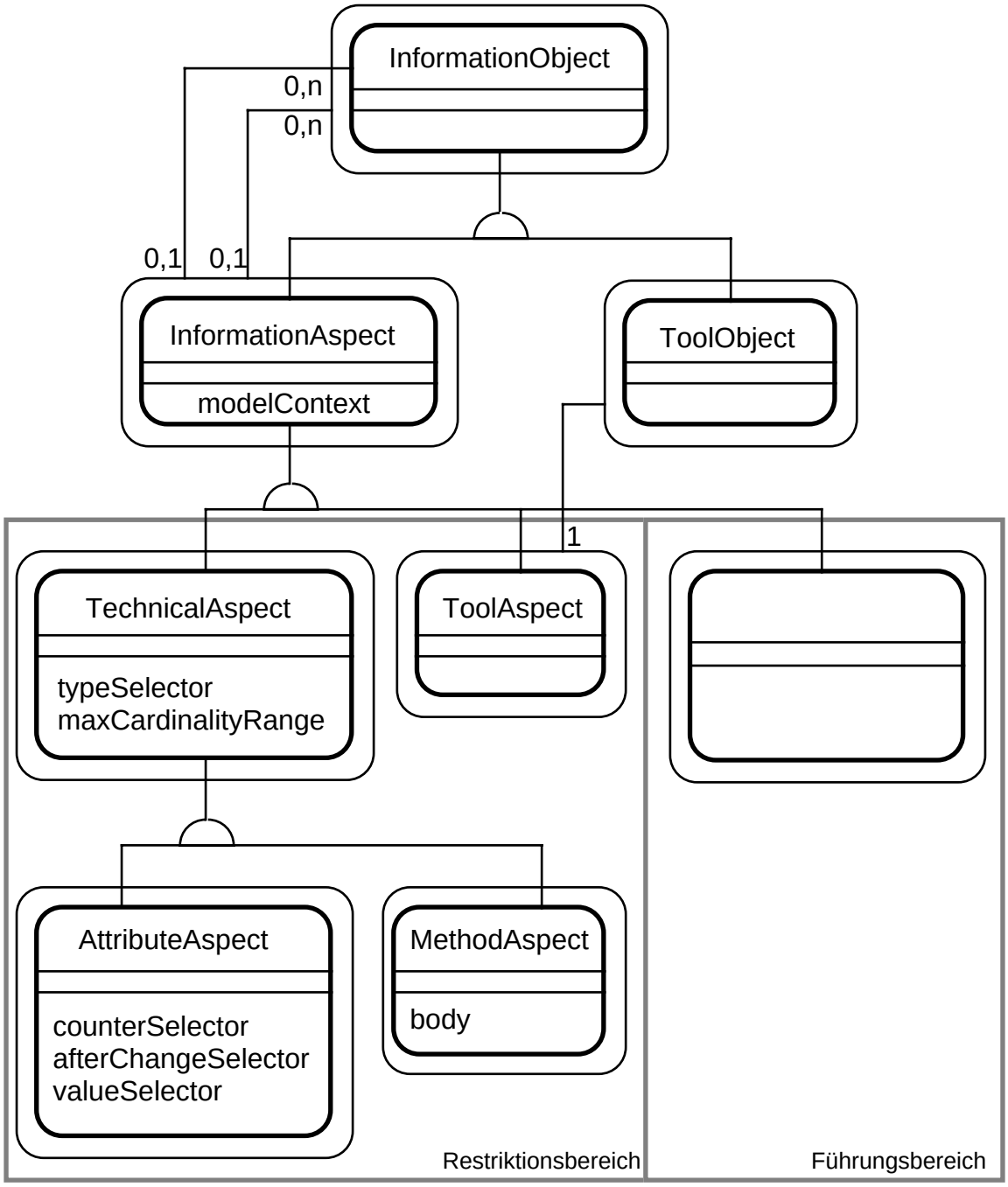

Abbildung 7.6: Objektmodell der Informationsaspekte mit ToolAspect (Objektnotation nach [CoYo94a])

Die fachlichen Aspekte entstehen im Unterschied zu den technischen bei der Anwendung des ooMSS. Jede entstehende Beziehung besitzt eine fachliche Bedeutung und bildet einen elementaren Auswertungsbaustein, der jeweils auf einem komplexen Werkzeugobjekt basiert. Die fachlichen Restriktionsaspekte haben dabei die Aufgabe eines Adapters [GHJV95 S.139], der die Protokolle der zugrundeliegenden Werkzeugobjekte für den Anwender vermittelt.

Ausgangspunkt jeder Einrichtung eines Auswertungsbausteins ist jeweils ein neu erzeugter fachlicher Restriktionsaspekt. Der erste Einrichtungsschritt besteht für den Anwender darin, den gewünschten Modellbezug des Aspekts über die Eigenschaft `modelContext` zu spezifizieren. Bei der Aktivierung der Eigenschaft des Informationsmorph werden dem Anwender dazu die Namen von allen im ooMSS zur Verfügung stehenden Informationsklassen angeboten (siehe Abbildung 7.7:).

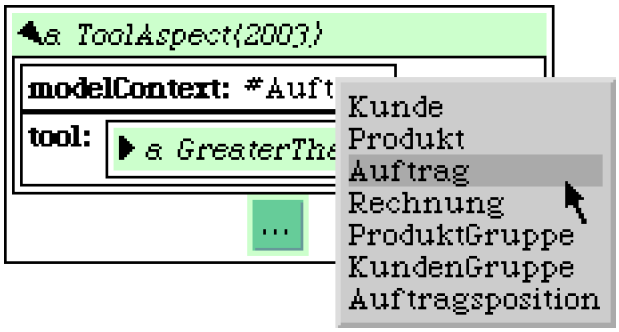

Abbildung 7.7: Modellkontext auswählen

Der Anwender besitzt zwei grundlegende Möglichkeiten, die den fachlichen Restriktionsaspekten zugrundeliegenden Werkzeugobjekte über das Interaktionsmodell einzurichten. Zum einem kann er an den Aspekten kontrolliert Parameter festlegen, durch die Attributwerte der zugrundeliegenden Werkzeugobjekte direkt geändert oder in Werkzeugkombinationen umgesetzt werden. Bei einem Sortieraspekt ist z.B. die Reihenfolge eine Eigenschaft, die direkt als Attributwert in einem `SortTool` verwendet wird. Eine vom Anwender eingegebene Pfadangabe wird dagegen erst durch die Umsetzung in einem `PathTool` Bestandteil eines `SortTool`.
Zum anderen kann der Anwender darüber hinaus beliebig komplexe Zusammenhänge durch die Kombinationen von Werkzeugen einrichten. Ein Beispiel ist die Erstellung von Abfragen auf der Basis von Instanzen einer Unterklasse von `BooleanTool`.
Die Abfragen bestehen, wie beim Werkzeugmodell beschrieben, aus einer Kombination von Instanzen der Klassen `BooleanTool`, `ValueTool` und `PathTool`. Sie werden zunächst über das Interaktionsmodell vom Anwender visuell zusammengesetzt und dann mit der Eigenschaft eines fachlichen Aspekts verbunden (siehe Abbildung 7.8).

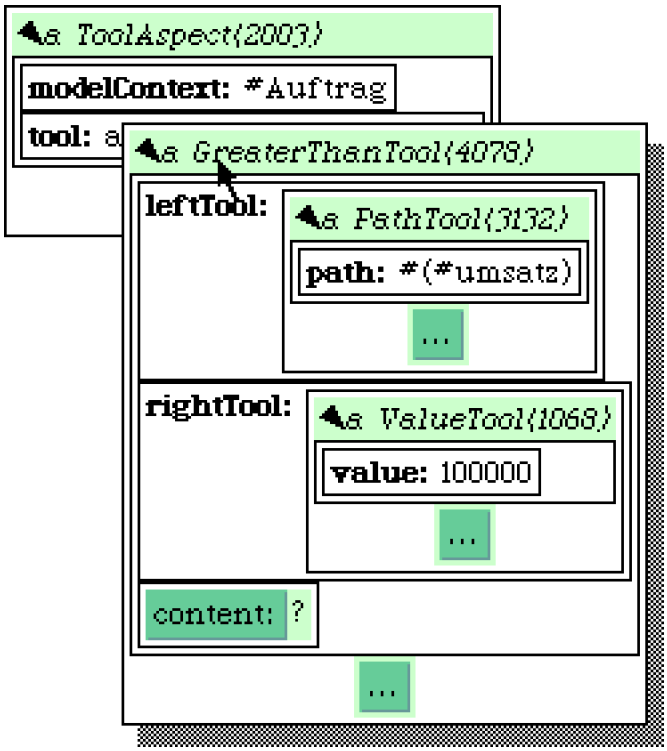

Abbildung 7.8: Erstellung einer Abfrage

Damit in einer Anwendung Bestandteile einer Auswertung verwaltet werden können, werden die fachlichen Aspekte auf der Klassenebene verwaltet und besitzen über die Eigenschaft `modelContext` den Bezug zu ihrer Domänenklasse, deren Instanzen durch ihre zugrundeliegenden Werkzeugobjekte verarbeitet werden.

Im folgenden wird auf die fachlichen Aspektarten eingegangen. Es wird beschrieben, wie sie über das Interaktionsmodell an den Anwender vermittelt und auf welche spezifische Weise er bei ihrer Anpassung unterstützt wird.

SourceAspekt
: SourceAspekte beschreiben, jeweils unter Verwendung eines `SourceTool`, welche Informationsobjekte in einer Unterstützungssituation berücksichtigt werden. Bei einem SourceAspekt wird automatisch als Objektquelle die in der Eigenschaft `modelContext` verwaltete Klasse verwendet. Ergänzend dazu kann der Anwender die Menge der Informationsobjekte beschränken, indem er die SourceAspekt-Eigenschaft `query` mit einer von ihm erstellten passenden Abfrage verbindet.

Aggregationsaspekt
: Die Aggregationsaspekte stellen dar, wie Informationsobjekte aggregiert werden. Es werden so applikationsübergreifende und klassenbezogene Kennzahlen beschrieben. Dazu benutzt der Aspekt ein `Aggregation-Tool`, durch das spezifiziert wird, über welche Eigenschaft zur Aggregation verwendet und nach welcher Art aggregiert wird. Der Pfad der zu aggregierenden Eigenschaft wird dabei direkt eingelesen und die alternativen Aggregationsarten ausgewählt.

Sortieraspekt
: Sortieraspekte beschreiben, wie Informationsobjekte in eine bestimmte Reihenfolge gebracht werden. Anhand einer Eigenschaft werden durch das zugrundel iegende `SourceTool` Rankings erstellt. Die Interaktion ist vergleichbar mit den Möglichkeiten der Aggregationsaspekte: Der Pfad der Eigenschaft, die als Sortierkriterium zu verwenden ist und die alternativen Sortierungsarten werden direkt ausgewählt.

Filteraspekt
: Die Filteraspekte beschreiben, wie eine Menge von Informationsobjekten durch ein `FilterTool` beschränkt wird. Eingerichtet wird ein Filteraspekt, indem seine Eigenschaft query mit einer Abfrage verbunden wird.

Dimensionsaspekt
: Durch Dimensionsaspekte wird abgebildet, wie Informationsobjekte durch ein `CubeTool` in einer n-dimensionale Struktur organisiert werden. Für die Einrichtung eines Dimensionsaspekts werden über die Methode `addDimension` vom Anwender Pfade eingegeben, durch die jeweils eine Dimension beschrieben wird.

7.3 Führungsaspekte

7.3.1 Allgemeine Eigenschaften und Klassifizierung

Bei der Zusammenstellung der Restriktionsaspekte wird der Anwender bereits in erheblichem Maß aktiv geführt. Gleichwohl liegt der Schwerpunkt der Führungsaspekte im ooMSS bei der informativen Führung. Dies erfolgt durch die Dokumentation der Zusammenhänge im Informationsmodell. Die Bereitstellung von Auswertungsobjekten (`ReportAspect`) und Bewertungsobjekten (`NoteAspect`) bilden dafür die Basis (siehe Abbildung 7.9).

