Zusammenfassung

Die Ingenieurswissenschaften verfügen über Konstruktionshandbücher, welche es ihren Anwendern ermöglichen, auf erprobte Lösungen für wiederkehrende Probleme zurückzugreifen. In der objektorientierten Softwareentwicklung wird seit einiger Zeit ebenfalls versucht, das Wissen erfahrener Entwickler in geeigneter Form festzuhalten. Dies geschieht mit Hilfe von „Mustern“, welche als Vorlage zur Lösung immer wiederkehrender Probleme dienen. Muster werden zur Konstruktion und Dokumentation von Softwaresystemen verwendet und erleichtern die Kommunikation unter Entwicklern. Diese Arbeit diskutiert die bisherige Literatur zum Thema Muster und definiert auf eigenen Erfahrungen aufbauend einen softwaretechnisch orientierten Musterbegriff. Es wird herausgearbeitet, wie Muster Erfahrungswissen effizient darstellen können. Die Struktur von Musterabhängigkeiten wird diskutiert und ein Ordnungsschema vorgestellt, welches im Rahmen eines Handbuchs eingesetzt werden kann. Die Definitionen werden an einem großen Beispiel, dem Softwareentwurf nach der Werkzeug und Material Metapher, erprobt. Dabei werden die Metaphern und die sie implementierenden Konzepte zur Werkzeugkonstruktion und -integration als eine Menge aufeinander abgestimmter Muster präsentiert. An einem die Musterdarstellung begleitenden Beispiel wird gezeigt, wie Muster Softwaresysteme und Softwarearchitekturen erklären und dokumentieren können.

Dies ist die leicht revidierte Fassung einer Arbeit, welche dem Fachbereich Informatik der Universität Hamburg als Diplomarbeit eingereicht wurde. Die Arbeit wurde von Prof. Dr. Heinz Züllighoven (Universität Hamburg) und Dr. Walter Bischofberger (UBILAB) betreut. Sie ist im Rahmen der SWT-Schriftenreihe als Mitteilung des Fachbereichs Informatik der Universität Hamburg und als UBILAB Technical Report erschienen.

Dirk Riehle

Zürich, im Juni 1995.

Inhaltsverzeichnis

1 Überblick über die Arbeit

Muster können als Vorlage für die Konstruktion und Dokumentation von Software dienen und die Kommunikation unter Entwicklern erleichtern. Diese Arbeit diskutiert bestehende Literatur zum Thema Muster und definiert einen softwaretechnisch orientierten Musterbegriff. Das Konzept des Musters wird am Beispiel der Werkzeug und Material Metapher erprobt, einem Ansatz zur Entwicklung interaktiver Softwaresysteme. Dieses Kapitel schildert die Motivation für die Arbeit, ihren Hintergrund und Ausgangspunkte. Die Arbeit wird im Überblick präsentiert.

1.1 Einleitung und Motivation

Die Komplexität moderner Softwaresysteme steigt beständig. Neue Anforderungen wie graphische Benutzungsschnittstellen und Client/Server Systeme stellen viele Softwareentwickler vor Aufgaben, welche außerhalb ihres bisherigen Erfahrungshorizonts liegen. Gefordert sind gleichermaßen neue Konzepte, diese Aufgaben zu bewältigen, als auch Mittel und Wege, die Konzepte effizient zu kommunizieren und anzuwenden.

Seit einiger Zeit wird versucht, diesen Problemen mit dem Einsatz sogenannter Entwurfsmuster („design patterns“) zu begegnen. Ein Entwurfsmuster entsteht in der Erfahrung von Softwareentwicklern. Es stellt die Lösung für ein immer wiederkehrendes Problem dar. Entwurfsmuster können dann als Vorlage zur Lösung dieser Probleme verwendet werden. Sie erleichtern die Konstruktion und Dokumentation von Softwaresystemen und die Kommunikation unter Entwicklern.

An der Universität Hamburg beschäftigt sich der Arbeitsbereich Softwaretechnik des Fachbereichs Informatik unter anderem mit der Entwicklung interaktiver Softwaresysteme. Ein wichtiger Ansatz ist die Softwareentwicklung nach der Werkzeug und Material Metapher. Sie basiert auf sogenannten Entwurfsmetaphern, welcher der benutzungsorientierten Gestaltung von Softwaresystemen dienen, sowie Implementationskonzepten, welche die Metapher in konkrete Software umsetzen helfen.

Entwurfsmuster sind den Entwurfsmetaphern und Implementationskonzepten sehr ähnlich. Es entstand der Entschluß, den Begriff des Musters genauer zu analysieren und für die Softwaretechnik fruchtbar zu machen. Erster großer Prüfstein sollte die Reinterpretation der Werkzeug und Material Metapher als eine Menge aufeinander abgestimmter Muster sein. Das Ergebnis des Unternehmens liegt mit dieser Arbeit vor.

Während der Arbeit zeigte sich schnell, daß der Begriff Entwurfsmuster sinnvoll verallgemeinert werden kann. Dies wurde auch schon von anderen Autoren getan. In dieser Arbeit wird aber nicht nur ein allgemeiner softwaretechnisch orientierter Musterbegriff definiert, sondern es werden auch drei verschiedene Spezialisierungen betrachtet: Interpretations- und Gestaltungsmuster, Entwurfsmuster und Programmiermuster. Jeder dieser drei Musterarten kommt eine unterschiedliche Bedeutung für die Entwicklung von Softwaresystemen zu.

Weiterhin wurde klar, daß die Arbeit sich nicht nur mit einzelnen Mustern zu befassen hatte, sondern mit einer Menge zusammenhängender Muster. Somit mußte die Struktur von Musterzusammenhängen analysiert und ein Ordnungsschema entwickelt werden. Darauf aufbauend wurden schließlich die Entwurfsmetaphern und Implementationskonzepte der Werkzeug und Material Metapher reinterpretiert und als eine Menge aufeinander Bezug nehmender Muster herausgearbeitet.

Der empirische Hintergrund der geschilderten Muster ist ein am Arbeitsbereich Softwaretechnik entwickelter objektorientierter Anwendungsrahmen für die Werkzeug und Material Metapher. Er stellt die Basis für viele Studien- und Diplomarbeiten dar. Eine wichtige Anwendung ist SANE, ein System zur Unterstützung aufgabenorientierter Anforderungsermittlung [Kei89], welches intensiv in Lehre und Forschung eingesetzt wird [Rie93, Lil95, DL95].

Das Konzept des Musters ist inzwischen einer breiten Öffentlichkeit bekannt geworden und hat viel Interesse auf sich gezogen. Die Anzahl der Veröffentlichungen zum Thema steigt beständig. Es gibt bereits Kolumnen in Zeitschriften, welche sich vorrangig dem Thema Muster widmen. Auf den großen Konferenzen für objektorientierte Softwareentwicklung (OOPSLA, ECOOP) sind „Patterns“ als eigenständiges Thema anerkannt. Seit 1994 gibt sogar die alljährlich stattfindende „Pattern Languages of Programming“ Konferenz (PLoP-94, [CS95]), auf der auch Teile dieser Arbeit bereits vorgetragen wurden [RZ95].

Viele Wissenschaftler hoffen, daß sich Muster als ein effizientes Mittel herausstellen werden, das Wissen erfahrener Entwickler festzuhalten und zu kommunizieren. Gestaltungs-, Architektur- und Entwurfshandbücher für die Softwareentwicklung können dann auf Muster als zentrale Bausteine zurückgreifen. Der Autor hofft, mit dieser Arbeit einen Beitrag zum Verständnis des Musterbegriffs und der Struktur von Musterzusammenhängen geleistet zu haben, welche den genannten Zielen in der Softwareentwicklung dienen kann.

1.2 Ausgangspunkte und Überblick über die Arbeit

Die vorliegende Arbeit besitzt zwei voneinander zu unterscheidende Ausgangspunkte: die Diskussion um Entwurfsmuster und Mustersprachen [Coa92, Joh92, GHJ+93] und den Softwareentwurf nach der Werkzeug und Material Metapher [BZ92, BCS92, GZ92].

Die Diskussion um Entwurfsmuster und Mustersprachen begann Ende der achtziger Jahre auf den OOPSLA Workshops über Muster und Softwarearchitektur, geleitet von Peter Coad und Bruce Anderson respektive. Sie führte zu einer Ausarbeitung des Musterbegriffs in [Coa92, GHJ+93]. Ein wichtiger Beitrag wurde zuletzt von Gamma et al. mit [GHJ+95] gesetzt, welche aufgrund ihres großen Erfolgs den Musterbegriff einer breiteren Öffentlichkeit zugänglich machten. Andere Arbeiten über Softwarearchitektur, welche sicherlich indirekt Einfluß auf diese Arbeit genommen haben, aber nicht weiter explizit verfolgt werden, stammen von Garlan et al. [AG92, GAO94] und Shaw et al. [SDK+95, GS93].

Die Entwicklung der Werkzeug und Material Metapher wurde von Reinhard Budde, Karl-Heinz Sylla und Heinz Züllighoven am Projektbereich WOK der GMD Ende der achtziger Jahre vorangetragen [BZ92, BCS92]. Inzwischen ist sie in eine breite Diskussion verschiedener aber zusammenhängender Gruppen gemündet. Sie wird hauptsächlich vom Arbeitsbereich Softwaretechnik des Fachbereichs Informatik der Universität Hamburg, Wissenschaftlern des Projektbereichs Hardware/Software Codesign der GMD und dem Projekt GeBOS der RWG/Stuttgart getragen.

Kapitel 2 erläutert die Werkzeug und Material Metapher und fügt sie in ein Schema der Softwareentwicklungsmethoden ein, welches in Hamburg maßgeblich von Christiane Floyd und Heinz Züllighoven entwickelt wurde. Die wesentlichen Differenzierungskriterien sind Vorgehensweise, Leitbild, Entwurfsmetaphern und Konstruktionstechniken.

Kapitel 3 stellt den Musterbegriff dieser Arbeit vor. Ausgehend von der Analyse bisheriger Arbeiten wird ein allgemeiner Musterbegriff definiert und in die Bereiche Interpretations-, Design- und Programmiermuster unterteilt. Die Struktur von Musterzusammenhängen wird betrachtet und ein Ordnungsschema vorgestellt. Die Definitionen legen den Grundstein für die Präsentation der Muster für die Werkzeug und Material Metapher in den nächsten Kapiteln.

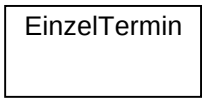
Kapitel 4 bis 6 beschreiben Muster zur Werkzeug und Material Metapher. Kapitel 4 erläutert die dem Ansatz zugrundeliegenden Interpretations- und Gestaltungsmuster. Die Darstellung basiert auf dem im vorangehenden Kapitel definierten Problem-Kontext-Lösung Schema. Kapitel 5 stellt die objektorientierten Entwurfsmuster zur Konstruktion von Softwarewerkzeugen vor. Als konkrete softwaretechnische Entwurfsmuster lösen sie Entwurfsprobleme, wie sie bei der Konstruktion von Systemen nach der Werkzeug und Material Metapher auftreten. Kapitel 6 führt weitere objektorientierte Entwurfsmuster zur Integration der zuvor entworfenen Softwarewerkzeuge an einem einzelnen Arbeitsplatz ein.

Das letzte Kapitel erprobt die Muster an einem Beispiel und diskutiert ihre Anwendbarkeit. Die Arbeit wird kritisch reflektiert und zusammengefaßt und es wird ein Ausblick auf zukünftige Entwicklungen gegeben.

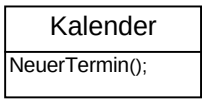
1.3 Sprachliche und graphische Konventionen

Die verwendeten Begriffe für objektorientierte Konzepte entstammen der Welt statisch typisierter Programmiersprachen, ohne dabei sprachspezifisch zu werden. Gesprochen wird von Ober- und Unterklassen, von Vererbung, von Operationen und Funktionen, von Attributen, Polymorphie und spätem Binden.

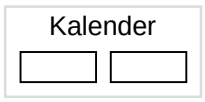
Die verwendete graphische Notation für objektorientierte Entwurfsdiagramme basiert auf der OMT-Notation [RBP+91, Rum95] und ist im folgenden dargestellt:



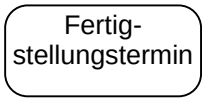
Ein Rechteck repräsentiert eine Klasse. Ist der Schriftzug kursiv gesetzt, so ist die Klasse abstrakt.



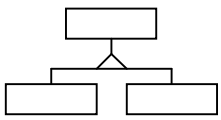
Operationen einer Klasse werden unterhalb einer Trennlinie, welche sie vom Klassennamen abtrennt, aufgeführt.



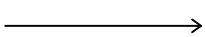
Klassen können logisch zu einer Gruppe von Klassen zusammengefaßt werden. Eine Gruppe wird durch einen dicken grauen Rahmen dargestellt.



Objekte werden durch Rechtecke mit abgerundeten Ecken dargestellt.



Die Vererbungsbeziehung ist durch gerade Striche dargestellt, welche die Klassen verbinden. Das Dreieck zeigt mit seiner Spitze auf die Oberklasse.



Die Benutzungsbeziehung wird durch einen einfachen Pfeil dargestellt. Ein offener Pfeilkopf besitzt die Multiplizität 1.

Ein ausgefüllter Pfeilkopf besitzt die Multiplizität n (beliebig viele oder auch keines). Links ist eine n zu 1 Beziehung dargestellt.

Wird in dieser Arbeit von „dem Benutzer“ gesprochen, so ist damit gleichermaßen der männliche wie auch der weibliche Benutzer, d.h. die Benutzerin, gemeint. Der Autor sieht hinreichend viele Unterschiede zwischen dem grammatischen Geschlecht und der Vielfalt lebender Menschen, um diese Wahl als gerechtfertigt zu erachten. Er hofft trotzdem, eines Tages von einer alle Seiten befriedigenden Lösung zu hören. Er wird sie dankbar verwenden.

2 Die Werkzeug und Material Metapher

Die Softwareentwicklung nach der Werkzeug und Material Metapher basiert auf allgemeinverständlichen Metaphern, welche zur Gestaltung interaktiver Softwaresysteme verwendet werden. Das hinter ihr stehende Leitbild ist der softwaretechnisch zu erweiternde Arbeitsplatz für qualifizierte menschliche Arbeitstätigkeit. Metaphern wie Werkzeug und Material wirken sprachbildend für Entwickler und Benutzer. Die Metaphern werden mit Hilfe von Konstruktionstechniken in einen softwaretechnischen Entwurf umgesetzt. Dieses Kapitel schildert die Einbettung der Metaphern in den Gesamtzusammenhang der Softwareentwicklung nach der Werkzeug und Material Metapher. Es werden die vier Ebenen Vorgehensmodell, Leitbild, Entwurfsmetaphern und Konstruktionstechnik erläutert.

2.1 Ein Modell für einen methodischen Rahmen

Die Entwicklung von Software nach der Werkzeug und Material Metapher ist eingebettet in einen methodischen Rahmen, der sich durch die folgenden vier Dimensionen beschreiben läßt:

- Die *Vorgehensweise* beschreibt das verwendete Vorgehensmodell. Beispiele sind das bekannte Phasenmodell [Roy70], das Spiralmodell [Boe88] oder evolutionäres Prototyping [Flo84, BKK+92, BP92].
- Das *Leitbild* des Entwicklungsprozesses erläutert das der Softwareentwicklung zugrundeliegende Menschenbild. Es fragt nach dem Verhältnis der zukünftigen Benutzer zum zu entwickelnden Softwaresystem. Es klärt Grundannahmen über Qualifikation und Verantwortlichkeit der Benutzenden [Cle94].
- Die *Entwurfsmetaphern* setzen die Annahmen des Leitbildes in konkrete sprachbildende Metaphern um. Die Metaphern dienen den Entwicklern als Analysemittel, in dem sie helfen, den Anwendungsbereichs zu interpretieren und gedanklich vorzustrukturieren. Sie sind weiterhin ein Entwurfsmittel, in dem sie den Entwicklern helfen, das zu entwickelnde System gedanklich vorwegzunehmen. Beispiele für Metaphern sind die System- und Maschinenmetapher, die Kommunikationsmetapher oder eben die Metaphern Werkzeug und Material (für die verschiedenen Metaphern siehe [MO92]).
- Im Laufe des Softwareentwicklungsprozesses werden verschiedene Dokumente erstellt. Eine *Konstruktionstechnik* beschreibt Aufbau und Techniken der Erstellung und Weiterführung eines Dokuments. Dokumente sind z.B. eine Anforderungsspezifikation, Systemvision oder Entwurfsdiagramme und Quelltexte.

Die Auswirkungen unterschiedlicher Leitbilder sind mannigfaltig und müssen explizit diskutiert werden. Das Leitbild, Software für wenig qualifizierte Benutzer zu entwickeln, wird dazu führen, Arbeitsabläufe fest vorzugeben und fachlich inkonsistente Systemzustände nicht zuzulassen. Das Resultat ist ein restriktives System.

2.2 Einordnung der Werkzeug und Material Metapher

In diesem Abschnitt wird der Softwareentwicklungsansatz „Die Werkzeug und Material Metapher“ beschrieben. Wichtiger Bestandteil sind die Entwurfsmetaphern Werkzeug und Material. Der Ansatz läßt sich wie folgt in den erläuterten methodischen Rahmen einordnen.

2.2.1 Evolutionäres Prototyping

Die gewählte Vorgehensweise der Softwareentwicklung basiert auf Prototyping [BKK+92, BP92] und ist evolutionär und partizipativ. Die Benutzenden des Systems werden als Experten des Fachgebiets erachtet und ihnen wird entsprechender Raum zur Partizipation eingeräumt.

Im Rahmen von evolutionärem Prototyping wird Softwareentwicklung als ein Prozeß gegenseitigen Lernens und gegenseitiger Rückkopplung organisiert, in welchen Entwickelnde und Benutzende gleichermaßen eingebunden sind.

Gemeinsame Sprachbildung sowie Rückkopplung zwischen Entwicklern und Anwendern findet in der Diskussion um Prototypen statt, welche von den Entwicklern vorgestellt und mit Benutzern diskutiert werden. Aus der Diskussion und unter Verwendung verschiedener Techniken wie Szenarios, Systemvisionen und dem Aufbau von Glossaren entwickelt sich ein fachliches Verständnis der Entwickler für die Arbeit der Benutzer. Gleichzeitig entwickeln die (zukünftigen) Benutzenden ein Verständnis der möglichen und tatsächlichen Leistungsfähigkeit des zu erstellenden Softwaresystems.

Evolutionäres Prototyping nach [KGZ93] verwendet zur Steuerung des Projektverlaufs und Kontrolle erreichter Ergebnisse Projektstadien und Referenzlinien. Projektstadien beschreiben terminlich gebundene Zustände des Projekts, welche zu erreichende Merkmale von Dokumenten nennen. Die Terminfestlegung kann innerhalb des Projekts (Fortschrittskontrolle, Revisionen, Reviews) oder außerhalb des Projekts (Vertragsunterschriften, Messen, Vorstandssitzungen) begründet sein. Referenzlinien beschreiben Zustände von Dokumenten. Sie basieren zumeist auf Qualitätsmerkmalen, die es zu erreichen gilt. Somit können Referenzlinien zur Beschreibung von Projektstadien verwendet werden.

Projektstadien wie auch Referenzlinien können situativ entwickelt und fortgeschrieben werden. Es besteht keine Notwendigkeit, zu Beginn eines Projekts Meilensteine und zugehörige Dokumente zu fixieren.

2.2.2 Das Leitbild des Arbeitsplatzes für qualifizierte menschliche Arbeit

Nach der Werkzeug und Material Metapher entwickelte Software basiert auf der Grundannahme, daß die zukünftigen Benutzenden des Softwaresystems in ihrer Tätigkeit ausreichend qualifiziert sind, um die angebotenen Freiheitsgrade sinnvoll und verantwortlich nutzen zu können. Der Arbeitsplatz wird um eine Softwareumgebung erweitert und zum Teil auch in ihr modelliert. Zielsetzung ist, qualifizierte menschliche Arbeitstätigkeiten zu unterstützen. Das zugrundeliegende Leitbild macht folgende Annahmen:

- Die Benutzer besitzen die notwendige Kompetenz und Fähigkeit, ihre Arbeit selbständig zu erledigen.
- Es gibt keinen Grund, einen vordefinierten Arbeitsablauf einzuführen, da die Steuerung der Vorgänge durch die Benutzenden selbst flexibel gestaltet werden kann und soll.
- Die zu erledigenden anspruchsvollen Aufgaben verlangen individuell einstellbare und organisierbare (Software-)Umgebungen.

Das Leitbild und der Hintergrund der Werkzeug und Material Metapher, vorgestellt in Kapitel 4, erläutert diese Annahmen im Detail. Aus diesen Grundannahmen kann die Anwendungsdomäne abgeleitet werden, in welcher die Werkzeug und Material Metapher zum tragen kommen kann. Dies sind z.B. büro- und werkstattartige Umgebungen, in denen der Fokus auf den Tätigkeiten des einzelnen qualifizierten Mitarbeiters ruht.

Dieses Leitbild richtet sich gegen taylorisierte Arbeitsumgebungen, in denen die Benutzer, insbesondere durch den Einsatz von EDV in Arbeitsabläufe „eingespannt“ und zu „Bedienern“ der Maschine gemacht werden. Der Einsatz nach dem Leitbild entwickelter Software ermöglicht es Anwendern, ihre Qualifikation in Bereiche zu tragen, welche mangels adäquater Werkzeugunterstützung bisher nicht erreichbar waren. Einerseits werden die Anwender von lästigen Routinearbeiten befreit, andererseits werden ihre Arbeitsmöglichkeiten durch den Einsatz von Software drastisch erweitert.

Das genannte Leitbild reiht sich ein in die Diskussion um mehr Verantwortlichkeit, Flexibilisierung und Dezentralisierung am Arbeitsplatz wie er unter den Stichworten „Virtuelle Fabrik“, „Fraktale Fabrik“, „Lean Management“ usw. geführt wird [War92, Woh94, Cle94].

2.2.3 Die Entwurfsmetaphern Werkzeug und Material

Die Gestaltung des Systems basiert auf den Entwurfsmetaphern Werkzeug und Material. Für diese Metaphern ist üblicherweise bereits ein intuitives Verständnis auf Seiten der Benutzer wie Entwickler vorhanden ist. Dadurch wird der Einstieg in eine gemeinsame Sprachebene erleichtert. Sie bilden einen fruchtbaren Ausgangspunkt für die Diskussionen zur Entwicklung eines gemeinsamen fachlichen Verständnisses des Anwendungsbereichs.

Die beiden Metaphern gaben dem vorgestellten Ansatz ihren Namen. Es gibt allerdings noch weitere Metaphern, so z.B. die Metaphern Automat und Umgebung, welche die erstgenannten Metaphern ergänzen. Neue Anwendungsbereiche werden möglicherweise neue oder erweiterte Metaphern notwendig machen.

Werkzeuge sind Arbeitsmittel, welche zur Sondierung und Manipulation der Arbeitsgegenstände, der Materialien, dienen. Diese Begriffe werden im übernächsten Kapitel als Muster reinterpretiert und detailliert beschrieben werden. Hier soll nur festgehalten werden, daß durch den Einsatz der Metaphern Entwicklern wie Benutzern ein Mittel an die Hand gegeben wird, welches ihnen hilft, konkrete Vorstellungen des zu entwickelnden Systems zu entwerfen und mit Hilfe der angebotenen Techniken auch zu realisieren.

Die Metaphern Werkzeug und Material werden durchgängig eingesetzt. Ihr Einsatz durchzieht alle Sprachebenen der Diskussion um das System. Sie helfen, die notwendige Brücke zwischen sozialen Anforderungen und softwaretechnischem Entwurf zu bauen. Sie ermöglichen es, Handhabung und Präsentation sowie Funktionalität des Systems gedanklich vorwegzunehmen. Und sie helfen den Entwickelnden bei der Interpretation der Realität der Anwendungswelt, für welche die Werkzeug und Material Metapher entwickelt wurde.

2.2.4 Konstruktionstechniken

Im Laufe eines Softwareprojektes sind viele Dokumente zu erstellen. Die wichtigsten sind Szenarios, Systemvisionen, Glossare, Entwurfsdiagramme und Quelltexte. Im Rahmen des hier vorgestellten Ansatzes basieren Entwurf und Quelltexte auf dem objektorientierten Programmierparadigma. Einen Überblick über die verschiedenen Dokumente gibt [KGZ93]. Für jeden Dokumenttyp gibt es Konstruktionstechniken, welche praktische Hilfe, Erfahrungswissen oder auch dezidierte Anleitung geben, wie das Dokument zu erstellen ist.