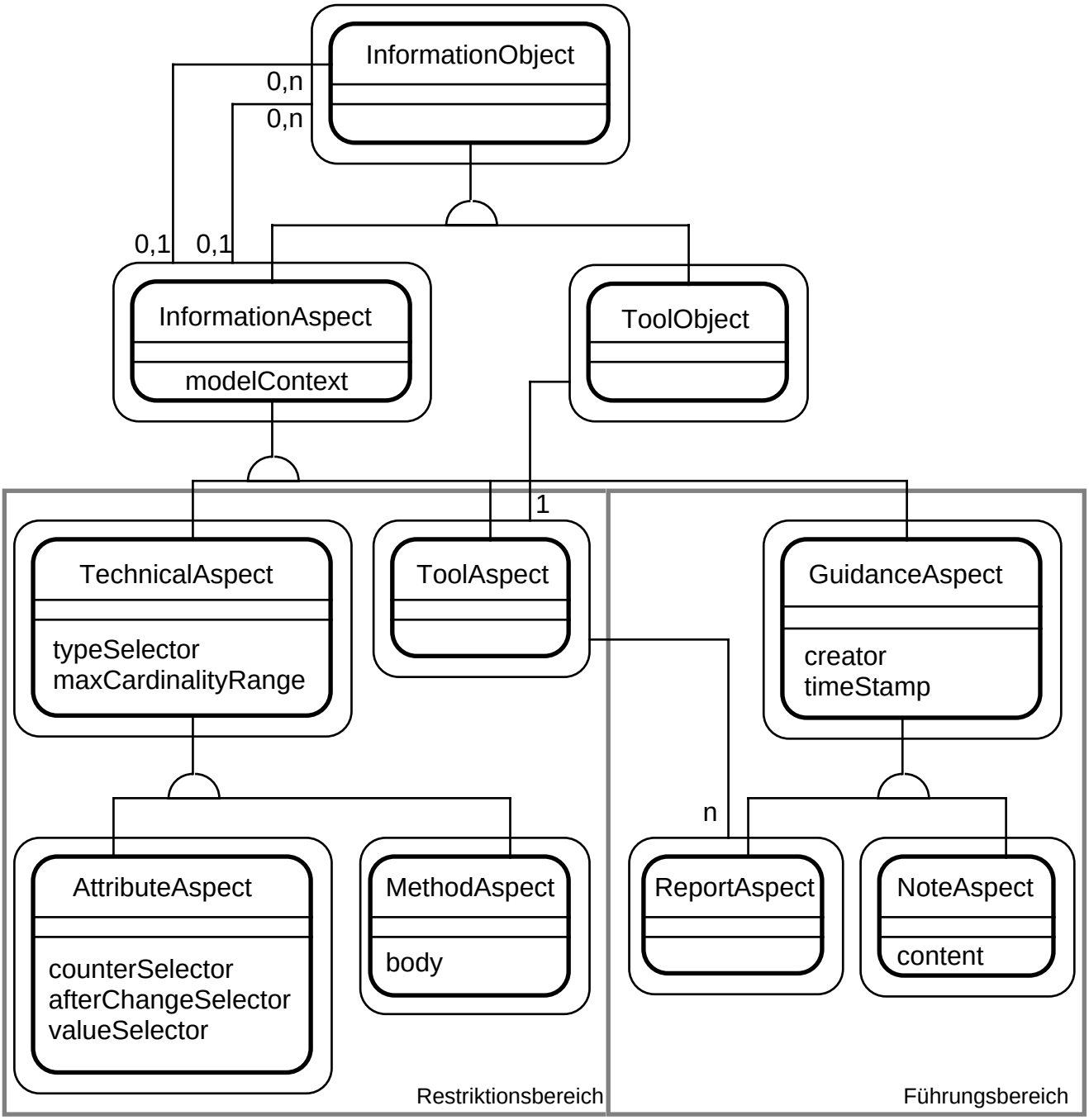

Abbildung 7.9: Objektmodell der Informationsaspekte mit GuidanceAspect (Objektnotation nach [CoYo94a])

Relevant sind für die Informationsobjekte beider Klassen einheitlich der Urheber (`creator`) und der Erstellungszeitpunkt (`timeStamp`). Beide Eigenschaften sind deshalb Bestandteil der abstrakten Oberklasse `GuidanceAspect`. Die Verwaltung des Urhebers dient dazu, individuelle Sichtweisen auf das ooMSS zu schaffen (neben den Möglichkeiten, die das Interaktionsmodell bereitstellt). Der zeitliche Bezug dokumentiert die Aktualität einer Information. Die Werte beider Eigenschaften werden bei der Erzeugung von Auswertungsobjekten und Bewertungs-

objekten automatisch gesetzt und lassen sich nachträglich nicht ändern.
Auf die abweichenden Eigenschaften beider Informationsaspektklassen (Auswertung und Bewertung) wird in den folgenden Abschnitten im Detail eingegangen.

7.3.2 Auswertungsobjekte

Auswertungen werden im ooMSS durch die Klasse `ReportAspect` abgebildet. Die fachliche Auswertung besteht aus einer Kombination der oben beschriebenen fachlichen Aspekte. Die Auswertungen unterstützen ihre Zusammenstellung, indem bei der Kombination der Klassenbezug berücksichtigt wird. Sie lassen sich um Aspekte ergänzen, die einen passenden Klassenbezug besitzen und können aktiv passende Aspekte anbieten.

Ausgangspunkt bei der Erstellung einer Auswertung ist die Wahl einer Klasse und einer mit ihr verbundenen Kennzahl in Form eines Aggregationsaspekts.

Z.B. können für die Auswertungen von Produkten nur die bei der Klasse `Produkt` als Aggregationsaspekt verwalteten Kennzahlen (Umsätze, Kosten, Gewinne, ...) verwendet werden.

Der verwendete Aggregationsaspekt einer Auswertung beeinflußt den Klassenbezug der weiteren Aspekte. Z.B. ist bei der Auswertung von Kundenumsätzen der Klassenbezug für die Filter- und Dimensionseigenschaften die Klasse `Auftrag`, d.h., es werden die Umsätze der Kunden dadurch ermittelt, daß über alle Aufträge der Kunden die Auftragsumsätze summiert werden. Alle bei der Auftragsklasse verwalteten Beschränkungen und Gruppierungen können für die Auswertung verwendet werden. Dazu zählen

(a) das Auftragsdatum,
(b) die Kunden,
(c) die Kundengruppen,
(d) die Produkte und
(e) die Produktgruppen.

Diese Filter- und Dimensionsaspekte besitzen einheitlich als Klassenbezug die Auftragsklasse. Die folgende Abbildung veranschaulicht diesen Zusammenhang (siehe Abbildung 7.10).

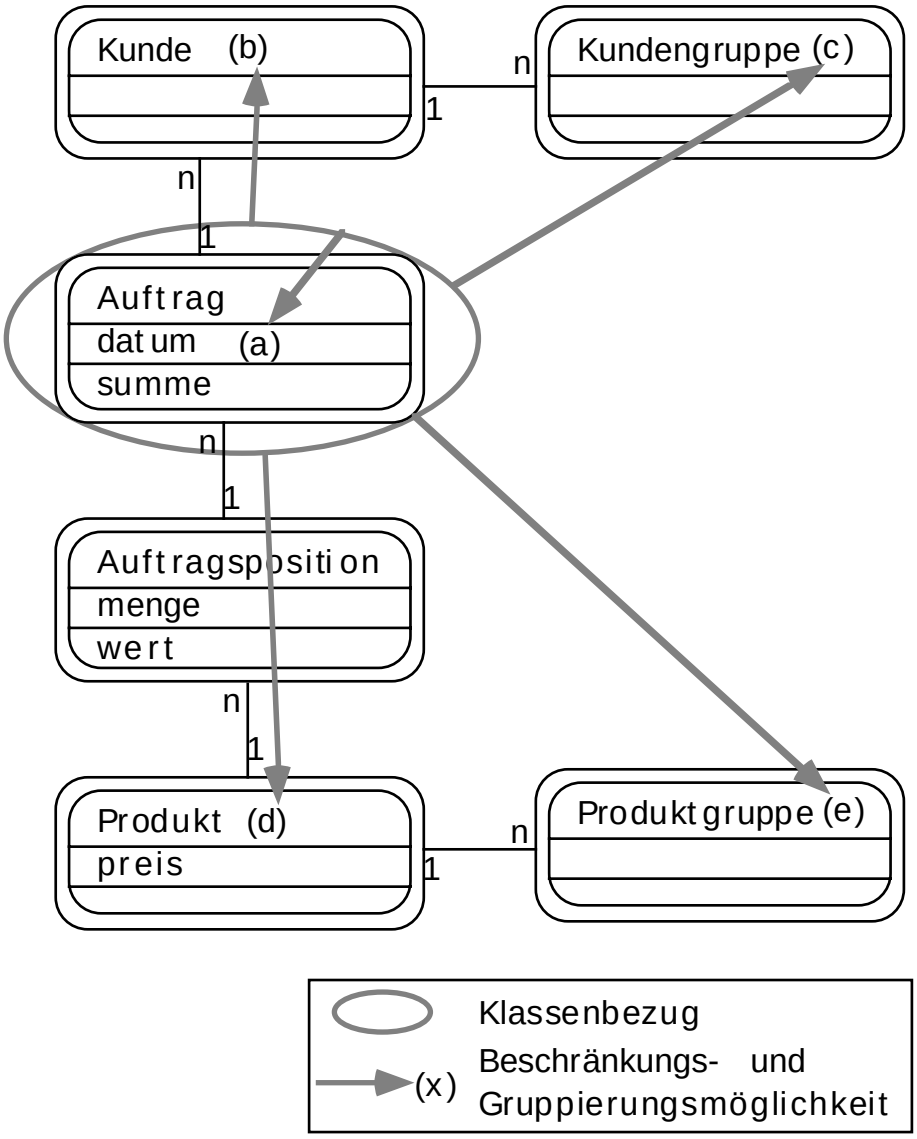

Abbildung 7.10: Auswertung Kundenumsatz
(Objektnotation in Anlehnung an [CoYo94a])

Für die Bereitstellung der beschriebenen Funktionalität besitzen die Auswertungsobjekte die folgenden Protokolle:

```
(aReportAspect := ReportAspect
    onAggregationAspect: anAggregationAspect)
```
=> eine neue Auswertung wird auf der Basis eines Aggregationsaspekts erzeugt.

```
(eineAuswertungKundenumsaetze := ReportAspect
    onAggregationAspect:
          (Kunde atToolAspect: #aggregierterUmsatz))
```
=> eine neue Auswertung der Kundenumsätze wird auf der Basis eines Aggregationsaspekts der Klasse `Kunde` erzeugt.

`(aReportAspect classContext)`
=> die Auswertung antwortet mit ihrem aktuellen Klassenbezug.

`(eineAuswertungKundenumsaetze classContext)`
=> die Auswertung antwortet mit ihrem aktuellen Klassenbezug, der in diesem Fall durch den Aggregationsaspekt `#aggregierterUmsatz` zu der Klasse `Auftrag` verschoben wurde.

`(aReportAspect additionalAspects)`
=> die Auswertung antwortet mit den möglichen fachlichen Aspekten, um die die Auswertung ergänzt werden könnte.

`(aReportAspect addAspect: aToolAspect)`
=> die Auswertung wird um einen passenden fachlichen Aspekt ergänzt.

`(eineAuswertungKundenumsaetze additionalAspects)`
=> die Auswertung antwortet mit Filter und Dimensionsaspekten, die die Auswertungsinhalte nach dem Auftragsdatum, den Kunden, den Kundengruppen, den Produkten und den Produktgruppen zusätzlich beschränken bzw. n-dimensional gliedern.

7.3.3 Bewertungsobjekte

Das zweite Standbein der informativen Führung im ooMSS basiert auf der expliziten Dokumentation der Informationszusammenhänge. Auf drei Arten wird dies geleistet:

1. Zum einen können Informationsobjekte, wie z.B. Auswertungen, von jedem Anwender subjektiv bewertet werden. Implizites Wissen wird im Sinne des Knowledge Managements explizit gemacht [Nona91 S.28f.]. Basis für die subjektive Bewertung ist die von der Oberklasse `GuidanceAspect` geerbte Eigenschaft `creator`, durch die für jede Bewertung der Urheber verwaltet wird.
2. Da ein Bewertungsobjekt selbst ein bewertbares Informationsobjekt darstellt, ist es weiterhin möglich, Bewertungen von Bewertungen zu verwalten. Diese Meta-Bewertungen sind ein wesentlicher Bestandteil für die Bildung von Wissensstrukturen. Bei [RiKM00 S.372] wird in diesem Zusammenhang die besondere Eignung der Objektorientierung beschrieben.

3. Die dritte Möglichkeit besteht darin, daß durch die Bewertungen Sequenzen von MSS-Auswertungsschritten dokumentiert werden, wie im MSS-Kriterienbereich (Abschnitt 2.3.2) gefordert wurde.