Im folgenden sind nur drei verschiedene Dokumente von Interesse: Szenarios, Visionen und das Entwurfsdokument. Ein Szenario schildert einen alltäglichen Arbeitsvorgang ohne den Einsatz der neu zu entwickelnden Software. Es beschreibt also einen Teil des Ist-Zustands. Szenarios werden von Entwicklern geschrieben. Sie werden von Benutzern überprüft und bilden somit eine Grundlage für Diskussionen und Entwicklung eines gemeinsamen Verständnisses. Anhand von Szenarios werden die Begriffe der Fachsprache herausgearbeitet, welche einen wesentlichen Bestandteil der objektorientierten Modellierung bilden.

Eine Systemvision schildert einen Aspekt des zukünftigen Softwaresystems durch Antizipation des zu entwerfenden Systems. Sie wird von Entwicklern geschrieben, die sich vorstellen, daß das geschilderte System real ist. Systemvisionen dienen der Diskussion unter Entwicklern. Es gibt mehrere Arten von Systemvisionen, welche unterschiedliche Ziele verfolgen. Eine spezielle Systemvision, die Handhabungsvision, schildert den antizipierten Umgang eines Benutzers mit dem System.

Das Entwurfsdokument beschreibt die Softwarearchitektur des zukünftigen Systems. Seine Komplexität ist groß und es ist somit besonders schwer zu erstellen. In iterativen und an Prototyping orientierten Vorgehensmodellen wird es nie an einem Stück entwickelt, sondern immer wieder verbessert, revidiert und fortgeschrieben. Es schildert die im Softwaresystem verwendeten Subsysteme, Module, Klassen und Objekte sowie ihre Beziehungen untereinander. Zu seiner Realisierung gibt es verschiedene sogenannte Implementationskonzepte: Softwarewerkzeuge werden in Interaktions- und Funktionskomponenten unterteilt, Funktionskomponenten greifen auf Materialobjekte nur über Aspektklassen zu usw.

Die Implementationskonzepte liegen explizit vor, sind aber nur auf Ebene der einzelnen Klassen und Objekte benennbar. Teil des Anspruchs dieser Arbeit ist es, mit Hilfe der Muster für die Werkzeug und Material Metapher eine Ebene einzuziehen, welche die Brücke zwischen Systemvisionen und dem sie realisierenden Entwurf bildet. Dazu wurden die Implementationskonzepte analysiert, verändert und erweitert und dann in Muster überführt. Muster sind größere Einheiten als Klassen und Objekte und machen die resultierende Softwarearchitektur durchschaubarer und verständlicher.

2.3 Muster für die Werkzeug und Material Metapher

Wie im vorigen Abschnitt geschildert, sollen die in den nächsten Kapiteln eingeführten Muster für die Werkzeug und Material Metapher nicht nur Konzepte neu und besser darstellen, sondern auch die Lücke zwischen Systemvisionen und dem eigentlichen Entwurfsdokument schließen helfen. Der Sprung, den die Entwickler machen müssen, ist weit: Ausgehend von Systemvisionen, welche verschiedene Aspekte des antizipierten Systems schildern, soll der Weg zu einem adäquaten objektorientierten Entwurf gefunden werden.

Muster sollen diesen Prozeß unterstützen helfen. Muster sind Vorlagen, welche in großem Umfang und auf zum Teil ganz unterschiedlichen Ebenen eingesetzt werden können. Sie dienen gleichermaßen als Vorlage für die Gestaltung von Softwaresystemen als auch als Vorlagen für die Erstellung eines detaillierten objektorientierten Entwurf und seiner Implementation. Sie sind in der Lage, sowohl die der Werkzeug und Material Metapher zugrunde liegenden Metaphern als auch die Implementationskonzepte zu beschreiben. Dadurch, daß Muster einen so weiten Bereich überspannen, bauen sie eine Brücke und begleiten Softwareentwickler von den Systemvisionen bis zum detaillierten objektorientierten Entwurf.

Um ein Verständnis für die Mächtigkeit des Einsatzes von Mustern für den Softwareentwurf zu gewinnen, wird an dieser Stelle auf ein Beispiel vorgegriffen, mit dessen Hilfe die Muster später selbst erläutert werden: der Entwurf eines Zeitplanungssystems. Hier wird zuerst nur ein einfacher Kalender vorgestellt (siehe Abb. 2-1).

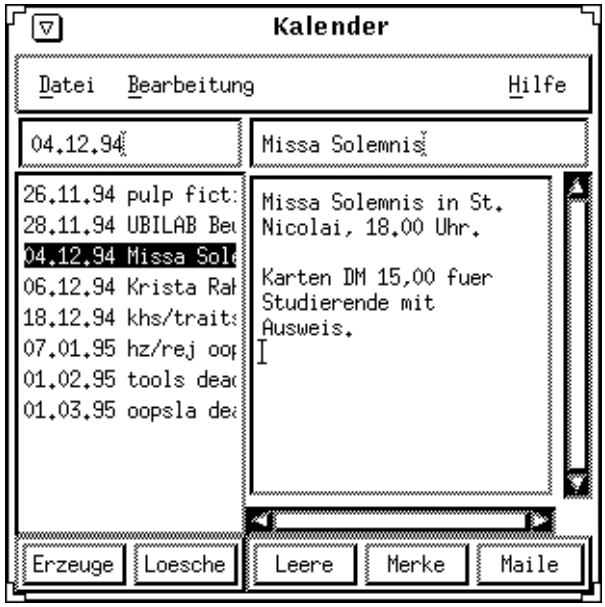


Abb. 2-1: Die Benutzungsoberfläche des antizipierten Kalenders.

Die nächsten Unterabschnitte schildern ein Szenario sowie eine Handhabungsvision für das Kalenderbeispiel. Ein Szenario schildert einen Arbeitsablauf, wie der Anwender ihn mit bisherigen Mitteln ausführen würde. Eine Handhabungsvision schildert einen Arbeitsablauf, wie er mit dem zu entwickelnden System in Zukunft geschehen könnte.

Die Handhabungsvision wird in der Diskussion der Anwendung von Mustern in Kapitel 7 wieder aufgegriffen und neu geschrieben werden. Es wird von dem Entwurfsdokument gefolgt werden, anhand dessen der Mustereinsatz und seine Fruchtbarkeit untersucht werden wird.

2.3.1 Szenario: Arbeiten mit einem Taschenkalender

Der Benutzer möchte einen *Termin* für ein Weihnachtskonzert aufschreiben. Der dazu benötigte *Kalender* befindet sich auf seinem *Schreibtisch*. Nachdem er ihn unter einem *Stapel von Unterlagen* gesucht und gefunden hat, *blättert* er zu jener *Seite* vor, welche dem zu notierenden *Datum* entspricht. Es ist der 4. Dezember 1994. Die ersten Seiten überblättert er ihrer *Abnutzung* wegen im *Überschlag*, die letzten Seiten blättert er Seite für Seite vorwärts, bis er jene *Woche* gefunden hat, in welcher der Termin liegt.

Der für einen ganzen *Tag* freigelassene Platz ist nicht sehr umfangreich. Maximal drei *Verabredungen* mit etwas zusätzlichem Text lassen sich auf ihm unterbringen. Der 4. Dezember ist noch frei, es ist ein Sonntag. Der Benutzer greift zu einem *Stift* und schreibt die wichtigsten *Daten* nieder: *Uhrzeit*, *Ort* und den Namen des Stücks sowie einen *Vermerk*, daß er für verbilligte Karten seinen Studentenausweis nicht vergessen soll. Nachdem er das getan hat, legt er den Kalender wieder ab, der sofort von allein *zufällt*.

2.3.2 Handhabungsvision: Arbeiten mit einem Kalenderwerkzeug

Der Benutzer will einen neuen Termin eingeben. Er sucht das Kalenderwerkzeug im Zeitplanungsordner, wo er auch seine verschiedenen Terminbücher weiß. Er klickt auf das Terminbuch für private Verabredungen, da der einzugebende Termin ein Privattermin ist.

Da ihm das Datum bekannt ist, tippt er es direkt in das Eingabefeld für das Datum links oben unterhalb der Menüzeile (siehe Abb. 2-1). Mittels Tabulator springt er weiter in die Eingabezeile für den Titel, der für einen Termin vergeben werden kann. Nach Eingabe des Titels greift er zur Maus und setzt den Eingabecursor in das Editierfeld auf der rechten Seite des Kalenders.

Mit dem Klick in das Editierfeld gilt der Termin als zu übernehmen. Er erscheint links in der Liste für die Termine als neuer Termin. Der angezeigte Listeneintrag besteht aus dem Datum

und Titel des Termins. Er ist chronologisch einsortiert. Der sichtbare Ausschnitt der Liste wird automatisch auf den richtigen Zeitbereich gesetzt, so daß der Benutzer sehen kann, welche Termine zeitlich in der Nähe liegen. Für das angegebene Datum existiert aber noch kein weiterer Termin.

Der Benutzer tippt seine Anmerkungen in das Editierfeld und kommentiert somit den Termin. Weitere Schritte unternimmt er nicht. Er klickt zweimal auf den Schließenknopf des Fensters. Die Rückfrage des Editors, ob die neu eingegebenen Termine gesichert werden sollen, bejaht er. Beim Beenden des Kalenders wird der neu eingegebene Termin gesichert.

3 Zum Musterbegriff

In diesem Kapitel wird der für diese Arbeit grundlegende Begriffs des Musters definiert. Die Abschnitte 3.1 bis 3.3 diskutieren bisherige Arbeiten zu diesem Thema. In Abschnitt 3.4 wird ein allgemeiner Musterbegriff sowie drei softwaretechnische Ausprägungen (Interpretations- und Gestaltungsmuster, Entwurfsmuster, Programmiermuster) definiert und diskutiert. Die Definitionen können unabhängig vom Rest des Kapitels gelesen werden. In den Abschnitten 3.5 und 3.6 wird diskutiert, was für Abhängigkeiten es zwischen Mustern geben kann und was für Konsequenzen sich für die Anordnung von Mustern ergeben.

3.1 Der allgemeine und disziplinäre Begriff des Musters

In diesem Abschnitt wird eine dem Brockhaus entnommene allgemeine Definition des Musterbegriffs vorgestellt und diskutiert. Es schließen sich disziplinäre Definitionen aus den Sozial- und Sprachwissenschaften an. Ziel der Rezeption ist nicht der umfassende Überblick, sondern das Abstecken des Diskursbereichs für eine mögliche Musterdefinition. Die gewonnenen Dimensionen werden zusammengefaßt.

3.1.1 Allgemeine Definition von „Muster“

Die Brockhaus Enzyklopädie definiert den Begriff „Muster“ folgendermaßen:

> **Muster [...]**
>
> allgemein: 1) Vorlage, nach der etwas hergestellt wird; 2) etwas in seiner Art Vollkommenes, Vorbild; 3) Flächendekor, der auf einer Kombination eines einzelnen oder mehrerer Elemente (Figuren) beruht. [...] [Bro93, Bd. 15]

Der allgemeinen Musterdefinition des Brockhaus kann entnommen werden, daß ein Muster etwas ist, das als Vorlage für die Konstruktion von „etwas“ anderem dient. Über den Konstruktionsprozeß wird nichts gesagt, auch nicht darüber, wie Muster als Vorlage innerhalb des Prozesses eingesetzt werden. Aufgrund der Bedeutung von „Vorlage“ (ebenfalls Brockhaus), „nach der etwas hergestellt wird“, ist zu vermuten, das die Vorlage, also das Muster, und das hergestellte „Etwas“, einander ähneln. Offen bleibt, ob die Ähnlichkeit Gleichheit oder auch nur Strukturgleichheit impliziert, oder ob sie nicht isomorph ableitbare Variationen erlaubt.

Weiterhin wird unter 2) ein Muster als etwas Vollkommenes beschrieben, das zugleich als Vorbild dient. Etwas Vollkommenes ist nicht mit der Fehlerhaftigkeit der physischen Welt befaßt: Somit ist ein Muster nach der zweiten Definition immer abstrakt.

Die Definition eines Musters als Flächendekor ist zwar bereits sehr spezifisch, wirft aber eine in den beiden vorangegangenen Definitionsvarianten nicht gestellte Frage auf: Der Flächendekor basiert auf der Kombination einzelner oder mehrere Elemente. Aus was besteht ein Muster, das den beiden vorangegangenen Definitionen genügt? Der Brockhaus bleibt die Antwort schuldig, was folgern läßt, daß sie allgemein nicht bestimmbar ist und somit vom Kontext der Anwendung des Musterbegriffs abhängt.