Modelliert wird ein Bewertungsobjekt durch die Klasse `NoteAspect`. Ein `NoteAspect` besitzt folgende Eigenschaften:

Der Bewertungsbezug ist die Eigenschaft, durch die beschrieben wird, auf welchen Bereich des Informationsmodells sich die Bewertung bezieht. Auf der Basis der von der Klasse `InformationAspect` geerbten Eigenschaft `modelContext` ist es möglich, beliebige Informationsobjekte und Informationsklassen zu bewerten. Klassen können dabei entweder direkt bewertet werden, oder die Bewertung besteht in der Betrachtung der Menge der Bewertungen über die Instanzen, auf die über das Selbstbeschreibungsprotokoll zugegriffen wird. Es lassen sich so z.B. die Stärken und Schwächen einer Auswertungsklasse dokumentieren.

Mit der Bewertungsaussage (`content`) wird der Inhalt des Informationsmodellausschnitts bewertet. Die Bewertungsaussage kann aus beliebigen Objekten bestehen. Die gesamte Bandbreite der im ooMSS zur Verfügung stehenden Datentypen ist nutzbar. Am häufigsten sind freie textuelle Bewertungen [Devl97 S.57].

Die einer Anwendung der Bewertungsobjekte zugrundeliegenden Protokolle sind folgendermaßen ausgeprägt:

```
(aNoteAspect := NoteAspect modelContext: anInformationObject.
aNoteAspect content: anObject)
```

=> Ein Bewertungsobjekt wird für einen Ausschnitt des Informationsmodells erzeugt und mit einer Bewertungsaussage verbunden.

```
(eineBewertung1 := NoteAspect modelContext: einAuswertung1.
   eineBewertung1 content: 'Durch die Auswertung ist deutlich
   erkennbar, daß die Umsätze der Produktgruppe Buch im Juni
   1999 im Vergleich zum Vorjahr signifikant zurückgegangen
   sind.')
```

=> Ein Bewertungsobjekt wird für eine Auswertung erzeugt und mit einer verbalen Bewertungsaussage verbunden.

```
(eineBewertung2 := NoteAspect modelContext: eineBewertung1.
eineBewertung1 content:
    (InformationCollection with:  'Ich möchte darauf hinweisen,
    daß dieser Trend nicht für die Region Osnabrück gilt.
    Beachten Sie dazu bitte die beigefügte Auswertung.' with:
    eineAuswertung2)
```

=> Ein Bewertungsobjekt wird für eine Bewertung erzeugt und mit einer verbalen Bewertungsaussage und einer weiteren Auswertung verbunden.

Im Rahmen der Anwendung des MSS wird das MSS-Modell mit seinen Führungsaspekten und dem Interaktionsmodell verbunden. Der Anwender kommuniziert dabei über die morph-basierte Benutzerschnittstelle mit den hier beschriebenen Informationsobjekten. Die Anwendung wird als Schwerpunkt im folgenden Anwendungsabschnitt behandelt.

8. Anwendung

8.1 Anwendungsdomäne

Nachdem in den Kapiteln 5-7 die einzelnen Teilmodelle beschrieben wurden, soll nun in Kapitel 8 das Gesamtkonzept angewendet werden. Hierfür wird eine Domäne verwendet, die in drei sechsmonatigen Praxisprojekten mit unterschiedlichen Untersuchungszielen von einem sechsköpfigen Projektteam erprobt wurde [RiMK98, BHJ+96]. Da hier nur die Hauptaspekte herausgearbeitet werden sollen, wurde die Domäne für eine bessere Übersicht vereinfacht (siehe Abbildung 8.1)

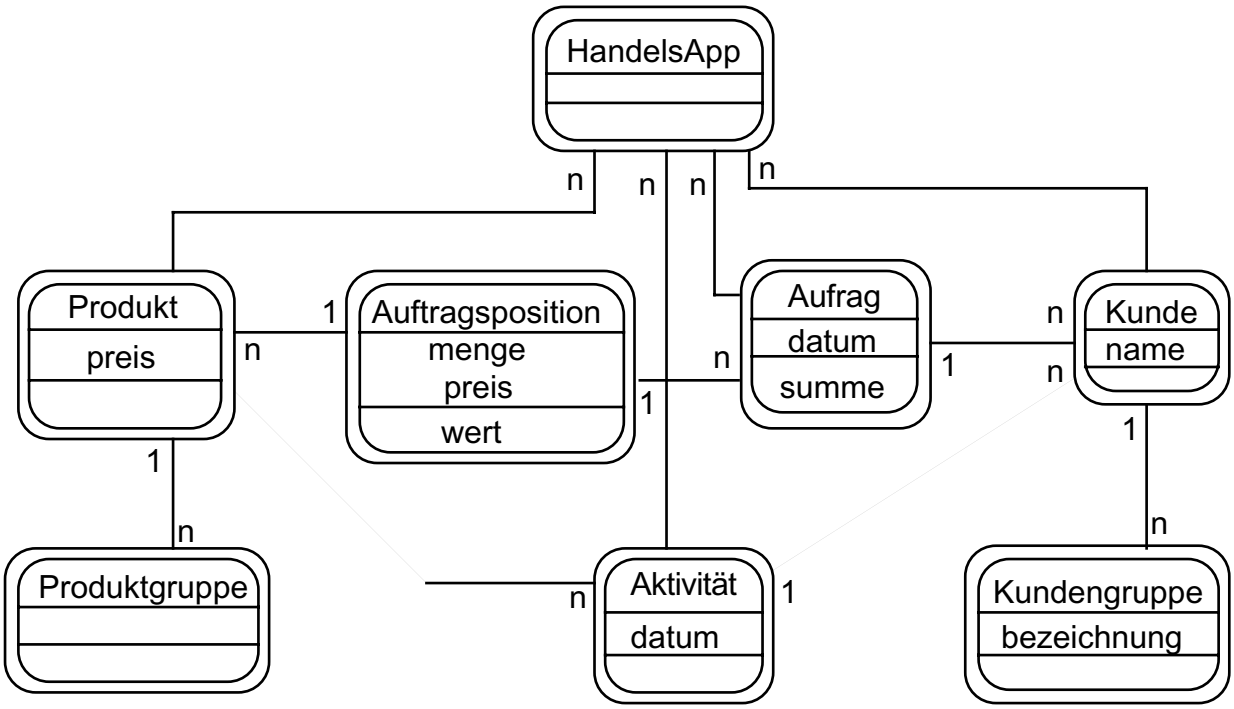

Abbildung 8.1: Objektmodell der Domäne
(Objektnotation nach [CoYo94a])

Durch das Domänenmodell wird ein Unternehmen, das mit Medien, also Büchern, CDs und Videos handelt, abgebildet. Alle Produkte, Kunden und Aufträge des Unternehmens werden direkt als Klasse modelliert. Die Instanzen der Klassen stehen jeweils in Beziehung zu einer Instanz der Klasse `HandelsApp`-(likation). Mit einer HandelsApp wird die gesamte Domäne für das Unternehmen repräsentiert. Zusätzlich dient sie als Ausgangspunkt für den Zugriff auf alle operativen und analytischen Objekte.
Ergänzt wird die Domäne um die Klasse `Aktivität`, mit der Aktionen, die im Rahmen von Marketing-Aktivitäten (Stichwort: Cross-Selling) durchgeführt wurden, modelliert werden. Bei den Aktionen werden Kunden auf Produkte hingewiesen, die sich noch nicht in ihrem Warenkorb befinden [Hasl96 S.22].

Die Klasse `Auftragsposition` ist notwendig, um die beidseitige multiple Assoziation zwischen den Aufträgen und Produkten mit zusätzlichen Eigenschaften zu ergänzen, z.B. die in einem Auftrag enthaltene Menge eines Produkts [Vett95 S.112]. Mengen von Produkten und Kunden werden jeweils zu Gruppen zusammengefaßt, die durch Instanzen der Klassen `Produktgruppe` und `Kundengruppe` abgebildet werden.
Die Anwendung des ooMSS für die beschriebene Domäne eines Medienhandels wird im folgenden in drei Stufen von Anwendergruppen beschrieben. Tabelle 8.1 zeigt dabei, welche Anwendergruppe welche Funktion ausführen darf.

Stufe	1	2	3
Anwendergruppe	Einfacher Anwender	Fachanwender	Entwickler
Abrufen von Auswertungen & Bemerkungen verwalten	√	√	√
Auswertungen verwalten	–	√	√
Aspekte verwalten	–	–	√

Tabelle 8.1: Übersicht auf die unterstützten Aktivitäten

8.2 Erste Anwendungsstufe: Abrufen von Auswertungen und Verwalten von Bemerkungen

Auf der ersten Anwendungsstufe ermöglicht das ooMSS dem Anwender existierende Auswertungen abzurufen. Vor dem Abrufen von Auswertungen muß der Anwender sich zunächst beim ooMSS anmelden. Durch die Anmeldung erkennt das System die Identität des Anwenders. Auf der Basis der Identität und der im System verwalteten Anwenderstufen wird festgelegt, inwieweit der Anwender das System verwenden kann. Dem Anwender werden die dazu passenden Interaktionsmöglichkeiten zur Verfügung gestellt. Je höher die Stufe eines Anwenders ist, desto mehr Anpassungsmöglichkeiten stellt das System ihm bereit. Bei einer Anpassung wird die Änderung des Systems zusammen mit der Identität des Anwenders protokolliert.

Anwender auf der ersten Stufe besitzen die Möglichkeit, über das Interaktionsmodell bestehende Auswertungen abzurufen und mit repräsentierten Auswertungen auf einfache Weise umzugehen. Die im ooMSS zur Verfügung stehenden Auswertungen werden in einem Morph-Fenster aufgelistet (siehe Abbildung 8.2).

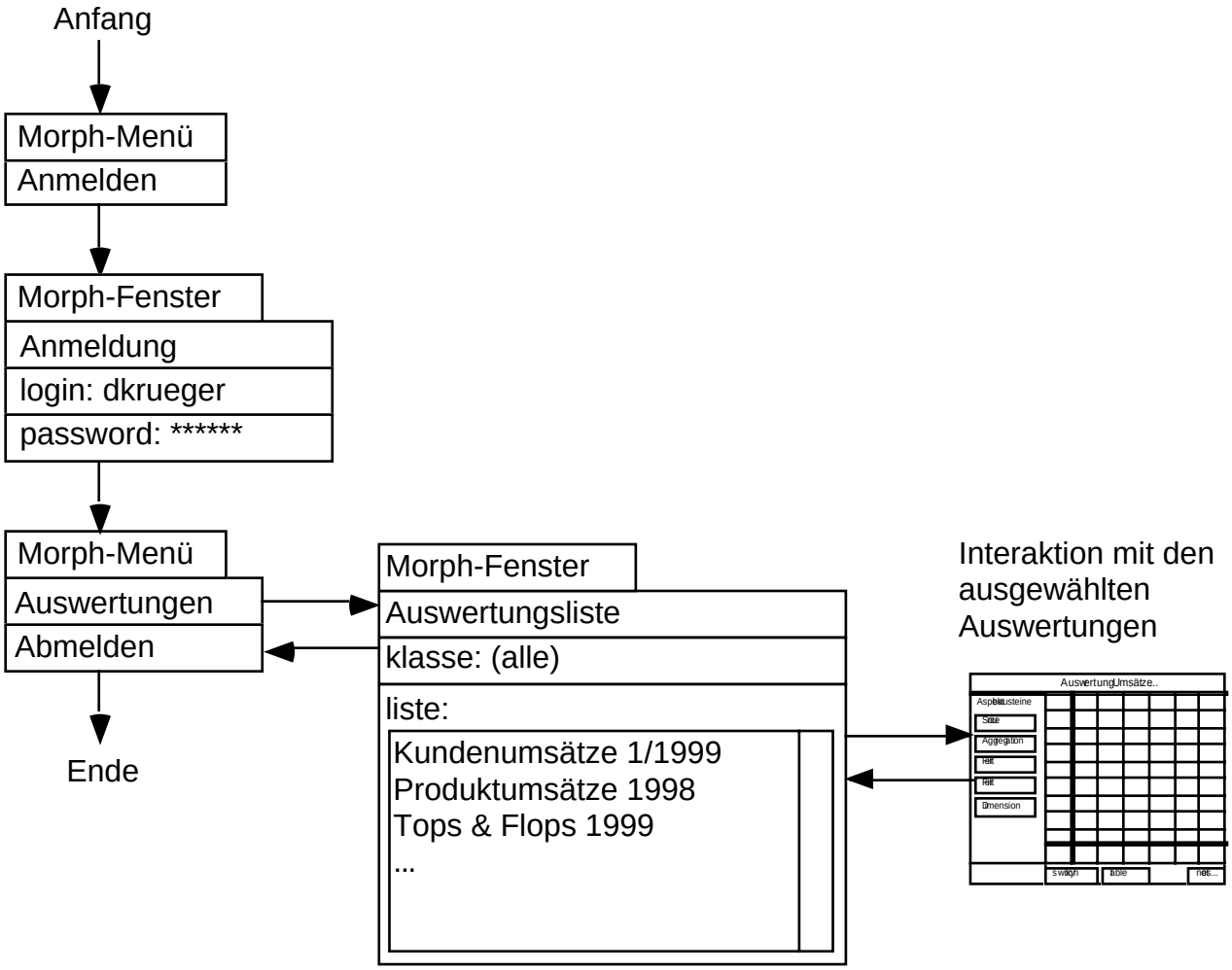

Abbildung 8.2: Anwendungsprozeß und Morph-Fenster

Über das Morph-Fenster hat der Anwender die Möglichkeit die Klasse, für die die Auswertungen aufgelistet werden, einzuschränken und die mit ihren Namen aufgelisteten Auswertungen (Klasse `ReportAspect`) auszuwählen. Der Anwender wird im ersten Schritt dadurch unterstützt, daß er auf die im System zur Verfügung stehenden Auswertungen selbst zugreifen kann.
Die für das Anwendungsbeispiel zur Verfügung gestellten Auswertungsinhalte eines Medienhandels sind domänenübergreifend oder besitzen einen Bezug zu einer Domänenklasse, wie `Kunde`, `Produkt` oder `Auftrag`. Umsatzwerte können allgemein oder Domänenklassen-spezifisch abgerufen werden. MSS-typische Fragestellungen sind z.B.

- Welches sind die umsatzstärksten Kunden,

- Mit welchen Kunden ist ein überdurchschnittlich hoher Umsatz verbunden,
- Was sind, bezogen aus den Umsatz, gute Produkte?
- Was ist der durchschnittliche Umsatz der Aufträge, an dem ein Produkt beteiligt ist?
- Wie sehen für die Produkte die Tops und Flops gemessen am Ertrag aus?

Mit den Auswertungsinhalten lassen sich Variationen, wie Gruppierungen nach Produktgruppen, Zeiträumen (z.B. Umsätze je Artikelgruppe pro Monat von zwei Jahren) oder Eigenschaften (wie z.B. Altersgruppe, Geschlecht) und Beschränkungen nach zeitlichen Kriterien, erstellen.
Für die hier beschriebene Anwendung muß das ooMSS den Zugriff auf die existierenden Auswertungen bereitstellen. Bei dem zugrundeliegenden Ablauf werden im ersten Schritt alle Unterklassen von der Klasse `DomainObject` ermittelt. Die Unterklassen sind Bestandteil der Selbstbeschreibungsschnittstelle einer Klasse. Im zweiten Schritt werden die Auswertungen bestimmt, die zu allen Klassen oder einer ausgewählten Klasse gehören. Jede Klasse antwortet, ebenfalls als Eigenschaft ihrer Selbstbeschreibung, mit ihren Auswertungen. Das folgende Interaktionsdiagramm beschreibt diesen Vorgang (siehe Abbildung 8.3).

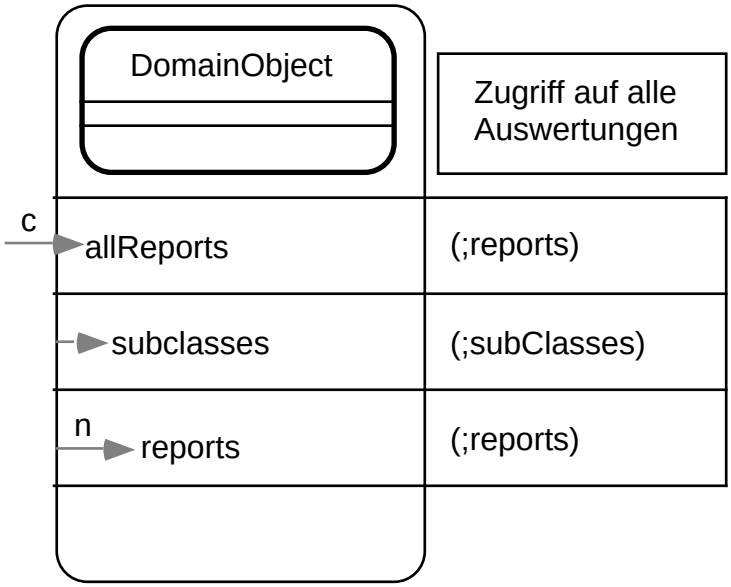

Abbildung 8.3: Interaktionsdiagramm für den Zugriff auf die Auswertungen (Notation in Anlehnung an [AlBW98 S.377])

Mit der Selektion einer Auswertung wird diese im Interaktionsmodell durch ein Morph angezeigt und kann durch den Anwender in seiner Umgebung plaziert werden (siehe Abbildung 8.4).

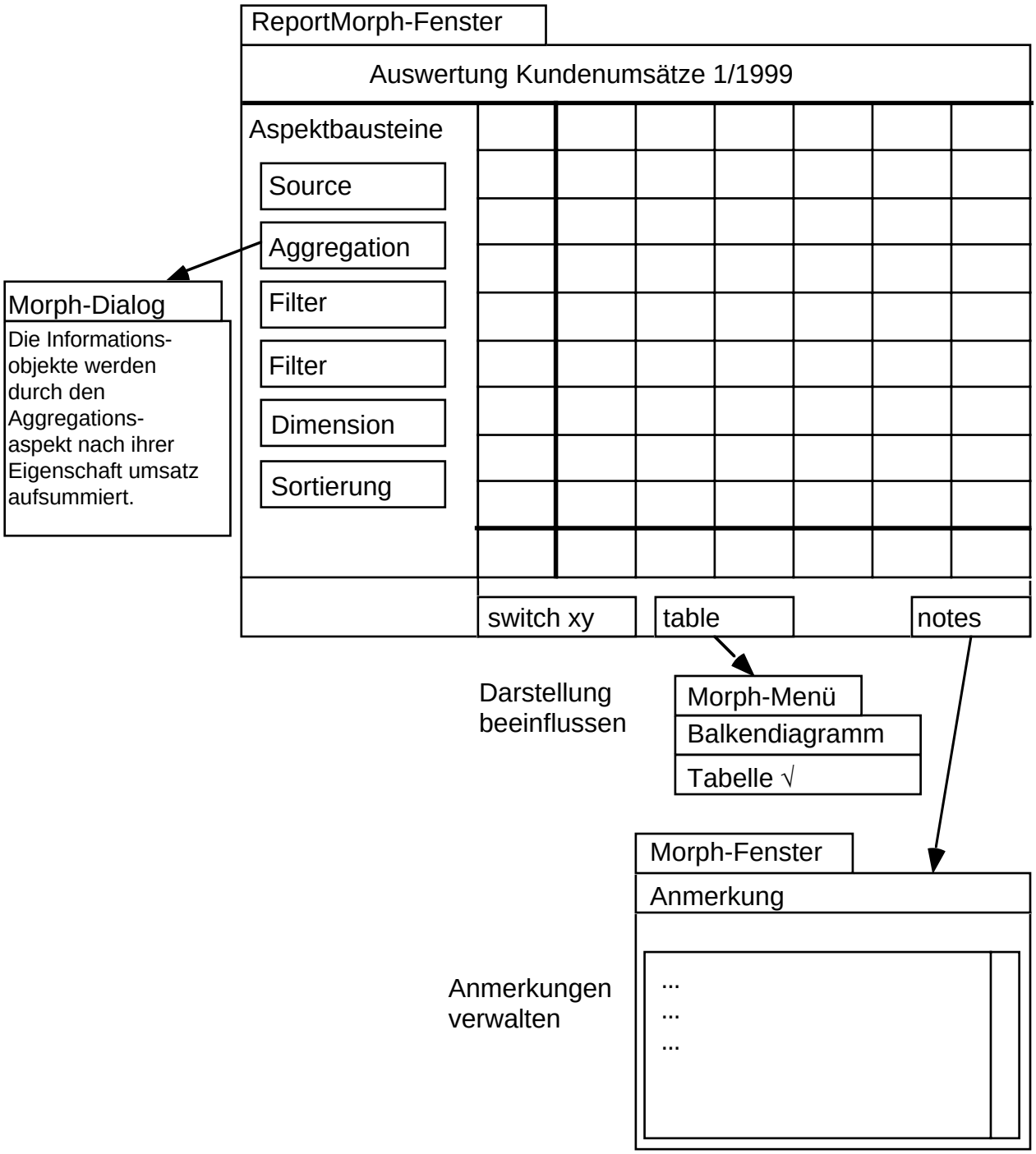

Abbildung 8.4: Anwendung ReportMorph (Stufe 1)

Der Morph visualisiert links alle fachlichen domänenspezifischen Aspektbausteine, aus denen sich die Auswertung zusammensetzt. In der Mitte ist das Auswertungsergebnis und im unteren Bereich sind die grundlegenden Interaktionsmöglichkeiten des Anwenders enthalten.

Der Anwender hat die Möglichkeit, für jeden im linken Bereich enthaltenen fachlichen Aspektbaustein eine detaillierte Erläuterung zu erhalten. Zu diesem Zweck erscheint auf der GUI solange ein Dialog-Morph, wie sich der Mauszeiger über einem Aspekt-Morph befindet.

Standardmäßig werden im Rahmen der Anwendung alle Informationsobjekte, die Bestandteil der `HandelsApp` sind, durch jeweils ein `SourceAspect` bereitgestellt. Dies sind die Objekte der Klassen `Kunde`, `Kundengruppe`, `Produkt`, `Produktgruppe`, `Auftrag` und `Aktion`. Beispiele für die durch Aggregationsaspekte beschriebenen Kennzahlen der Domäne sind die Kundenumsätze, die Anzahl der Kunden und die Produktumsätze. Die Sortieraspekte erstellen Rankings, indem sie z.B. auf die mit Kunden und Produkten verbundenen Umsätze Bezug nehmen. Die Filteraspekte beschränken z.B. dadurch, daß nur Kunden mit einem bestimmten Alter oder Produkte, die zu einer bestimmten Produktgruppe gehören, beschrieben werden. Die bei den Dimensionsaspekten verwendeten Kriterien stimmen i.d.R. mit denen der Filteraspekte überein. Die Kunden können nach ihrem Alter, die Produkte nach ihrer Produktgruppe gruppiert werden.

Das Auswertungsergebnis wird mit einer passenden Darstellungsart im mittleren Bereich angezeigt: Ein einzelner Zahlenwert durch ein Textfeld, eine Menge von Objekten durch eine Liste, eine Menge von Zahlen durch eine Liste oder ein Balkendiagramm und ein zweidimensionaler Ausschnitt einer n-dimensionalen Ergebnismenge durch eine Tabelle.
Ausgehend von einer bestehenden Auswertung werden neue Auswertungen durch die Selektion (Zelle, Spalte, Zeile, Balken) eines Ausschnitts des Ergebnisses erzeugt. Diese Funktionalität ermöglicht dem Anwender durch analytische Informationsstrukturen, wie durch operative Domänenstrukturen zu navigieren.

Im unteren Bereich kann der Anwender das Auswertungsergebnis in seiner Darstellung beeinflussen und kommentieren. Für eine zweidimensionale Darstellung kann der Anwender die abgetragenen Dimensionen durch den Widget-Knopf “`switch xy`” austauschen. Wenn für ein Ergebnis alternative Darstellungsarten möglich sind, kann er diese im unteren Bereich wählen. Die Kommentare bezüglich der angezeigten Auswertung in Form eines Notes-Morph lassen sich über den Knopf “`notes...`” abrufen. Über den Notes-Morph können zusätzlich neue Kommentare erstellt und damit systemweit zur Verfügung gestellt werden. Als besonders relevante Anwendungsform wird, wie im Kriterienbereich gefordert, die Möglichkeit unterstützt, Sequenzen von MSS-Analyseschritten zu dokumentieren.

Nachdem der Anwender seine Arbeit mit dem System beendet hat, verläßt er es, indem er sich durch das Menü abmeldet (siehe Abbildung 8.2). Bei der nächsten Anmeldung findet der Anwender seine Arbeitsumgebung so vor, wie er sie verlassen hat. Seine individuelle Anpassung des ooMSS über das Interaktionsmodell bleibt erhalten.

8.3 Zweite Anwendungsstufe: Erzeugung und Verwaltung von Auswertungen

In der zweiten Stufe werden durch Fachanwender mit geringem Informationstechnologie-Wissen neue Auswertungen erstellt und bestehende angepaßt. Neue Auswertungen werden durch das Menü des Auswertunglistenmorph erzeugt (siehe Abbildung 8.5).