Die letzte Definition führte den Begriff der Struktur von Mustern ein. Es ist zu vermuten, daß Struktur hier statisch zu verstehen ist. Keine der Definitionsmöglichkeiten läßt eine Aussage über innere Dynamik von Mustern ableiten. Eine genauere Bestimmung der Qualitäten der Struktur eines Mustern bleibt ebenfalls offen.

3.1.2 Sozialwissenschaftliche Definition von „Pattern“

Eine interessante Brücke zwischen dem deutschen Begriff „Muster“ und dem englischen Begriff „Pattern“ findet sich in den Sozialwissenschaften, welche als Fachbegriff im Deutschen das Konzept des „Pattern“ verwenden. Die sozialwissenschaftliche Definition ist an dieser Stelle relevant, weil ihr Gegenstandsbereich, vergleichbar der Softwareentwicklung, nicht dinglich sondern abstrakt ist. Weiterhin gibt sie einen interessanten Einblick in mögliche Strukturen von Mustern, welcher der Ausarbeitung einer Musterdefinition dienlich ist.

> **Pattern [...]**
>
> ein bestimmtes (ritualisiertes oder institutionalisiertes) Verhaltensmuster, eine soziale Grundstruktur, ein Denkmodell, eine gegliederte Anordnung oder ein aus bestimmten, immer wiederkehrenden Elementen zusammengefügtes Ablaufschema (u.a. Verhaltensprogramm oder Testvorlage). [Bro93, Bd. 16]

Ein „Pattern“ ist, so die Nennung an erster Stelle, ein bestimmtes „Verhaltensmuster“. Somit wird es als eine Ausprägung des allgemeinen zuvor definierten Musterbegriffs charakterisiert. Es ist weiterhin eine soziale Grundstruktur, mithin nicht beliebig für den Gegenstandsbereich der Sozialwissenschaften, sondern grundlegend („Grund“-Struktur).

Interessant sind hier die Strukturaspekte: „Gegliederte Anordnung“ verweist auf eine statische Struktur, die einer nicht weiter beschriebenen Gliederung genügt. Ein „aus immer wiederkehrenden Elementen zusammengefügtes Ablaufschema“ hingegen verweist auf eine Struktur, welche die dynamischen Aspekte in den Vordergrund stellt.

Laut Anleitung zur Interpretation der Begriffserläuterungen im Brockhaus ist die nicht numerierte Aneinanderreihung als Umschreibung desselben Sachverhalts zu sehen, so daß die genannten Begriffe Verhaltensmuster, soziale Grundstruktur, Denkmodell, Anordnung und Ablaufschema verschiedene aber miteinander verbundene Sichtweisen auf den zu definierenden Begriff darstellen. In der Abstraktion von Verhaltensmuster zu Muster stellt sich also die Frage, ob man letzteres somit auch als Grundstruktur, Denkmodell, gegliederte Anordnung oder gar Ablaufschema beschreiben kann. Dies muß hier zwar offen bleiben, erweitert aber den Diskussionsraum um den später zu definierenden Begriff.

3.1.3 Sprachwissenschaftliche Definition von „Pattern“

Da Muster in dieser Arbeit im Zusammenhang mit anderen Mustern verwendet werden, und einer der zentralen Autoren diesen Zusammenhang als „Mustersprache“ bezeichnet, ist die Be-

deutung des Musterbegriffs in der Sprachwissenschaft interessant. Dort wird wieder der englische Begriff „Pattern“ verwendet:

> **Pattern [...]**
>
> Strukturmodell, nach dem eine unbegrenzte Anzahl gleich gebauter Sätze mit unterschiedlichen lexikalischen Elementen gebildet werden kann, so Artikel + Substantiv (im Nominativ) + Verb (im Imperfekt) + Artikel + Substantiv (im Akkusativ). [...] [Bro93, Bd. 16]

Die sprachwissenschaftliche Definition charakterisiert den Begriff „Pattern“ über seine Struktur. Das Strukturmodell ist offenkundig eine Schablone, in welcher Platzhalter durch konkrete lexikalische Elemente des passenden Typs ersetzt werden und so zu einer unbegrenzten Anzahl strukturgleicher in der Ausprägung aber unterschiedlicher Sätze führen.

3.1.4 Zusammenfassung

Die Diskussion der verschiedenen Definitionen hat mehrere Dimensionen geöffnet, unter denen der Musterbegriff betrachtet werden kann und die somit Grundlage einer noch zu erfolgenden Definition sein können:

- Muster können abstrakt oder konkret sein.
- Muster können als Vorlage für die Konstruktion dienen (ohne den Prozeß festzulegen).
- Ein Muster kann als Schablone zur Konstruktion verwendet werden.
- Muster besitzen eine innere Struktur.
- Ein Muster kann statische oder dynamische Vorgänge beschreiben.

Es ist festzuhalten, daß bei all diesen Eigenschaften es Menschen sind, die ein Muster als solches erkennen oder zur Beschreibung von etwas entwickeln.

3.2 Der Musterbegriff bei Christopher Alexander

Auch die Architektur bietet einen interessanten Musterbegriff, der in den Arbeiten von Alexander et al. definiert wird [AIS77, Ale79]. Diesen Arbeiten kommt eine besondere Bedeutung zu, weil sie auf die softwaretechnischen Definitionsversuche des Begriffs Muster einen wichtigen Einfluß gehabt haben. Alle im nächsten Abschnitt betrachtete softwaretechnische Literatur beruft sich explizit oder implizit auf sie [Lea94].

3.2.1 Grundlagen des Musterbegriffs

Alexander gibt keine explizite Definition des Musterbegriffs, sondern charakterisiert ihn aus verschiedenen Perspektiven. Um die resultierende Begriffsbildung zu verstehen, muß das übergeordnete Ziel geklärt werden, aufgrund dessen Alexander sich überhaupt mit Mustern beschäftigt. In [Ale79] schildert er seine zentrale Prämisse:

> „There is one timeless way of building. It is thousands of years old, and the same today as it has always been. [...] It is not possible to make great buildings, or great towns, beautiful places, places where you feel yourself, places where you feel alive, except by following this way.“ [Ale79, S. 7]

Dieser „Zeitlose Weg des Bauens“[1] ist dadurch charakterisiert, daß seine „Produkte“ erkennbar „leben“. Der Weg selbst ist ein Prozeß, der in jedem Menschen lebt, mit ihm geboren wird aber heutzutage vergessen ist, tief in den Menschen selbst verborgen, so daß sie keinen Zugang mehr zu ihm besitzen. Dieser verborgene Prozeß ist operational und präzise. Er ist nicht mechanisch ausführbar. Obwohl als Methode bezeichnet, ist er keine Methode, im Gegenteil: Einmal freigesetzt, befreit er die Menschen von jeder Methode [Ale79, S. 10ff].

Im Herzen des „Zeitlosen Wegs zu Bauens“ liegt die „Qualität ohne Namen“, welche sich in jenen lebenden Gebäuden findet, die dem „Zeitlosen Weg“ zufolge gebaut wurden. Die „Qualität ohne Namen“, die den Unterschied zwischen guten und schlechten Gebäuden macht, heißt so, weil sie nicht benannt werden kann. Sie findet ihren Ausdruck in guten Gebäuden. Erkennbar wird sie daran, daß diese Gebäude auf eine subtile Art und Weise keine inneren Widersprüche und Spannungen besitzen. Alexander umschreibt die nicht benennbare Qualität folgendermaßen: Sie lebt, sie ist ganz, sie ist angenehm, sie ist frei, sie ist exakt, sie ist egolos und sie ist ewig. Jedes eingeführte Attribut wird im nächsten Schritt wieder abgeschwächt, da seine Beschreibungskraft nicht ausreicht, die Qualität ohne Namen ausreichend zu charakterisieren [Ale79, S. 25ff].

Da Menschen heutzutage der unmittelbare Zugang zum „Zeitlosen Weg des Bauens“ versperrt ist, müssen sie sich einer Designdisziplin unterwerfen. Diese Disziplin ist die Anwendung von Mustern im Rahmen einer Mustersprache, welche durch ihren Sprachcharakter das Miteinanderbauen zum Leben erweckt. Dabei wird das Pferd von hinten aufgezäumt: Muster werden vom „Zeitlosen Weg des Bauens“ generiert [Ale79, S. 11], sind also sein Ausdruck in einer konkreten baulichen Situation. Durch das Verständnis und Anwenden dieser Muster können sich Menschen wieder dem ursprünglichen Prozeß annähern und die „Qualität ohne Namen“ erreichen. Ist die Qualität einmal erreicht, so bedürfen die Menschen der Designdisziplin nicht weiter und können sie ablegen [Ale79, S. 15f].

3.2.2 Charakterisierung des Musterbegriffs

Muster sind also ein Hilfsmittel, die „Qualität ohne Namen“ zu erreichen. Was aber genau sind Muster? Alexander gibt folgende einführende Charakterisierung:

> „In order to define this quality in buildings and in towns, we must begin by understanding that every place is given its character by certain *patterns of events* [Hervorhebung, DR] that keep on happening there. [..] These patterns of events are always interlocked with certain geometric *patterns in the space* [Hervorhebung, DR]. Indeed, as we shall see, each building and each town is ultimately made out of these patterns in the space, and out of nothing else: they are the atoms and the molecules from which a building or a town is made.“ [Ale79, S. x]

Alexander gibt keine eigenständige Definition des Musterbegriffs, beschreibt aber, offenkundig ausgehend von einem Allgemeinverständnis des Wortes Muster, seine Eigenschaften. Als erstes sind Muster von Ereignissen und Muster im Raum zu unterscheiden.

---

[1] Die Wahl der Übersetzung von „timeless way of building“ als „Zeitloser Weg des Bauens“ ist nicht unkritisch. Es heißt nicht „[...] des Konstruierens“, weil dies die Konnotation eines rationalisierten Prozesses enthalten würde. Es heißt auch nicht „[...] des Wachsens“, weil dies zufallsbedingte Einflüße und Irrationalität zu sehr in den Vordergrund rücken würde. „Der Zeitlose Weg des Bauens“ soll die Intention Alexanders am klarsten ausdrücken: Der Weg ist eine Art und Weise zu Bauen, welcher als kontinuierlicher Prozeß zu verstehen ist, in dem Menschen schrittweise, sensibel die Umgebung einbeziehend, gleichwohl vorausplanend, Räume, Gebäude, Städte und anders mehr entwickeln und gestalten.

Muster von Ereignissen erfassen die Dynamik von immer wiederkehrenden Situationen. Menschen können an den dynamischen Abläufen unmittelbar beteiligt sein, müssen es aber nicht. Muster von Ereignissen sind an den Raum gebunden, passieren in ihm [Ale79, S. 63ff].

Muster im Raum beschreiben eine immer wiederkehrende statische Struktur, eine bauliche Konstellation, welche durch das Verhältnis der beteiligten Elemente charakterisiert ist. Muster im Raum sind somit immer Muster von Beziehungen zwischen Elementen. Die Beziehungen sind sogar Teil der Elemente. Sie besitzen Definitionscharakter für sie. Alexander geht noch einen Schritt weiter:

> „When we look closer still, we realize that even this view is still not very accurate. For it is not merely true that the relationships are attached to the elements: the fact is that the elements *themselves* are patterns of relationships." [Ale79, S. 88]
>
> „And finally, the things which seem like elements dissolve, and leave a fabric of relationships behind, which is the stuff that actually repeats itself, and gives the structure to a building or a town." [Ale79, S. 89]

Die beiden Zitate geben eine sehr pointierte Beurteilung der statischen Struktur eines Musters. Alexander arbeitet sie folgendermaßen aus: Was auch immer als ein konkretes oder abstraktes Element eines Musters betrachtet werden kann, entpuppt sich bei genauerem Hinsehen als weiteres Muster von Beziehungen zwischen kleineren Elementen, welche wiederum bei genauerem Hinsehen sich als Muster von Beziehungen entpuppen usw. Es läßt sich beliebig rekursiv fortsetzen. Der Begriff der physisch abgrenzbaren Entität wird zwar als pragmatisch sinnvolles Konzept beibehalten, besitzt aber keinen ontologischen Status an sich mehr.[2]

Muster von Ereignissen und Muster im Raum, Dynamik und Statik, bedingen einander und sind zwar konzeptuell, nicht aber faktisch in der konkreten Situation voneinander trennbar. Muster bilden die Elemente, aus denen jeder Raum, jedes Gebäude und jede Stadt gemacht ist. In der weiteren Ausarbeitung des Begriffs nennt Alexander drei zentrale charakterisierende Eigenschaften:

- Muster sind komplexe und mächtige Felder. Sie finden ihren Ausdruck in für Menschen erkennbaren Dingen, sind aber nicht mit ihnen gleichzusetzen. Muster sind somit die abstrakten, nicht sichtbaren Regeln und Beziehungen hinter den Dingen, welche sie zueinander in ein Verhältnis setzen.
- Muster ergeben sich als Ausgleich von Kräften mit dem Ziel, eine angespannte Situation zu „heilen". Muster sind die Reaktion auf äußere Zwänge, welche sie im Rahmen ihrer Möglichkeiten zur Disposition von Beziehungen und Dingen auszugleichen versuchen.
- Muster sind dreiteilige Regeln zum Erschaffen von konkreten Dingen, „Wesenheiten". Sie formulieren ein Problem in einem Kontext und zeigen eine Lösung auf, welche im obigen Sinne das Problem löst. Muster sind somit Regeln für den Ausgleich angespannter Beziehungen.

---

[2] Dies ist eine Beobachtung, welche auch die Physik auf mikroskopischer Ebene gemacht hat. Im Schritt von der klassischen Physik zur Atomphysik wurde der Korpuskel- respektive Objektbegriff aufgelöst und durch den Welle/Teilchen Dualismus ersetzt. Im nächsten Schritt zur Quantenfeldtheorie schließlich wurde der Welle/Teilchen Dualismus durch einen Feldbegriff, also die Charakterisierung von, in diesem Fall, sich selbst genügenden mathematischen Beziehungen, ersetzt. Der Verweis auf die Mathematik erscheint an dieser Stelle unproblematisch, weil sich die „mathematischen Wahrheiten" der Physik aus Experimenten ergeben und durch sie verifziert werden. Sie sind nicht aus einen formalen Kalkül ableitbar.

Insbesondere der dritte Punkt sei an dieser Stelle betont, weil sich aus ihm das Verständnis der im nächsten Abschnitt rezipierten softwaretechnischen Arbeiten ableitet:

> „Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." [AIS77, S. x]

Ein Muster ist also die Lösung zu einem Problem, welches vor einem bestimmten Kontext erkennbar wird. Zur Schilderung der Muster in [AIS77] greift Alexander auf genau diese Aussage zurück, indem er die Darstellung eines Musters in einen Problem- und Kontextabschnitt und einen Lösungsabschnitt unterteilt.

Alexander nennt weitere Eigenschaften von Mustern, welche die obigen charakteristischen Bedingungen ergänzen. Muster sind Abstraktionen über Dingen, gewinnen eine räumliche Ausprägung, ergeben sich aus sozialen Ereignissen, beschreiben Probleme, gleichen Kräfte aus, sind immer untereinander verbunden, haben unterschiedliche Größen, können hierarchisch angeordnet werden, sind und beinhalten Regeln, können allgemein oder projektspezifisch sein, sind Kommunikationsvehikel wie auch Kommunikationsgegenstand.

Muster besitzen eine innere Dynamik, in dem sie im Wechselspiel mit dem sie einbettenden Kontext Dinge so gestalten, daß die erwünschten Ereignisse möglichst reibungsfrei stattfinden können.

Muster dienen Menschen zur Gestaltung und zur Konstruktion von Artefakten. Die Anforderungen ergeben sich als Kontext, welcher hilft, passende Muster auszuwählen. Der Kontext im Zusammenspiel mit dem als Problem formulierbaren Wunsch, bestimmte Ereignisse zu ermöglichen, drückt sich in einer äußeren Form, dem Muster, aus. Arno Rolf schreibt im Rahmen der Gestaltungsdebatte in der Informatik über Christopher Alexander:

> „Für ihn wie die moderne Designtheorie ist Gestaltung stets in der Einheit von Kontext und Form zu betrachten: Die Form ist die Lösung des Problems und der Kontext definiert das Problem. Wenn wir über Gestaltung sprechen, so ist das tatsächliche Objekt der Diskussion also nicht allein die Form, es ist das Ensemble von Form und Kontext." [Rol93, S. 16f].

Mit dieser Betrachtung ist der wesentliche Grundstein gelegt, Muster in Form eines Problem-Kontext-Lösung Schemas zu beschreiben, wie es für diese Arbeit richtungsweisend war. Alexander wie auch der ihn zitierende Rolf betonen dabei das Wechselspiel zwischen Form, der Lösung eines Problems, und seinem Kontext, ohne im mechanistischen Sinne eines „form follows function" das eine durch das andere determiniert zu sehen (siehe auch die Diskussionen um Richtungen des Bauhaus [Wes91]).

3.2.3 Zusammenfassung

Alexander definiert unabhängig von Problemen der Softwaretechnik den Musterbegriff, der für ihn und die Architektur relevant ist. Wesentliche Aspekte eines auch in der Softwaretechnik anwendbaren Musterverständnisses sind bereits bei ihm zu finden, so die Betonung des Zusammenspiels von Form und Kontext und die für Muster resultierende Problem-Kontext-Lösung Beschreibung. Alexander wird im Abschnitt über Musterzusammenhänge noch einmal aufgegriffen.

3.3 Der Musterbegriff in der Softwaretechnik

Die ersten zwei Artikel im Bereich der Informatik, welche den Begriff Muster in ihren Mittelpunkt rückten und einem größeren Publikum bekannt machten, waren [Coa92] und [GHJ+93]. Diese beiden, wie auch die ersten größeren Beispiele in [Gam92, später GHJ+95] beschränken sich auf die Domäne des objektorientierten Entwurfs.

3.3.1 Objektorientierte Muster bei Peter Coad

In [Coa92] gibt Peter Coad zuerst eine sehr allgemeine Definition basierend auf [Web86], Merriam Webster's International Dictionary. Muster sind „Patterns," definiert als „a fully realized form, original or model accepted or proposed for imitation: something regarded as a normative example to be copied; archetype; exemplar."

Nach einem kurzen aber weitschweifenden Überblick über den Musterbegriff in anderen Disziplinen kommt er schließlich zur Softwareentwicklung, genauer: objektorientierter Analyse und Design. Hier bilden Muster bestehend aus mehreren Objekten und Klassen effektivere Bausteine für objektorientierte Analyse und Design als das einzelne Objekt oder die einzelne Klasse. Er gelangt zur Definition von objektorientierten Mustern:

> „An object-oriented pattern is an abstraction of a doublet, triplet, or other small grouping of classes that is likely to be helpful again and again in object-oriented development." [Coa92, S. 153]

Durch diese Definition fokussiert Coad seinen im Ansatz allgemeinen Musterbegriff auf die objektorientierte Konstruktion, definiert also objektorientierte Muster (Analyse- oder Entwurfsmuster). Die genannte Definition betont die Form des Musters, indem sie die Struktur eines Tupels von Klassen in den Vordergrund rückt.

Die von ihm angeführten Beispiele sind recht allgemein gehalten und könnten auch gut aus Lehrbüchern über objektorientierte Modellierung stammen. Die genannten Muster bestehen meist aus wenigen (2-4) Klassen und dürften den meisten Softwareentwicklern unmittelbar einleuchten. Es handelt sich dabei zumeist um spezialisierte Verwendungen der bekannten Vererbungs- und Aggregationsbeziehungen zwischen Klassen und Objekten.

Wichtig ist noch, daß Coad, wie auch die meisten anderen Forscher auf diesem Gebiet, die pragmatische Seite der Mustergewinnung betont: Muster erwachsen aus in der Konstruktion objektorientierter Systeme gewonnener Erfahrung. Coad schließt mit den von Alexander entlehnten Worten: „Patterns are the molecules from which one may apply OOA and OOD more effectively".

3.3.2 Entwurfsmuster bei Gamma et al.

Die rasch steigende Bedeutung, welche Entwurfsmuster in der internationalen Diskussion gewonnen haben, ist zu einem erheblichen Teil auf die Arbeiten der sog. Hillside-Group zurückzuführen.[3] Hinter diesem Namen verbergen sich mehrere Wissenschaftler, die zuerst auf den OOPSLA Workshops von Bruce Anderson für objektorientierte Architektur und später durch Veröffentlichungen die Diskussion vorangebracht und einem breiten Publikum eröffnet haben.

---

[3] Maßstab für eine solche Aussage ist hier die Anzahl von Veröffentlichungen zum gegebenen Thema. Zählt man sie zusammen (was noch möglich ist), so stellt man fest, daß ein Großteil der Autoren aus dieser Gruppierung stammt oder ihr nahe steht.

Die Hillside-Group läßt sich in ihrer öffentlichen Wahrnehmung in mehrere Strömungen untergliedern: die „gang-of-four“[4] um Ralph E. Johnson [GHJ+93, GHJ+95], die traditionell zusammenarbeitenden Kent Beck und Ward Cunningham [BC89, Bec94], Richard P. Gabriel [Gab93, Gab94] sowie die Workshops von Bruce Anderson [And93, And94].

Grundlage der folgenden Diskussion sind die Arbeiten der „gang-of-four“, bestehend aus Erich Gamma, Richard Helm, Ralph E. Johnson und John Vlissides. Die wichtigsten Arbeiten sind [GHJ+93] und der sich anschließende Entwurfsmuster-Katalog [GHJ+95] sowie die deutschsprachige Dissertation von Erich Gamma [Gam92]. In ihrem ECOOP-93 Papier [GHJ+93] charakterisieren die genannten Wissenschaftler „design patterns“, d.h. Entwurfsmuster, folgendermaßen:

> „Design patterns identify, name, and abstract common themes in object-oriented design. They preserve design information by capturing the intent behind a design. They identify classes, instances, their roles, collaborations, and the distribution of responsibilities.“ [GHJ+93, S. 407]

Entwurfsmuster sind ein Mittel, Erfahrungen im objektorientierten Softwareentwurf festzuhalten und zu kommunizieren. Sie beschreiben die Bedeutung, den Zusammenhang und die Verteilung von Verantwortlichkeiten zwischen Objekten und Klassen und stellen die Abstraktion von konkreten, immer wiederkehrenden Problemlösungen dar. Ihre Entwicklung basiert auf entsprechender Entwurfserfahrung. Die genannten Wissenschaftler führen folgende Qualitäten an:

- Entwurfsmuster stellen für den Softwareentwurf eine Terminologie zur Verfügung, welche die Kommunikation und Dokumentation erleichtert und erweitert. Die Abstraktionsbildung von konkreten Lösungen erhöht die Übertragbarkeit einer Lösung und erleichtert das Verständnis des diskutierten Entwurfs. Es ermöglicht somit tiefer als zuvor den Vergleich mehrerer Entwurfsalternativen.
- Entwurfsmuster bilden eine wiederverwendbare Basis von durch Entwurfserfahrung gewonnen Lösungen. Sie bieten die Möglichkeit, langjährige Erfahrung von Experten festzuhalten und zu vermitteln. Entwurfsmuster bilden Mikroarchitekturen, welche als Bausteine für den Entwurf von Softwarearchitekturen verwendet werden können.
- Entwurfsmuster verringern die Einarbeitungszeit in bestehende Software, indem sie den Blick auf das Wesentliche lenken. Sind die zentralen Muster einmal verstanden, so können sie in ihren konkreten Ausprägungen leicht wiedererkannt werden. Dies hilft, schnell ein Verständnis existierender Softwarearchitekturen zu erwerben.
- Entwurfsmuster stellen ein Hilfsmittel zur Überprüfung und zum Umbau bestehender Software dar. Im Vergleich kann Software besser bewertet werden und gegebenenfalls nach bekannten Mustern reorganisiert werden („musterbasiertes Reengineering“). Die frühzeitige Anwendung von Entwurfsmustern verhindert allzuhäufige Reorganisation von Klassenbäumen und Benutzungsstrukturen, da funktionsfähige Lösungen schneller erkannt und verwendet werden können.

Der Erfahrungshintergrund der Autoren ist beeindruckend, da sich hier vier Wissenschaftler zusammengefunden haben, deren Anwendungsrahmen in Praxis und wissenschaftlicher Literatur einen exponierten Platz genießen: Erich Gamma ist Mitentwickler von ET++ [WGM89, WG94], Richard Helm arbeitete direkt oder indirekt am Demeter System [HHG90] sowie QOCA mit [HHM+92], Ralph Johnson verwendete und publizierte über Hotdraw [Joh92,

---

[4] Die Bezeichnung „gang-of-four“ ist von den Genannten selbst gewählt worden.

BJ94], entworfen und entwickelt von Kent Beck und Ward Cunningham, und zu guter Letzt John Vlissides, der Mitautor von Interviews und Unidraw ist [LVC89, Vli90].