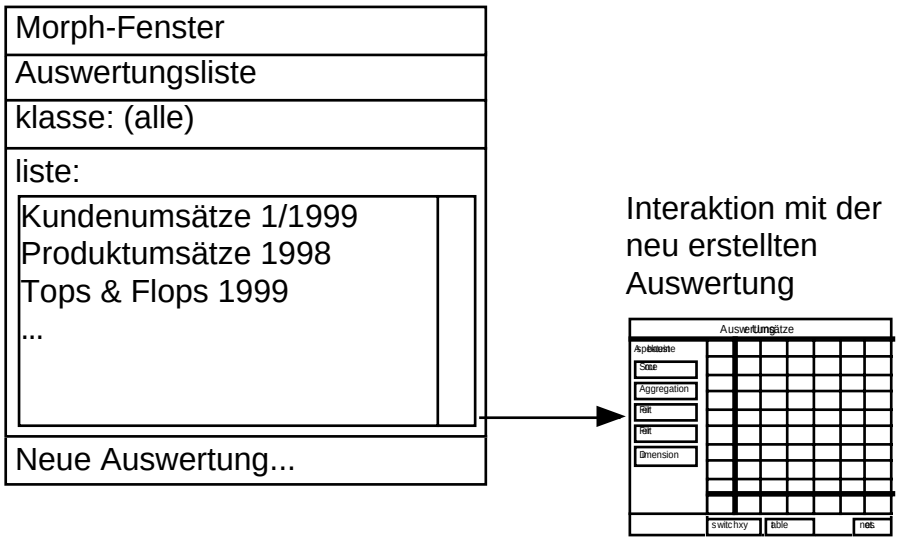

Abbildung 8.5: Auswertung erstellen

Der Fachanwender, der eine neue Auswertung im ooMSS erstellt, ist der Eigentümer der Auswertung. Standardmäßig können alle Anwender eine Auswertung abrufen und der Eigentümer sie verändern. Dem Eigentümer ist es zusätzlich gestattet, die Zugriffsrechte der übrigen Anwender zu beschränken oder zu erweitern. Durch eine Beschränkung kann eine Auswertung nicht mehr abgerufen werden. Eine Erweiterung der Zugriffsrechte ermöglicht, daß die Auswertung frei änderbar ist.
Bestehende und für ihn zugängliche Auswertungen kann der Fachanwender aus dem Auswertunglistenmorph selektieren. Besitzt er nur einen lesenden Zugriff, kann er sich von der Auswertung eine 1:1 Kopie machen. Die nach dem Prototyp-Pattern erstellte Kopie kann er anpassen, wobei das Original unverändert bleibt [GHJV95 S.117]. Die Verwendung einer konkreten Vorlage als Ausgangs-

punkt einer Auswertung ist eine zusätzliche natürliche Unterstützung für den Fachanwender, weil bei der Anwendung des ooMSS auf abstrakte Klassen-Vorlagen verzichtet werden kann.
Auswertungen werden durch die Veränderung ihrer Aspektbausteine angepaßt. Bei zwei grundlegenden Arten der Anpassung wird der Anwender durch das ooMSS aktiv unterstützt. Zum einen kann der Fachanwender die Aspektbausteine einer Auswertung als Einheit austauschen, zum anderen die einzelnen Attributwerte eines Aspekts verändern:

- Bei der Veränderung einer Auswertung durch das drag & drop von Aspektobjekten in die Aspektleiste wird die einer Auswertung zugrundeliegende Syntax der technischen und fachlichen Aspekte beachtet. Berücksichtigt wird im einzelnen, an welcher Position ein Aspekt in der Aspektleiste einzuordnen ist, ob ein Aspekttyp ein- oder mehrfach Bestandteil einer Auswertung sein kann und ob ein Aspekt ein notwendiger Bestandteil einer Auswertung ist.
  Die Aspekte einer Auswertung werden in der festen Reihenfolge Source, Aggregation/Sortier, Filter und Dimension angeordnet. Bei der Zusammenstellung wird die Reihenfolge durch die Schnittstelle eingehalten und die neue mögliche Position in der Aspektleiste visualisiert. Von einem SourceAspect, AggregationAspect, SortAspect ist jeweils nur eine Ausprägung an einer Auswertung beteiligt. Wird ein Aspekt auf die Leiste bewegt, wird angezeigt, daß ein bestehender Aspekt ausgetauscht wird. Zusätzlich wird berücksichtigt, daß Aggregations- und Sortieraspekte in einer Auswertung nur alternativ angewendet werden können (siehe Abbildung 8.6).

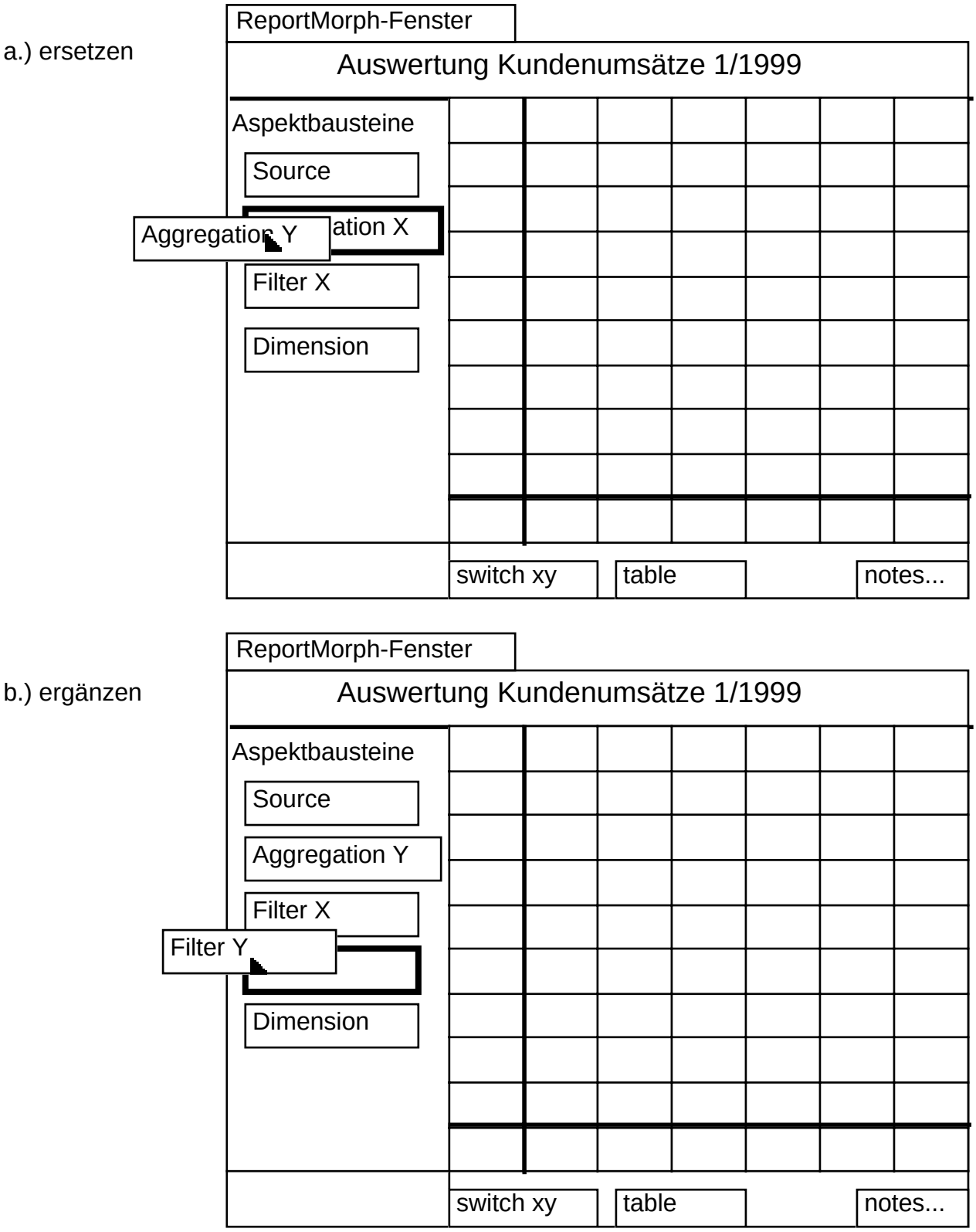

Abbildung 8.6: Verwaltung von Aspekten

Bei den mehrfach vorkommenden `FilterAspect` und `Dimension-Aspect` hat der Anwender die Wahl, sie zu ersetzen (a) oder zu ergänzen (b). Ein `SourceAspect` ist notwendig. Er läßt sich nur gegen einen anderen `SourceAspect` austauschen und nicht einfach entfernen.

- Für die zweite Art der Anpassung werden die Attributwerte der beteiligten Aspektbausteine modifiziert. Das ooMSS führt den Anwender, indem es ihm für die Aspektwerte über ein Menü alternative Attributwerte anbietet. Die der Auswertung zugrundeliegenden Tools werden so angepaßt (siehe Abbildung 8.7).

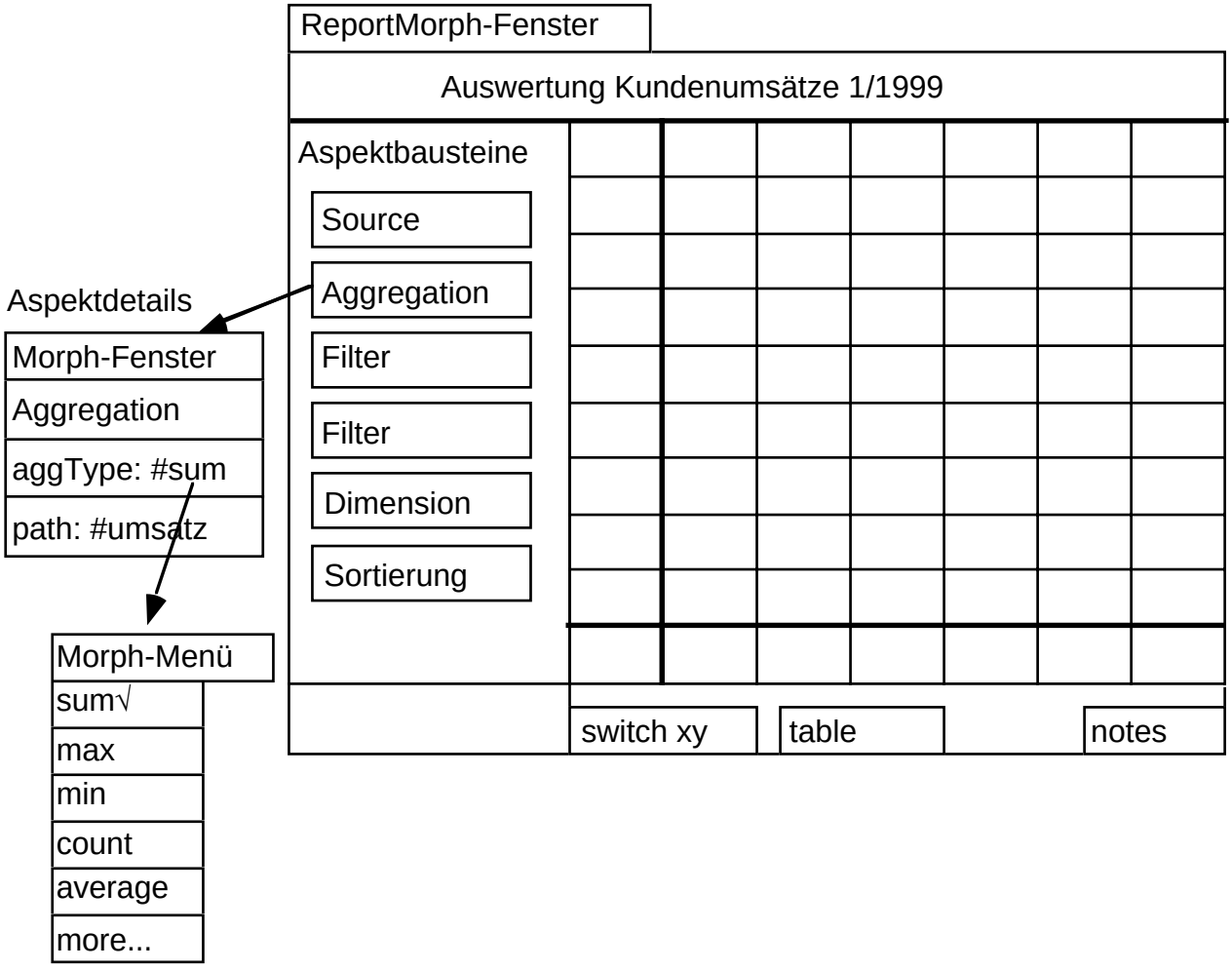

Abbildung 8.7: Anwendung ReportMorph (Stufe 2)

Im einzelnen hat der Fachanwender die Möglichkeit, die jeweiligen Pfade der Aspekte und zusätzlich bei einem Aggregationsaspekt die Aggregationsart, bei einem Sortieraspekt die Sortierfolge und bei einem Filteraspekt die Filterwerte zu ändern. Bei der Änderung werden vom System aspektspezifische technische Beschränkungen beachtet, wie z.B., daß bei einem Aggregationsaspekt ein Pfad auf eine aggregierbare, d.h.

eine numerische, Eigenschaft verweist, wenn die Aggregationsart keine einfache Aufzählung (`count`) ist.

Neue Auswertungen und Auswertungskopien können aus der zweiten Anwendungsstufe systemweit gespeichert und damit übergreifend zugänglich gemacht werden. Dazu werden die neuen Auswertungen einfach per drag & drop in den Auswertungslistenmorph bewegt (siehe Abbildung 8.8).

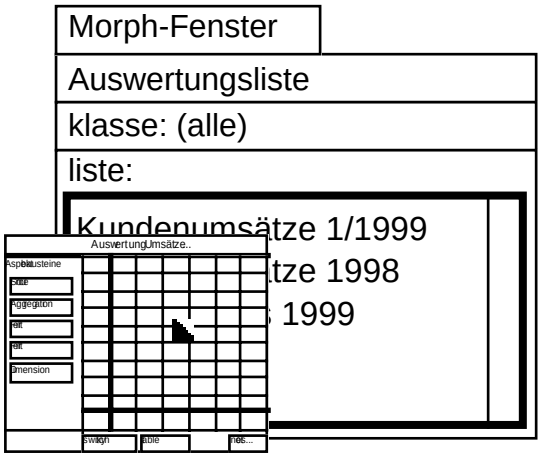

Abbildung 8.8: Speicherung einer Auswertung

8.4 Dritte Anwendungsstufe: Verwalten der Aspekte

Der Anwender mit Informationstechnologie-Kenntnissen eines Entwicklers ruft über den AspectBrowserMorph die im Informationsmodell verwalteten fachlich aussagekräftigen Kombinationen von Domänen- und Werkzeugobjekten ab. Der AspectBrowserMorph enthält drei horizontal angeordnete Listen (siehe Abbildung 8.9).

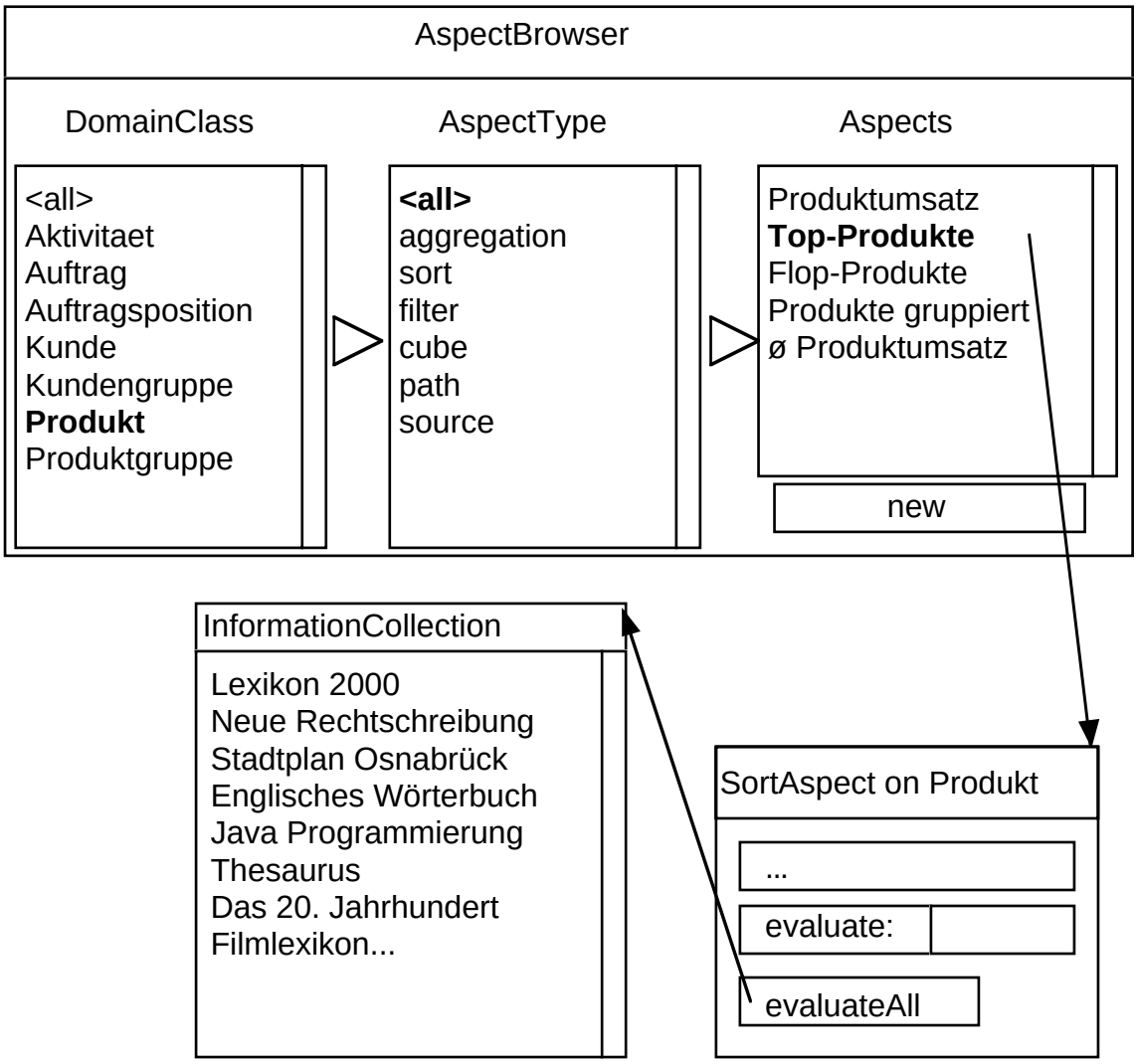

Abbildung 8.9: Aufbau des AspectBrowserMorphs

Mit der linken und der mittleren Liste wird beeinflußt, welche konkreten Aspekte in der rechten Liste angezeigt werden. Die linke Liste enthält dazu alle Domänenklassen. Mit einer Selektion einer Domänenklasse werden nur die Auswertungsaspekte berücksichtigt, die mit der Klasse verbunden sind. Alternativ wird mit der Selektion der Bezeichnung `<all>` eine Beschränkung auf eine Klasse vermieden. In der mittleren Liste wird durch die Auswahl einer Aspektart die Anzahl auf diese Aspektart beschränkt. In der Abbildung werden z.B. von der Klasse `Produkt` alle Aspektarten angezeigt.

Neue Aspekte lassen sich über den unter der Aspektliste plazierten Widget-Knopf “`new`” erzeugen. Mit der Betätigung des Knopfs wird ein Informationsmorph für den neuen Aspekt geöffnet. Der Anwender spezifiziert über den Morph die Eigenschaften des Aspekts und speichert ihn, indem er seinen Morph in die Aspektliste verschiebt.

Auf die Eigenschaften der bestehenden Aspekte kann der Anwender ebenfalls über einen Informationsmorph zugreifen. Als besonders hilfreiche Eigenschaften für das Austesten von Aspekten außerhalb von Auswertungen enthalten die Informationsmorphs die visualisierten Methoden “`evaluate:`” und “`evaluateAll`”. Sie ermöglichen es, den Aspekt eigenständig auf eine Menge von Informationsobjekten direkt anzuwenden und ihn so interaktiv zu testen. Bei “evaluate:” wird als Parameter die Informationsobjektmenge explizit angegeben, bei “`evaluateAll`” werden alle Instanzen der selektierten Domänenklasse verwendet. Im Beispiel werden durch den Aspekt„Top-Produkte“ alle Produkte abgerufen (siehe Abbildung 8.9).

9. Fazit

Die Ergebnisse der Arbeit werden im folgenden Abschnitt kurz zusammengefaßt. Das vorgestellte Konzept eines ooMSS bietet mehrere Erweiterungsmöglichkeiten. Im Ausblick wird auf konzeptionelle und die dafür notwendigen technischen Ergänzungen eingegangen.

9.1 Zusammenfassung

Im Rahmen der Arbeit ist es gelungen, durch die konsequente Anwendung der objektorientierten Konzepte die spezifische Funktionalität von MSS homogen abzubilden. Zu diesem Zweck wurden die Potentiale des objektorientierten Paradigmas für die Konzeption eines MSS identifiziert und angewendet. Dazu wurde zunächst ein Kriterienkatalog aufgestellt, der sich an den spezifischen integrativen Anforderungen an ein MSS orientiert. Ausgehend von grundlegenden Software-Qualitätskriterien umfaßt der Katalog zusätzlich das Natürlichkeitskriterium für eine anwendernahe Vermittlung der Konzepte und Elemente eines MSS und spezielle MSS-Kriterien, durch die der Umgang mit Kombinationen von MSS-relevanten Informationen und Funktionen (Restriktion, Führung, Anpassung) beschrieben wird.

Als Grundlage des Entwurfkonzepts wurde das objektorientierte Paradigma auf Konzeptebene beschrieben und bzgl. seiner Anwendbarkeit im MSS-Bereich analysiert. Als relevante Eigenschaften der Objektorientierung für das Anwendungsgebiet wurden die Komplexitätsbewältigung, die natürliche Modellierungssichtweise und die Unterstützung durch Entwurfsmuster herausgestellt.

Im Kernbereich der Arbeit stand das entwickelte MSS-Konzept, das in Form eines konkreten ooMSS dargestellt wurde. Die einzelnen Modellbereiche Informationsmodell, Interaktionsmodell und MSS-Modell wurden beschrieben und zum Kriterienkatalog in Beziehung gesetzt. Als Ergebnisse in den drei Modellbereichen sind zu nennen:

Mit dem Informationsmodell wurde eine leistungsfähige, qualitative und natürliche Grundlage für das ooMSS geschaffen. Die Informationsobjekte besitzen dazu erweiterte Eigenschaften im Bereich ihrer Selbstbeschreibung, der Eigenschaftsverwaltung und der Verwaltung von Informationsobjektmengen. Durch den Werkzeugbereich des Informationsmodells wurden die Elemente des ooMSS bereitgestellt, mit denen sich getrennt von dem speziellen Domänenbereich auf einfache Weise analytische Beziehungen und Auswertungen abbilden lassen.

Da das Informationsmodell frei von reinen Informationstechnologie-Konzepten und nach dem Anwender ausgerichtet wurde, kann das Interaktionsmodell durch eine direkte Übertragung eine natürliche Sicht auf die Domäne bereitstellen. Das Interaktionsmodell verwendet auf der Basis des Morphic-Framework ausgewählte Aspekte der physikalischen Realität und unterstützt so die natürliche Wahrnehmung des Anwenders. Zusätzlich ermöglicht es dem Anwender, sich selbständig individuelle Sichten auf das Informationsmodell einzurichten.

Durch das MSS-Modell als Meta-Ebene des Informationsmodells wurden neben technischen Möglichkeiten fachliche Inhalte in Form von Auswertungen und Bemerkungen in das ooMSS integriert. Die Auswertungen sind als Kombinationen von Informationsobjekten die analytischen Grundbausteine des MSS. Die Bemerkungen sind die Basis für die Verwaltung des Anwenderwissens über die Zusammenhänge zwischen den Informationsobjekten.
Damit basiert das ooMSS nicht auf heterogenen Werkzeugen, wie sie für konventionelle MSS-Systeme eingesetzt werden. Die gesamte Funktionalität wurde integriert, auf Basis von Objekten zur Verfügung gestellt. Weiterhin zeigt sich die Leistungsfähigkeit des entstandenen Systems darin, daß spezielle MSS-Funktionalitäten miteinander kombinierbar sind. Multidimensionale Auswertungen sind z.B. nicht mehr auf quantitative Elemente beschränkt, sondern lassen sich auf beliebige Inhalte anwenden.

Abschließend wurde die Anwendung des Gesamtkonzepts in Form von MSS-Unterstützungsszenarien auf den Anpassungsebenen Fachanwender, Fortgeschrittener und Entwickler dokumentiert. Die verwendeten Szenarien mit ihrem zugrundeliegenden Domänenmodell wurden in konkreten Praxisprojekten erprobt. Mit diesem Bezug wurde die praktische Relevanz des im Rahmen dieser Arbeit erstellten Konzepts eines ooMSS verdeutlicht.