3.3.3 Meta Patterns bei Wolfgang Pree

Als Reaktion auf Coad und Gamma et al. führt Wolfgang Pree „Meta Patterns“ ein: „We introduce the term *meta patterns* for a set of design patterns that describes how to construct frameworks independent of a specific domain“ [Pre94]. Pree argumentiert, daß sich die genannten Entwurfsmuster von Coad und Gamma et al. (die bereits angeführten [Coa92, GHJ+93, GHJ+95]) nur bis zu einem bestimmten Grad von ihrer ursprünglichen Anwendungsdomäne entfernt haben und trotz Abstraktionsbildung domänenspezifisch geblieben sind.

Die von Pree angeführten Meta Patterns sind so allgemein, daß sie domänenunspezifisch sind. Sie entspringen der Diskussion über Qualitätsmerkmale objektorientierten Softwareentwurfs der letzten Jahre und basieren auf bekannten Entwurfsmustern, etwa von Gamma et al. Genannt werden als Meta Patterns Template und Hook Methods, Narrow Inheritance Interface Principle sowie Kompositionsstrukturen aus 1-2 Klassen.

Meta Patterns sind gemäß der Bedeutung des Wortes Metamuster von Mustern, in dem sie domänenspezifische Muster als Ausprägung von Mustern auf einer Metaebene begreifen. Sie sind als domänenunabhängige Entwurfsmuster folgendermaßen einsetzbar:

- Meta Patterns können zur Beschreibung domänenspezifischer Entwurfsmuster verwendet werden, welche Pree als „framework example design patterns“ bezeichnet.
- Sie ermöglichen die domänenunabhängige Beschreibung eines Anwendungsrahmens.
- Und sie können als grundlegende Abstraktionen zum Entwurf von Entwurfsmusterwerkzeugen dienen, vermutlich also als allgemeine Beschreibungsform die Grundlage für die Datenbasis eines solchen Werkzeugs legen.

Meta Patterns finden ihren Platz als Ergänzung bisheriger Entwurfsmuster. Pree: „So meta patterns do not replace state-of-the-art design patterns approaches but complement them.“

3.3.4 Zusammenfassung

Coad definiert zuerst einen allgemeinen und dann einen objektorientierten Musterbegriff. Die von ihm genannten Beispiele sind recht allgemein gehalten. Gamma et al. führen erstmals den Präfix Design, also auf Konstruktion ausgerichtet, ein. Die Autoren greifen auf umfangreiche persönliche Erfahrung im Entwurf von Anwendungsrahmen zurück. Coad bezieht sich dabei explizit auf Alexander, wohingegen Gamma et al. ihre eigenen Definitionen geben. Pree ergänzt die beiden Papiere um eine weitere abstrakte Sicht auf grundlegende Muster im objektorientierten Entwurf. Unklar bleibt, ob das Wort „Meta Patterns“ gerechtfertigt ist, oder ob es sich nicht eher um einfache, domänenunabhängige Muster handelt.

3.4 Definitionen der Musterbegriffe

Im Anschluß an die Diskussion existierender Definitionen wird der dieser Arbeit zugrundeliegende Musterbegriff definiert. Die gewonnenen Erkenntnisse werden zusammengefaßt, der Musterbegriff ausgearbeitet und von verschiedenen Perspektiven aus betrachtet. Es werden drei Spezialisierungen des allgemeinen Begriffs, namentlich Interpretations- und Gestaltungsmuster, Entwurfsmuster und Programmiermuster vorgestellt.

3.4.1 Definition und Diskussion Muster

Der Begriff „Muster“ sei folgendermaßen definiert:

> **Definition 3-1: Muster**
>
> Ein Muster ist eine in einem bestimmten Kontext erkennbare Form. Es dient als Vorlage zum Erkennen, Vergleichen und Erzeugen von Musterexemplaren. Ein Muster ist die Essenz aus Erfahrung und Analyse immer wiederkehrender Situationen. Es besitzt eine innere Struktur und Dynamik.

In dieser Definition sind verschiedene Bereiche angelegt, welche im folgenden diskutiert werden: Ein Muster ist eine Form in einem Kontext, ein Muster dient als Vorlage zum Erkennen, Vergleichen und Erzeugen von Musterexemplaren; es wird entdeckt; es besitzt eine innere Struktur und Dynamik.

Ein Muster ist eine in einem bestimmten Kontext erkennbare Form. **Muster sind immer im Kontext zu sehen, welcher ihre Form und Struktur verständlich macht. Die Form eines Musters läßt sich al**s Reaktion auf seinen Kontext beschreiben. Muster entstehen nicht nur in einem bestimmten Kontext, sondern werden auch immer in einem bestimmten Kontext angewendet. Um ein Muster zu verstehen, muß die Form des Muster immer mit seinem Kontext zusammen beschrieben werden.

Ein Muster dient als Vorlage zum Erkennen, Vergleichen und Erzeugen von Musterexemplaren. Muster erfassen das Wesentliche, all ihren Exemplaren Gemeinsame. Hierbei ist Erkennen, Vergleichen und Erzeugen von Musterexemplaren ein unbestimmter Prozeß. Weder eine Verfahrensvorschrift **noch eine allgemeine Methodik sind vorgegeben.**

Ein Muster entsteht durch Reflexion, Erfahrung und Analyse von abstrakten oder physischen Artefakten. Somit werden sie von Menschen gemacht. Dies heißt nicht, daß bestimmte Strukturen nicht existieren würden. Es sind aber immer Menschen, welche bestimmte Formen anschauen, versuchen sie zu verstehen, versuchen ihren Kontext zu verstehen und dabei notwendig konstruierend vorgehen. Ergebnis eines solchen Prozesses ist die Beschreibung und das Verständnis einer Form als Muster. Menschen geben ihr eine Bedeutung.

John Vlissides, einer der Autoren des Entwurfsmuster Katalogs [GHJ+95], schreibt über den Prozeß des Findens von Mustern:

> „The single most important activity [...] is reflection. You must take time off periodically to reflect on what you’ve done. [...] Another important activity is to look at as many systems as you can, systems designed by other people. The best way to learn from other systems is to actually build with them. [...] Seek an understanding of the problems they address and how they address them. [...] Try to identify patterns that you already know. [...] If you do find something that seems new, make sure it is applicable in other contexts before you try to write it up as a pattern. We had one inviolable rule as we developed *Design Patterns*: we had to find *two* existing examples of a problem and its solution before we would write a pattern for it.“ [Vli95, S. 7]

In diesem Ausschnitt werden alle wesentlichen Punkte angesprochen: Reflexion über die eigene Arbeit, Erfahrung mit eigenen und fremden Systemen und Analyse anderer Arbeiten. Ein Muster, so die Faustregel, sollte erst dann aufgeschrieben werden, wenn mindestens zwei Exemplare in unterschiedlichen Systemen gefunden wurden.

Ein Muster besitzt eine innere Struktur und Dynamik. Muster besitzen eine Form, welche sich als Struktur darstellen läßt, die den Aufbau und die inneren Zusammenhänge des Musters repräsentiert. Muster besitzen eine innere Dynamik, welche das Zusammenspiel mit dem Kontext erläutert. Struktur eines Musters und seine Dynamik bedingen einander, so wie bereits bei Alexander erläutert.

3.4.2 Definition und Diskussion Interpretations- und Gestaltungsmuster

Bis jetzt wurde nicht geklärt, auf welche Arten ein Muster als Vorlage dienen kann. Insbesondere wurde seine Bedeutung für die Softwaretechnik nicht geklärt. Dies soll nun nachgeholt werden. Es folgen drei verschiedene Möglichkeiten, Muster als Form im Kontext zu begreifen und in der Softwaretechnik zu verwenden.

> **Definition 3-2: Interpretations- und Gestaltungsmuster**
>
> Ein Interpretations- und Gestaltungsmuster ist ein Muster, welches zur Interpretation und Gestaltung von tatsächlichen oder antizipierten Anwendungssituationen und Softwaresystemen verwendet werden kann.

Interpretations- und Gestaltungsmuster sind *orientierend* für die Interpretation der Anwendungswelt und die Gestaltung eines in der Anwendungswelt eingebetteten Softwaresystems. Sie erläutern, wie Anwendungssituationen interpretiert und mit welchen Begriffen sie beschrieben und gestaltet werden können. Die in Kapitel 2 geschilderten Metaphern Werkzeug und Material stellen Vorläufer der Interpretations- und Gestaltungsmuster dar, wie sie im nächsten Kapitel ausgearbeitet werden. Interpretationsmuster besitzen verschiedene Qualitäten:

- Sie dienen der Interpretation der Anwendungswelt sowie der qualitativen Konstruktion des neuen Systems.
- Sie bieten aufgrund des allgemeinen Zugangs, den Menschen zu den verwendeten Begriffen (vgl. die Metaphern) haben, einen fruchtbaren Ausgangspunkt, Qualitäten des Systems herauszuarbeiten und zu diskutieren.
- Sie dienen als Katalysator für ein geteiltes und kommunizierbares Verständnis der Anwendungswelt.
- Sie lassen sich bei Bedarf selbst zum Thema machen und ermöglichen eine qualitative Diskussion ihres Einsatzes und des Arbeitskontexts.
- Und sie sind ausreichend nahe am Softwareentwurf, so daß der Übergang zum konkreten softwaretechnischen Entwurf leicht geschehen kann.

Interpretations- und Gestaltungsmuster dienen dazu, die Anwendungswelt zusammen mit den zukünftigen Benutzenden des Systems zu erfassen, zu interpretieren und auf Basis eines sich entwickelnden gemeinsamen Verständnisses der fachlichen Zusammenhänge das System auch qualitativ zu konstruieren.

Interpretationsmuster fallen nicht vom Himmel, sondern werden langsam erarbeitet. Entwickler bringen ein Verständnis der Muster in einen partizipativen Entwicklungsprozeß mit und bilden dort ein projektspezifisches Verständnis heraus, welches sich aus den Diskussionen im Projekt ergibt.

Im Vorfeld oder spätestens zu Beginn des Projekts muß geklärt werden, welche Interpretationsmuster zum Tragen kommen sollen. Hierbei wird zumeist auf das Wissen der Entwickler oder Projektbegleiter zurückgegriffen, welche die möglichen Muster anhand ihres Kontexts auf

ihre Verwendbarkeit hin evaluieren. Die Schwierigkeit liegt darin, nicht vorschnell bestimmte Muster in den Vordergrund zu stellen und somit aufgrund eines Modellmonopols [Bra72, Bra88] der Entwickler frühzeitig bestimmte Interpretationen festzulegen.

3.4.3 Definition und Diskussion Entwurfsmuster

Interpretationsmuster werden in dieser Arbeit von Entwurfsmustern ergänzt, welche als Vorlagen für die Konstruktion eines objektorientierten Entwurfs dienen.

> **Definition 3-3: Entwurfsmuster**
>
> Ein Entwurfsmuster ist ein Muster, das als Vorlage für die Konstruktion eines softwaretechnischen Entwurfs dient. Ein Entwurfsmuster besteht aus dem Zusammenspiel technischer Elemente wie Klassen, Objekte und Operationen. Die Struktur und Dynamik eines Entwurfsmusters klärt die beteiligten Komponenten, ihre Zusammenarbeit und die Verteilung der Verantwortlichkeiten.

Diese Definition entspricht weitgehend den von Gamma et al. genannten Qualitäten für objektorientierte[5] Entwurfsmuster. Es gilt:

- Entwurfsmuster bilden effiziente Vorlagen für den technischen Entwurf, welche von unwichtigen Details abstrahieren und den Blick auf das Wesentliche richten.
- Sie erleichtern Kommunikation, Verständlichkeit und Dokumentation eines Entwurfs.
- Sie bilden Mikroarchitekturen, welche flexibel kombiniert werden können, um eine Softwarearchitektur zu entwickeln.
- Sie können als Analysemittel bestehender Software dienen und zur ihrer Restrukturierung führen.
- Und sie erleichtern die Einarbeitung und das Verständnis von Software, welche mit ihrer Hilfe dokumentiert wurde.

Entwurfsmuster sind, im Gegensatz zu Interpretationsmustern, nur für das Entwicklungsteam gedacht. Es diskutiert mit ihrer Hilfe die zu entwickelnde Softwarearchitektur. Hierbei lenken Entwurfsmuster die Diskussion auf das Wesentliche eines Entwurfs, in dem sie von unwichtigen Details abstrahieren. In Form von Mikroarchitekturen, welche einzelne im Entwurfsprozeß auftretende Probleme lösen, tragen sie zur gesamten Architektur bei.

3.4.4 Definition Programmiermuster

Neben den genannten Musterarten gibt es noch Programmiermuster. Programmiermuster sind jene Muster, die ein Informatiker aufgrund seiner Ausbildung als elementares Wissen der Programmierung mitbringt. Sie sind den „Programming Idioms“ von James Coplien vergleichbar: Es sind wiederverwendbare Ausdrücke, mit denen ein Softwareentwurf einfach und klar in einer bestimmten Sprache implementiert werden kann [Cop92].

---

[5] Die Definition ist offenkundig objektorientiert ausgerichtet. Für andere Paradigmen, etwa applikative Sprachen und Sprachsysteme wird man auf andere Komponenten zurückgreifen müssen.

Definition 3-4: Programmiermuster

Ein Programmiermuster ist ein Muster, das als Vorlage für die Implementation eines Entwurfs dient. Sie basieren auf Erfahrungswissen der Programmierung und sind in der jeweiligen Programmierkultur weithin bekannt.

Elementares Wissen der Programmierung ist Erfahrungswissen, wie bestimmte Konzepte, so z.B. die Unterscheidung von Wert und Objekt [Mac82] oder das Substituierbarkeitsprinzip [Lis88], in Programmiersprachen umgesetzt werden können. Werte und Objekte können z.B. in C++ über unterschiedliche Verwendung von C++ Wert-, C++ Referenz- und C++ Zeigersemantik umgesetzt werden. Das Substituierbarkeitsprinzip wird zuerst durch Typüberprüfung vom Übersetzer abgesichert und später durch dynamisches Binden umgesetzt.

Die Verwendung von Programmiermustern ist weitgehend unbewußt, weil sie in langen Jahren eingeübt wurden und ihre Anwendung nicht weiter reflektiert zu werden braucht. Programmiermuster sind somit von der Programmierkultur und -sprache abhängig.

3.4.5 Die resultierenden Musterebenen

Aus der Definition der Begriffe Interpretations- und Gestaltungsmuster, Entwurfsmuster und Programmiermuster resultieren drei Ebenen, welchen in Softwareprojekten unterschiedliche Bedeutung zukommt:

Die Ebene der Interpretations- und Gestaltungsmuster. Interpretationsmuster dienen der Interpretation der Anwendungswelt, sind nicht technisch und müssen reflektiert und bewußt eingesetzt werden. Sie stellen die Ausarbeitung allgemeinverständlicher Muster dar, welche allen Beteiligten zur fachlichen Orientierung dient. Sie können zumeist analytisch wie konstruktiv verwendet werden. Sie erlauben die Formulierung qualitativer Anforderungen an ein Softwaresystem, ohne sich technischer Termini bedienen zu müssen. Sie stellen sowohl eine Perspektive auf die Anwendungswelt und das zu konstruierende Softwaresystem dar, als auch ein Hilfsmittel zur Gestaltung eben dieses Systems, sind Zugang und Weltbild gleichermaßen.

Die Ebene der Entwurfsmuster. Interpretations- und Gestaltungsmuster werden durch Entwurfsmuster softwaretechnisch umgesetzt. Objektorientierte Entwurfsmuster bestehen aus Klassen, Objekten und Methoden, welche dem Muster zufolge in einer bestimmten Art und Weise zusammenarbeiten, um ein gesetztes Ziel zu erreichen. Sie werden allein von Entwicklern verstanden, reflektiert und bewußt eingesetzt. Kontext eines Entwurfsmusters sind sowohl die technischen Randbedingungen sie umgebender Muster als auch der durch die Interpretationsmuster gesetzte Bedeutungsrahmen.

Die Ebene der Programmiermuster. Zur Umsetzung eines Softwareentwurfs in ein Programmsystem bedienen sich Softwareentwickler eines reichen Erfahrungsschatzes von Programmiermustern, welche sich konkret in Programmiersprachen formulieren lassen. Dies ist die Ebene der Programmierung, welche sich jeder Entwickler angeeignet hat und beständig weiterentwikkelt. Die Verfügbarkeit von Programmiermustern wird in der Kommunikation unter Entwicklern üblicherweise vorausgesetzt, da sie das elementare Wissen um gute Programmierung enthält, das zu einer Ausbildung gehört.

3.4.6 Eine Beschreibungsform für Muster

Ein Großteil dieser Arbeit, die Kapitel 4 bis 6, bestehen aus Mustern für die Werkzeug und Material Metapher. Die Darstellung der Muster bedarf eines einheitlichen Formats, um sie effizient ein- und anordnen, Bezüge herausstellen und sie besser erläutern zu können.

Die Darstellung eines Musters ist unterteilt in mehrere Abschnitte. Die drei wesentlichen Kategorien, von Alexander abgeleitet und in [RZ95] als tragfähig erwiesen, sind der Problem-, Kontext- und Lösungsabschnitt. Der Problemabschnitt schildert in 1 bis 2 Sätzen das Problem, der Kontextabschnitt schildert den Kontext, die Randbedingungen und die zu schlichtenden Kräfte, und der Lösungsabschnitt schildert eine Form, welche die Kräfte für die gegebenen Randbedingungen schlichtet. Diese Darstellungsform basiert weitgehend auf Alexanders Erläuterung von Mustern als Regel:

> „We see, in summary, that every pattern we define must be formulated in the form of a rule which establishes a relationship between a context, a system of forces which arise in that context, and a configuration which allows these forces to resolve themselves in that context.“ [AIS77, S. 253]

Die Regelhaftigkeit eines Musters führt dazu, den Lösungsabschnitt, der eine Konfiguration schildern soll, als eine Abfolge von Anweisungen zu verstehen. Folgt man den Anweisungen, so ergibt sich die erwünschte Konfiguration. Dabei ist festzuhalten, daß die Anweisungen jeweils ausreichend abstrakt bleiben und lediglich den geschilderten Randbedingungen genügen, nicht aber den Spielraum unnötig einengen.

Ein Muster beginnt mit einer kurzen Aufgabenbeschreibung von wenigen Zeilen. Es schließen sich mehrere Abschnitte an, welche in der anschließenden Auflistung erläutert werden. Das angeführte Musterformat wird für Interpretations- und Gestaltungsmuster sowie Entwurfsmuster gleichermaßen verwendet werden.

Problem Die Problemformulierung gibt eine möglichst kurze und präzise Erläuterung, was das Problem ist und warum es ein Problem ist, das gelöst werden muß.

Kontext Der Kontext erläutert das Umfeld des Problems, diskutiert seinen Hintergrund, seine Beziehung zu anderen Problemen und welche Gegebenheiten für eine Problemlösung zu beachten sind. Es klärt somit die Randbedingungen, denen die Dynamik und Statik einer Lösung genügen muß. Günstig ist es, hier ein Beispiel einzuführen, welches dann im folgenden weiter besprochen wird.

Lösung Dieser Abschnitt schildert in Form einer operationalisierbaren Beschreibung die für das Problem vorgeschlagene Lösung. Er arbeitet Struktur und Dynamik der Lösung heraus, diskutiert Vor- und Nachteile, zeigt Konsequenzen auf und führt praktische Erfahrungen an. Im Falle eines bereits eingeführten Beispiels wird die Lösung daran illustriert.

Zusatz Gibt es alternative Lösungen oder Erweiterungen, welche nicht zum eigentlichen Muster gehören aber genannt werden sollten, so werden sie in diesem Abschnitt angeführt. Hier kann auf andere Muster verwiesen werden, die nicht weiter aufgegriffen werden.

Vergleiche Dieser Abschnitt gibt Hinweise auf verwandte Arbeiten, in denen in ähnlicher Form das Muster bereits auftauchte oder eine das Muster berührende interessante Diskussion geführt wurde.

3.5 Der Musterzusammenhang bei anderen Autoren

Muster bei Christopher Alexander stehen nicht allein, sondern werden im Rahmen einer Mustersprache („pattern language“) präsentiert. Alexander versteht Muster als Elemente einer Mustersprache, welche für die Architektur die Ausdrucksmächtigkeit einer normalen Sprache besitzt und zur Gestaltung von Städten, Gebäuden und Räumen verwendet werden kann. Eine „lebende“ Sprache würde zu einer „lebenden“ Umwelt führen.

Auch in der Informatik wird unter Rückgriff auf Alexander der Begriff Mustersprache verwendet. Gleichwohl scheuen alle hier diskutierten Autoren davor zurück, den Begriff Sprache für ihre Arbeit zu verwenden. Dies gilt auch für den Autor dieser Arbeit.

Mit dem Sprachbegriff aufgeworfen wird die Frage, welche Abhängigkeiten zwischen mehreren Mustern bestehen können und wie sie darzustellen sind. Dazu wird in den nächsten Abschnitten wieder die entsprechende Literatur diskutiert und versucht, die relevanten Dimensionen herauszuarbeiten. Im Anschluß werden die Konsequenzen für die Darstellung von Musterzusammenhängen gezogen.

Der Entwurfsmuster-Katalog [GHJ+95] wurde trotz seiner guten Ausarbeitung nicht in die Liste der diskutierten Arbeiten aufgenommen, weil er in der Tat ein Katalog ist: Beziehungen zwischen Mustern werden zwar aufgeführt, aber nur wenig diskutiert. Das dem Katalog zugrundeliegende Verständnis wurde bereits unter 3.3.2 diskutiert.

3.5.1 Der Begriff der Mustersprache bei Christopher Alexander

Der Begriff Mustersprache wurde erstmals im Werk Alexanders [Ale79, AIS77] verwendet. Den Zusammenhang von Mustern bezeichnet Alexander als Mustersprache. Ein einzelnes Muster wie auch eine Mustersprache sind darauf ausgerichtet, einen Prozeß in Gang zu setzen und am Leben zu erhalten, welcher sich im „Zeitlosen Weg zu Bauen“ ausdrückt und die „Qualität ohne Namen“ in der Gestaltung von Gebäuden usw. widerspiegelt (siehe 3.2). Diese Erwartung an seine oder fremde Mustersprachen hat sich nicht erfüllt. Alexander schreibt:

> „All the architects and planners in Christendom, together with The Timeless Way of Building and the Pattern Language, could still not make buildings that are alive because it is other processes that play a more fundamental role, other changes that are more fundamental.“ Zitiert nach [Gab94, S. 84]

Dieses Zitat wurde an den Anfang der folgenden Diskussion über Musterzusammenhänge gestellt, um keine falschen Hoffnungen zu wecken und die Diskussion selbst zu fokussieren. Der Begriff der Mustersprache wird im folgenden lediglich auf die strukturellen Zusammenhänge von Mustern hin untersucht.

> „In short, no pattern is an isolated entity. Each pattern can exist in the world, only to the extent that it is supported by other patterns: the larger patterns in which it is embedded, the patterns of the same size that surround it, and the smaller patterns which are embedded in it.“ [AIS77, S. xiii]

Alexander geht von der Beobachtung aus, daß Muster nie isoliert betrachtet werden können, sondern immer im Zusammenhang gesehen werden müssen. Diesen Zusammenhang bezeichnet Alexander als Mustersprache.

> „A pattern language is a system which allows its users to create an infinite variety of those three dimensional combinations of patterns which we call buildings, gardens, towns. [...] In summary: both ordinary languages and pattern languages are finite combinatory systems which allow us to create an infinite variety of unique combinations, appropriate to different circumstances, at will.“ [Ale79, S. 186f]

Eine Mustersprache besteht aus Mustern, welche netzwerkartig, gleichwohl hierarchisch angeordnet sind. Das gewählte Ordnungskriterium ist die Größe eines Musters: Muster können von

Mustern umgeben sein und Muster können weitere Muster enthalten. Die resultierende Struktur läßt sich als azyklischer Graph beschreiben.

Es resultiert eine, wenngleich durch die Graphenstruktur eingeschränkte, so doch große kombinatorische Vielfalt von Mustern. Das Einfügen neuer oder das Ändern alter Muster läßt die Mustersprache kohärent wachsen. Jede konkrete Stadt, jedes Gebäude und jeder Raum lassen sich im Endeffekt als die Überlagerung von Exemplaren von Mustern begreifen. Somit lassen sich alle architektonischen Strukturen als Musteranwendungen beschreiben.