9.2 Ausblick

Konzeptionell läßt sich das ooMSS, neben den in Kapitel 8 beschriebenen Anpassungsmöglichkeiten, um zusätzliche Werkzeuge und Repräsentationsformen ergänzen. Mit der Erweiterung des Werkzeugumfangs, z.B. durch statistische Auswertungen, lassen sich dabei zusätzliche Anwendungsgebiete erschließen. Das im Werkzeugmodell beschriebene CubeTool kann dabei als Beispiel für die Anbindung komplexer Berechnungsverfahren dienen. Die durch das Interaktionsmodell bereitgestellten Darstellungsformen lassen sich um weitere Formen von Geschäftsgrafiken ergänzen. Z.B. könnten die n-dimensionalen Informationen des CubeTools gleichzeitig mit mehr als zwei Dimensionen repräsentiert werden. Die verwendete Morph-GUI erlaubt auf einfache Weise die Zusammen-

stellung neuer Darstellungsformen. Die vom ooMSS standardmäßig angebotenen Möglichkeiten, also Tabellen und Balkendiagramme, können hierfür mit ihren Protokollen als Vorlage dienen.

Auch ist ein vollständiger Wechsel der Domäne auf einfache Weise möglich, weil die durch das ooMSS zur Verfügung stehenden Modellstrukturen bis zu einem bestimmten Grad allgemein verwendbar sind und zum anderen die generischen von domänenspezifischen Konzepten eindeutig getrennt wurden.

Eine qualitative Erweiterung des Anwendungsbereichs wäre der Wechsel des Informationsmodells von einem Erklärungsmodelltyp zu einem Entscheidungsmodelltyp. Das Ziel wäre eine Art Lernumgebung, in der der Anwender aktiv durch Anpassungen der Zustände und Strukturen des Informationsmodells Einblicke über die MSS-spezifischen Zusammenhänge erhalten würde. Auf der technischen Seite würde dies eine Anpassung der grundlegenden Eigenschaften der Informationsobjekte notwendig machen. Das im ooMSS konsequent angewendete objektorientierte Paradigma ermöglicht dies. Auf der Basis der vorhandenen Mechanismen zur Sicherung der Struktur und Zustände der Informationsobjekte sind Modifikationen möglich, die es dem Anwender erlauben, für sich individuell Zustände und Strukturen von Informationsobjekten zu ändern, ohne die Korrrektheit des gesamten Systems zu gefährden. Ähnlich wie ein Anwender durch das Interaktionsmodell eine individuelle Sicht auf das Informationsmodell erhält, könnten zusätzlich alternative Szenarien auf das Informationsmodell verwaltet werden. Der Anwender könnte durch eine freie individuelle Anpassung von Zuständen und Strukturen ein Modellgestaltungsszenario schaffen und testen.

Übergreifend könnten die von den einzelnen Anwendern gemachten Erfahrungen durch eine über das Interaktionsmodell bereitgestellte einheitliche Sicht vermittelt werden. Die gemeinsame Sicht würde sich aus den von den Anwendern erstellten und für die Anwendergruppe explizit bereitgestellten individuellen Sichten zusammensetzen. Die gemeinsame Sicht wäre eine Ergänzung zu den individuellen Sichten, die die Zusammenarbeit der Anwender unterstützen würde. Technische Voraussetzung dafür wäre neben der beschriebenen Erweiterung des Informationsmodells eine Erweiterung des Interaktionsmodells, wobei auch in diesem Fall auf vorhandene Multi-User-Konzepte des Morphic-Frameworks zurückgegriffen werden könnte.

Literaturverzeichnis

[Abit97]
Abiteboul, Serge: Quering Semi-Structured Data, in: Proceedings of the International Conference on Database Theory, Delphi, 1997, <ftp://ftp.inria.fr/INRIA/Projects/verso/VersoReport-103.ps.gz> Stand: 15.12.1999.

[AlBW98]
Alpert, Sherman R.; Brown, Kyle; Woolf, Bobby: The Design Pattern Smalltalk Companion, Reading, Massachusetts 1998.

[AIS+77]
Alexander, Christopher; Ishikawa, Sara; Silverstein, Murry; Jacobson, Max; Fiksdahl-King, Ingrid; Angel, Shlomo: A Pattern Language, New York 1977.

[Alte96]
Alter, Steven: Information systems: a management perspective, 2nd ed., Menlo Park, California 1996.

[Appe95]
Appelfeller, Wieland: Wiederverwendung im objektorientiertem Softwareentwicklungsprozeß, dargestellt am Beispiel der Entwicklung eines Lagerlogistiksystems, Frankfurt am Main 1995.

[Appl87]
Apple Computer, Inc.: Human Interface Guidelines: The Apple Desktop Interface, Reading, Massachusetts 1987.

[ArHS92]
Artim, J.M.; Hary, J.M.; Speckhoff, F.J.: Dienste der Benutzerschnittstelle in AD/Cycle, in: Corzilius, Rudolf [Bearbeitung]: AD-Cycle: Ziele, Konzepte und Funktionen, Oldenbourg 1992, S.137-160.

[Balz98]
Blazert, Helmut: Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Heidelberg 1998.

[Beck97]
Beck, Kent: Smalltalk best practices patterns, Upper Saddle River, NJ 1997.

[BHJ+96]
Brinker, Silvia; Helms, Detlef; Jarchau, Dirk; Klakstein, Stefanie; Krüger, Dietmar; Lönker, Carsten; von Maur, Eitel; Trentmann, Jörg:
Führungsinformationssysteme in der Praxis - Projektbericht SS 1996, Universität Osnabrück, 1996, unveröffentlicht.

[Born79]
Borning, Alan Hamilton: ThingLab -- A Constraint - Oriented Simulation Laboratory, http://www.2share.com/thinglab/ThingLabReport.zip, Stand: 22.02.00.

[Bune97]
Buneman, Peter: Semistructured Data, in: roceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona 1997, S.117-121.

[ChGl99]
Chamoni, Peter; Gluchowski, Peter: Analytische Informationssysteme - Einordnung und Überblick, in: Gluchowski, Peter: Chamoni, Peter: Analytische Informationssysteme: Data Warehouse, On-Line Analytical Processing, Data Mining, 2. Auflage, Berlin 1999.

[Choo98]
Choo, Chun Wei: Information Management for the Intelligent Organisation, Medford, NJ 1998.

[Clau96]
Claus, Thorsten: Objektorientierte Simulation und Genetische Algorithmen zur Produktionsplannung und -steuerung, Frankfurt am Main 1996.

[Coat96]
Coates, Elliot: Broad New Vistas, http://www.access.digex.net/~ell/pragma.html, Stand: 12.12.1996.

[CoCS93]
Codd, E.F.; Codd, S.B.; Salley, C.T.: Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT-Approach, White Paper, Arbor, 1993.

[Codd70]
Codd, E.F.: A Relational Model for Large Shared Data Banks, in: Communication of the ACM, Volume 13 Number 6, S.370-387 (1970).

[CoMa96]
Coad, Peter; Mayfield, Mark: Java design: building better apps and applets, Upper Saddle River, NJ 1996.

[CoMM97]
Connelly, Richard; McNeill, Robin; Mosimann, Roland: The multidimensional Manager, Ottawa 1997.

[CoNi94]
Coad, Peter; Yourdon, Edward: Objektorientierte Programmierung, München 1994.

[CoNo86]
Cox, Brad J.; Andrew J. Novobilski: Object-Oriented Programming - An Evolutionary Approach, 2. Auflage, Reading Massachusetts 1986.

[Copl00]
Coplien, James O.: Multi-Paradigm Design, Brüssel 2000, http://www.bell-labs.com/~cope/Mpd/Thesis/Thesis.pdf, Stand: 23.08.2000.

[CoPo95]
Cotter, Sean; Potel, Mike: Inside Taligent Technology, Reading Massachusetts 1995.

[CoSh95]
Connell, John; Shafer, Linda: Object-Oriented Rapid Prototyping, Englewood Cliffs, New Jersey 1995.

[CoYo94a]
Coad, Peter; Yourdon, Edward: Objektorientierte Analyse, München 1994.

[CoYo94b]
Coad, Peter; Yourdon, Edward: Objektorientiertes Design, München 1994.

[DaNy66]
Dahl, V.; Nygaard, K.: SIMULA - An ALGOL-based Simulation-Language, in: Communication of the ACM, Volume 9 Number 9, S.671-678 (1966).

[Date95]
Date, C.J.: An Introduction to Database Systems, 6ed. Reading, Massachusetts 1995.

[DeMu88]
Devlin, B.A.; Murphy P.T.: An architekture for a business and information system, in: IBM Systems Journal, Vol 27. No 1, 1988, S.60-80.

[Devl97]
Devlin, Barry: Data Warehouse: from architecture to implementation, Reading, Massachusetts 1997.

[Dunn93]
Dunn, Robert H.: Software-Qualität: Konzepte und Pläne, München 1993.

[EhEn00]
Ehrmann, Stephan; Engler, Tobias: Apples Sherlock um eigene Plugins erweitern, in: c't 2/2000, S.188-195.

[Espe91]
Espen, Andersen: On the Near Future of Business Computing, http://www.espen.com/near_fut.htm, Stand 7.1.1999.

[Fowl97]
Fowler, Martin: Analysis Patterns: Reusable Object Models, Menlo Park, California 1997.

[Gamm96]
Gamma, Erich: Java und Design Patterns - eine vielversprechende Kombination, in: Java Spektrum, September/Oktober 1996, S.18-24.

[GHJV95]
Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: Design Patterns: elements of reusable object-oriented software, Reading, Massachusetts 1995.

[GoAl92]
Goldstein, Neal; Alger, Jeff: Developing Object-Oriented Software for the Macintosh: Analysis, Design, and Programming, Reading, Massachusetts 1992.

[GoRo89]
Goldberg, Adele J.; Robson David: Smalltalk-80: The Language, Reading Massachusetts 1989.

[Grah93]
Graham, Ian: Object Oriented Methods, 2. Auflage, Wokingham 1993.

[Hale93]
Hales, C.: Managing Through Organization: The Management Process, Forms of Organisation and the Work of Managers, London, Routledge 1993.

[Hare88]
Harel, David: On Visual Formalisms, in: Communication of the ACM, Volume 31 Number 5, S.514-528.

[Hasl96]
Haslhofer, Gerald: Database Marketing, Wien 1996.

[HeHe95]
Herget, J.; Hensler, S.: Online-Datenbanken in Wirtschaft und Wissenschaft: aktuelle Nachfragestrukturen und Nutzungstendenzen, Wirtschaftsinformatik 37(1995) 2, S.129-138.

[Heue97]
Heuer, Andreas: Objektorientierte Datenbanken - Konzepte, Modelle, Standards und Systeme, 2. Auflage, Bonn 1997.

[Hewi98]
Hewitt, Carl: Developing Business Object-based Applications in JBuilder, http://www.oop.com/white_papers/java/business_objects.htm, Stand: 31.01.00.

[Holu99]
Holub, Alan: Building user interfaces for object-oriented systems, Part 1, http://www.javaworld.com/javaworld/jw-07-1999/jw-07-toolbox.html, 1999.

[Howa95]
Howard, Tim: Segregating application and domain: Part 1, in: The Smalltalk Report May 1995 S.12-14.

[HoWh96]
Holsapple, Clyde W.; Whinston, Andrew B.: Decision Support Systems: A Knowledge-based Approach, Mineapolis/St. Paul 1996.

[Hube97]
Hubert, Richard: Grundzüge eines Modells für die Konvergenz zweier Welten, in: Computerwoche 37/1997 S.65-66.

[HuHN86]
Hutchins, Edwin L.; Hollan ,James D.; Norman, Donald A.: Direct manipulation interfaces, in: Norman, Donald A.; Draper, S.W. [Hrsg.]: User Centered System Design, Hillsdale, NJ 1986, S.87-124.

[JCJÖ95]
Jacobson, Ivar; Christerson, Magnus; Jonsson, Patrik; Övergaard, Gunnar: Object-Oriented Software Engineering - A Use Case Driven Approach, Wokingham, 1992.

[Knut97]
Knuth, Donald Ervin: The art of computer programming, vol. 1, Fundamental Algorithms, 3rd ed., Reading, Massachusetts 1997.

[KrPo88]
Krasner, Glenn E.; Pope, Stephen T.: A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, in: JOOP August/September 1988, S.26-49.

[Kreu90]
Kreutzer, Wolfgang: Grundkonzepte und Werkzeugsysteme objektorientierter Systementwicklung, in: Wirtschaftsinformatik, 32. Jahrgang, Heft 3, Juni 1990, S.212-227.

[LeDa88]
Lengel, R.H.; Daft, R.L.: The selection of communication media as executive skill, in: Academy of Management Executive, 11 (3), S.225-232.

[Lehm99]
Lehmann, Peter: Definierte Fachbegriffe als kritischer Erfolgsfaktor, in: focus Beilage (1/99) zur Computerwoche vom 23.4.1999, S.24-25.

[Maur00]
v. Maur, Eitel: Object Warehouse - Konzeption der Basis objektorientierter Management Support Systems am Beispiel von Smalltalk und dem ERP Baan, Universität Osnabrück, 2000.

[Malo95]
Maloney, John: Morphic: The Self User Interface Framework, Mountain View, CA 1995, http://self.smli.com/release/Self-4.0/manuals/Self-4.0-UI-Framework.ps.Z, Stand 12.12.1996.

[McCo93]
McConnell, Steven C.: Code complete: a pratical handbook of software construction, Redmond, Washington 1993.

[Meye90]
Meyer, Bertrand: Objektorientierte Softwareentwicklung, München 1990.

[Mint91]
Mintzberg, Henry: Mintzberg über Management: Führung und Organisation, Mythos und Realität, Wiesbaden 1991.

[Mösc96]
Mösching, Andreas: Das Speichern von Objekten in Tabellen, in: Objektspektrum 5/1996, S.72-79.

[Niel93]
Nielsen, Jakob: Noncommand user interfaces, in: Communication of the ACM, Volume 36 Number 4, S.83-99.

[Nona91]
Nonaka, Ikujiro: The Knowledge-Creating Company, in: Harvard Business Review on Knowledge Management, Boston, MA, S.21-45.

[Norm87]
Norman, Donald A.: Some Observations on Mental Models, in: Readings in Baecker, Ronald M.; Buxton, William A.S. [ed.]: Human-Computer Interaction: A Multidisciplinary Approach, San Mateo, California 1987, S.241-244.

[Oppe95]
Oppelt, R. Ulrich G.: Computerunterstützung für das Management: neue Möglichkeiten der computerbasierten Informationsunterstützung oberster Führungskräfte auf dem Weg von MIS zu EIS?, München 1995.

[Parc95]
ParcPlace-Digitalk, Inc.: VisualWorks User's Guide, Sunnyvale, CA, 1995.

[Parn72]
Parnas, David L.: On the Criteria to Be Used in Decomposing Systems into Modules, in: Communication of the ACM, Volume 5 Number 12, S.1053-1058.

[PaSi94]
Pagel, Bernd-Uwe; Six, Hans-Werner: Software Engineering: Die Phasen der Softwareentwicklung, Bonn 1994.

[Pend98]
Pendse, Nigel: Storing multidimensional data,
http://www.olapreport.com/products/MDStorage.htm, Stand 28.12.1998.

[Pira96]
Piraino, Keith: A performance challenge, in: The Smalltalk Report, Volume 5 Number 6, S.4-11.

[PoBl93]
Pomberger, Gustav; Blaschek, Günther: Grundlagen der Software engineering: Prototyping und objektorientierte Software-Entwicklung, München 1993.

[Prac90]
Pracht, William E.: Model Visualization: Graphical Support for DSS Problem Structuring and Knowledge Organization, in: Decision Support Systems 6 (1990), S.13-27.

[RaWa95]
Rainer, Jr., R. Kelly; Watson, Hugh J.: What does it take for successful executive information systems?, in: Decision Support Systems 14 (1995) S.147-156.

[RBIM98]
Roberts, Dave; Berry, Richard; Isensee, Scott; Mullaly, John: Designing Software using OVID: Bridging User Interface Design and Software Engineering, Indianapolis, IN 1998.

[ReSu95]
Repenning, Alexander; Sumner, Tamara: Agenstsheets: A Medium for Creating Domain-Oriented Visual Languages, in: IEEE, March 1995 S.17-25.

[Rieg93]
Rieger, Bodo: Der rechnerunterstützte Arbeitsplatz für Führungskräfte, Habilitation, Technische Universität Berlin, 1993.

[Riel96]
Riel, Arthur J.:Object-Oriented Design Heuristics, Reading, Massachusetts 1996.

[RiKM00]
Rieger, B.; Kleber, A.; von Maur, E.: Metadata-Based Integration of Qualitative and Quantitative Information Resources Approaching Knowledge Management, in: Hansen, Hans Robert; Bichler, Martin; Mahrer, Harald [Hrsg.]: Proceedings of the 8th European Conference on Information Systems, ECIS 2000 - A Cyberspace Odyssey, Wien 2000, S.372-377.

[RiMK98]
Rieger, Bodo; v.Maur, Eitel; Krüger, Dietmar: Kritische Erfolgsfaktoren für Projektkooperationen zur Entwicklung objektorientierter Management Support Systeme - Ein Erfahrungsbericht, in: Gens, W. [Hrsg.]: Tagungsband der STJA '98, 4. Fachkongreß Smalltalk und Java in Industrie und Ausbildung, Erfurt 1998 S. 238-246.

[Riva96]
Rivard, Fred: Smalltalk: a Reflective Language, in: Kiczales, G. [Hrsg.]: Proceedings of Reflection'96, San Francisco, 1996, S.21-38.

[RoAl90]
Rosson, Mary Beth; Alpert, Sherman R.: The Cognitive Consequences of Object-Oriented Design, in: Human-Computer Interaction, 1990, Volume 5, S.345-379.

[Schä94]
Schäfer, Steffen: Objektorientierte Entwurfsmethoden: Verfahren zum objektorientierten Softwareentwurf im Überblick, Bonn 1994.

[SeBa97]
Seufert, A.; Back, A.: State of the Art des Management Supports - Teil 1: Klassische Management Support Ansätze, Bericht Nr.: IM HSG/IWI3/3, Universität St. Gallen für Wirtschafts-, Rechts- und Sozialwissenschaften, 4.3.1997.

[Silv91]
Silver, M.S.: Systems that Support Decision Makers: Description and Analysis, New York 1991.

[Shne83]
Shneiderman, Ben: Direct Manipulation: A Step Beyound Programming Languages, in: IEEE Computer August 1983, S. 57-69.

[SmMU95]
Smith,Randall B.; Maloney, Ungar, David: The Self-4.0 User Interface: Manifesting a System-wide Vision of Concreteness, Uniformity, and Flexibility, in: OOPSLA '95 Conference Proceedings, Austin, Texas, 1995, S. 47-60.

[Snyd86]
Snyder, Alan: Encapsulation and inheritance in object-oriented languages, in: OOPSLA '86 Conference Proceedings, Portland, OR, 1986, S.38-45.

[SpCa82]
Sprague Jr., Ralph; Carlson, Eric D.: Building effective Decision Support Systems, Englewood Cliffs, New Jersey 1982.

[StHa97]
Stahlknecht, Peter; Hasenkamp, Ulrich: Einführung in die Wirtschaftsinformatik, 8. Auflage 1997.

[Tayl92]
Taylor, David A.: Object-Oriented Information Systems: Planning and Implementation, New York 1992.

[Tayl95]
Taylor, David A.: Business Engineering with Object Technology, New York 1995.

[Tesl81]
Tesler, Larry: The Smalltalk Environment, in: Byte 8/1981, S.90-147.

[Togn91]
Tognazzini, Bruce: Tog on Interface, Reading, Massachusetts 1995.

[Uhli88]
Uhlir, S.: Enabling the user interface, in: IBM Systems Journal, Vol 27, No 3, 1988, S.306-314.

[Vets95]
Vetschera, Rudolf: Informationssysteme der Unternehmensführung, Berlin 1995.

[Vett95]
Vetter, Max: Objektmodellierung: eine Einführung in die objektorientierte Analyse und das objektorientierte Design, Stuttgart 1995.

[WaHR97]
Watson, Hugh J.; Houdeshel, George; Rainer Jr., Rex Kelly: Building Executive Information Systems and other Decision Support Systems, New York 1997.

[WiGr88]
Witte, Th.; Grzybowski, R.: Objectbased Simulation: Foundations and Implementation in Modula 2, in: Huntsinger, R.C., et.al. [Hrsg.]: Proceedings of the European Simulation Multiconference, Nice 1988, S.193-198.

[Will87]
Williams, G.: HyperCard, in: Byte 12/1987, S.109-111.

[WiWW90]
Wirfs-Brock, Rebecca; Wilkerson, Brian; Wiener, Laura: Designing Object-Oriented Software, Englewood Cliffs, NJ 1990.

[WoSm95]
Wolczko, Mario; Smith, Randall B.: Prototype-Based Application Construction Using SELF 4.0, Sun Microsystem Laboraties, SMLI Document 95-0257, 1995.

Anhang

Abbildung A.1: Beispiel eines Visitor-Pattern

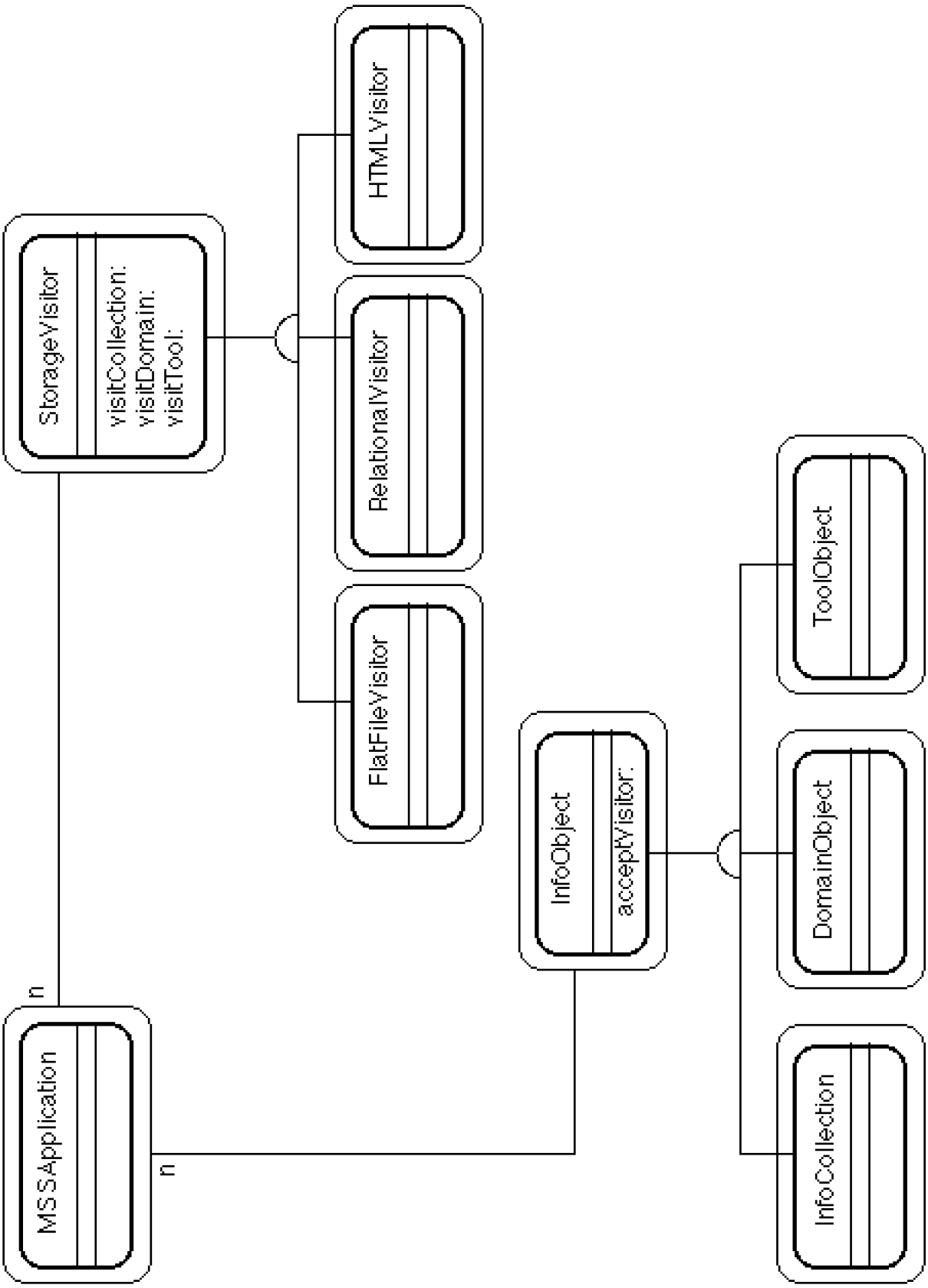

Abbildung A.2: Modell der Werkzeugobjekte