Prozeß und Ergebnis sind hierbei voneinander zu trennen. Es wird nichts über den Prozeß gesagt, sondern lediglich, daß als Ergebnis menschlichen Schaffens Strukturen entstehen, welche als Muster erfaßt und beschrieben werden können. Ob die Anwendung einer Mustersprache der täglichen Benutzung unserer Umgangssprache vergleichbar ist und die erhoffte „Qualität ohne Namen“ ihren Ausdruck findet, bleibt im Dunkeln. Zwar hegte Alexander diese Hoffnung, hat sie aber, wie bereits gesagt, wieder verwerfen müssen.

3.5.2 Muster bei Ralph Johnson

In [Joh92] widmet Ralph E. Johnson sich dem Thema der Benutzungsdokumentation von Anwendungsrahmen. Er greift auf das Alexandersche Verständnis von Mustersprachen zurück, welche er aber für seiner Arbeit von der Bezeichnung her durch das Wort Muster (im Plural) ersetzt („patterns“ statt „pattern language“), um einer Verwechslung mit formalen Sprachen vorzubeugen. Johnson:

> „A pattern language is a structured essay, not a mathematical object. Therefore, we will replace that term with the term patterns.“ [Joh92, S. 63]

Johnson sieht die Aufgabe von Mustern folgendermaßen:

> „The main purpose of a set of patterns is to show how to use a framework, not to show how it works, but patterns can also describe a great deal of the theory of its design.“ [Joh92, S. 63]

In der Dokumentation eines Anwendungsrahmens kommt dem ersten Muster eine zentrale Bedeutung zu:

> „The first pattern for a framework describes its application domain. It does this by giving examples, as do other ways of documenting frameworks. In addition, the first pattern introduces the rest of the patterns in the language, and it will usually tell which patterns should be read next. Thus it acts both as a catalog entry for the framework and as a road map.“ [Joh92, S. 64f]

Alle Entwurfsmuster sind von diesem ersten Muster aus erreichbar, dem eine zentrale Bedeutung zukommt. Es klärt das Anwendungsfeld des Anwendungsrahmens und liefert die Motivation für seine Verwendung. Es wird von Entwicklern als erstes gelesen, so daß an dieser Stelle bereits erkennbar ist, ob der Anwendungsrahmen für ihre Bedürfnisse prinzipiell passend ist.

Das sich entspinnende Beziehungsgeflecht der Muster stellt wiederum einen azyklischen Graphen dar. Je wahrscheinlicher die Verwendung und je umfassender der Anwendungsbereichs eines Musters ist, desto kürzer ist der Weg vom Ausgangsmuster zu ihm. Detaillierte Designin-

formation über Muster und Anwendungsrahmen selbst entfernt sich immer weiter vom ersten Muster.

Neben diesen strukturellen Aspekten basiert die Benutzungsdokumentation des Anwendungsrahmen massiv auf dem Einsatz von Beispielen, wie sie auch von Alexander durchgängig eingesetzt werden.

Als Beispiel für einen solcherart dokumentierten Anwendungsrahmen nennt Johnson Hotdraw, für das er Muster im Anhang des Papiers angibt. Hotdraw ist ein Anwendungsrahmen für die Erstellung graphischer semantischer Editoren. Der Einsatz der Muster als alleiniger Dokumentation des Anwendungsrahmens hat sich in der Praxis bewährt, obwohl er nie formal überprüft wurde.

3.5.3 Generative Muster bei Beck und Johnson

Zwei Jahre später stellten Kent Beck und Ralph Johnson eine andere Perspektive auf Muster vor. Sie analysieren wiederum Hotdraw, richten die gefundenen Muster aber auf einen neuen Zweck hin aus [BJ94]. Sie stellen auf den „generativen" Aspekt der Muster bei Alexander ab, welcher festhält, wann ein Muster angewendet werden soll. Ihre erklärte Absicht ist:

> „This paper shows that patterns can be used to derive architectures, much as a mathematical theorem can be proved from a set of axioms. When patterns are used in this way, they illuminate and motivate architectures. Deriving an architecture from patterns records the design decisions that were made, and why they were made that way." [BJ94, S. 140]

Im Gegensatz zu den vorigen Mustern für Hotdraw, welche aufzeigen, wie man Hotdraw verwendet, erläutern diese Muster, wie der Anwendungsrahmen selbst erzeugt wurde. Insofern handelt es sich nicht um eine umfassende Menge von Mustern, sondern vielmehr um eine Anordnung von Mustern zur Begründung der resultierenden Architektur.

Die Ableitung von Hotdraw wie sie die Autoren vorführen, ist nicht als Vorgehen in der Zeit mißzuverstehen: Es stellt vielmehr eine Rationalisierung des Enstehungsprozesses dar, so daß im Parnasschen Sinne [PC86][6] die vermeintliche Rationalität der Ableitung im nachhinein vorgetäuscht wird und der kreative Konstruktionsprozeß verborgen bleibt.

In der Darstellung der Ableitung wird jeweils ein Problem formuliert, für welches die Lösung mit Hilfe eines Musters ausgeführt wird. Die Abfolge der Probleme und Muster ist streng top-down, beginnend mit der Abstraktion „Benutzungsschnittstelle" über „Editor" bis zum „Umwickler". Neue Muster werden unter Bezugnahme auf alte Muster eingeführt, deren offengebliebene Fragen sie klären. Beck und Johnson:

> „Describing an architecture with patterns is like the process of cell division and specialization that drives growth in biological organisms. The design starts as a fuzzy cloud representing the system to be realized. As patterns are applied to the cloud, parts of it come into focus. When no more patterns are applicable, the design is finished." [BJ94, S. 142]

---

6 Parnas wird von Beck und Johnson selbst zitiert.

Zwar ist die Abfolge der Muster stringent, es fällt aber auf, daß jede Problemformulierung, die ein neues Muster motiviert, zu einem Teil auch immer aus externen an das Gesamtsystem gerichteten Anforderungen besteht. Sie bezieht sich also auf einen unausgesprochenen Kontext.

Die Muster des Entwurfsmuster-Katalogs [GHJ+95] sind im Sinne von Beck und Johnson nicht generativ. Die Autoren nehmen aber an, durch einige Versuche gestützt, daß sich die Muster generativ formulieren lassen. Somit läßt sich vermuten, daß generative Muster nur eine Verschiebung der Perspektive auf Muster im Rahmen der Begründung einer spezifischen Softwarearchitektur sind.

3.5.4 Zusammenfassung

Die vorgestellten Autoren stimmen in folgenden Grundannahmen überein: Muster sind nie isoliert zu betrachten; Muster können einer Ordnungsrelation zufolge angeordnet werden, so daß sich ein azyklischer Graph ergibt; Muster lassen sich in beliebiger kombinatorischer Vielfalt anwenden. Das nächste Unterkapitel zieht aus diesen Beobachtungen sowie den vorausgegangenen Definitionen die Konsequenzen.

3.6 Ein Anordnungsschema für Muster

Ein Muster steht selten allein. Im Gegenteil, Muster stützen sich oft auf andere Muster ab. Sie sind somit immer im Zusammenhang zu sehen. Dies macht ein Anordnungsschema für voneinander abhängige Muster notwendig. Ein solches Schema sollte nie nur formal, sondern immer auch pragmatisch ausgerichtet sein.

3.6.1 Anordnung im Graphen

Zentraler Ausgangspunkt, um Muster sinnvoll anordnen zu können, ist die Betrachtung des Kontexts eines Musters.

Um ein Muster anwenden zu können, muß man seinen Kontext verstehen. Um den Kontextabschnitt eines Musters verstehen zu können, muß man wiederum dessen Kontext kennen. Aus dieser Beobachtung ergibt sich im Einklang mit der vorhergehenden Literatur eine Hierarchie von Mustern, von denen jedes nur dann verständlich wird, wenn man das in der Hierarchie vorausgehende Muster kennt. Offen bleibt nur die Frage, ob der resultierende Graph azyklisch ist und über ein erstes Muster verfügt, von dem aus allen anderen erreichbar sind.

Das im folgenden vorgestellte Schema ordnet Muster in einem gerichteten Graphen an: Die Ordnungsrelation basiert auf der intuitiven Ordnung zwischen den Musterkontexten. Der Graph muß nicht azyklisch sein, da zyklische Begriffsbildung nicht immer zu vermeiden ist und auch erwünscht sein kann. Somit muß es nicht zwingend ein erstes Muster geben. Die Erläuterung des Hintergrunds der Muster wird an den Anfang gestellt und bildet den rekursiven Kontextabschluß. Die Darstellung von Mustern wird durch Übersichten und Beispiele ergänzt.

Anordnung von Mustern im Graphen. Muster werden als Knoten in einem gerichteten Graphen angeordnet. Im Text wird dieser Graph so sequentialisiert, das jedes Muster, welches zum Verständnis des Kontexts eines weiteren Musters benötigt wird, vor diesem gelesen wird. Da zyklische Begriffsbildung durchaus üblich ist, kann nicht prinzipiell davon ausgegangen werden, daß dies möglich ist. Zyklen müssen pragmatisch gelöst werden, so z.B. durch Vorwärtsreferenzen. Ein Muster verweist dazu auf gleichwohl einbettende wie auch eingebettete Muster. Es stellt sich unmittelbar die Frage nach der Ordnungsrelation des Graphen.

Ordnungsrelation des Mustergraphen. Die Ordnungsrelation des Graphen basiert auf dem Umfang des Kontexts eines Musters. Ist der Kontext eines Musters umfassender als der Kontext eines anderen Musters, wird also das erste Muster zum Verständnis des zweiten Musters benötigt, so erscheint es vor ihm im Graphen. Hierbei kommen die unterschiedlichen Musterarten Interpretations- und Gestaltungsmuster, Entwurfsmuster und Programmiermuster ins Spiel: Interpretations- und Gestaltungsmuster sind „größer“ als Entwurfsmuster, welche wiederum „größer“ als Programmiermuster sind. Interpretationsmuster betten Softwaresysteme in menschliche Zusammenhänge ein: Deswegen sind entweder unabhängig von Entwurfsmustern oder aber erscheinen vor ihnen.

Der Hintergrund für alle Muster steht am Anfang. Vor der Präsentation des ersten zu erläuternden Musters muß der Kontext aller Muster geklärt werden. Dieser übergeordnete Kontext erläutert das Leitbild und die grundsätzlichen Annahmen, auf denen die im Anschluß dargestellten Muster basieren. Die Darstellung muß präzise und möglichst umfassend sein und den Kontext in alle Richtungen hin abstecken. Dies ist nicht im Kontextabschnitt der Muster selbst und auch nicht durch ein erstes exponiertes Muster zu leisten: Auch der Kontext eines Musters muß wieder vor einem Kontext gesehen werden. Um den infiniten Regreß zu vermeiden, wird deswegen vor allen Mustern ein allgemeiner in Prosa verfaßter Hintergrund geschildert. Er erläutert den Anwendungsbereich und das hinter den Mustern stehende Leitbild.

Überblick und Beispiele. Da Mustern in der Softwaretechnik eine große pragmatische Bedeutung zukommen soll, müssen sie entsprechend gut geschrieben sein. Zur Verbesserung der Verständlichkeit und Orientierung bietet es sich an, an exponierten Stellen einen Überblick über die für den momentanen Kontext relevanten Muster zu geben. Weiterhin hat es sich als sinnvoll erwiesen, ein durchgängiges Beispiel zu verwenden.

Mit dem gewonnenen Verständnis lassen sich Muster herausarbeiten, in eine Form bringen und untereinander anordnen. Dies dient der Dokumentation von Interpretations-, Gestaltungs- und Entwurfserfahrung. Die Betonung der Kontextproblematik und Anordnung im Graphen dient dazu, unerfahrenen Lesern zu helfen. Kennt sich jemand mit dem Hintergrund und der Anwendungsdomäne gut aus, so kann er oder sie unproblematisch einzelne Muster herausgreifen und lesen, ohne notwendig den vorausgegangenen Kontext noch einmal aufschlagen zu müssen.

3.6.2 Handbuchcharakter der angeordneten Muster

Die nächsten drei Kapitel schildern die Muster für die Werkzeug und Material Metapher. Sie sind gemäß des vorgestellten Ordnungsschemas angeordnet. Es stellt sich die Frage, ob es sich bei den drei Kapiteln im Prinzip um ein Handbuch handelt. Der Brockhaus/Wahrig schreibt zum Begriff Handbuch:

> **Handbuch [...]**
>
> ein oder mehrbändiges Druckwerk in handlicher Form, das den grundlegenden Stoff eines Wissensgebietes in übersichtlicher Form enthält; [...] [BW81, Bd. 2]

Andere Wörterbücher und Enzyklopädien geben ebenfalls keine differenziertere Auskunft. Sieht man also von Formalien ab, so lassen sich die nächsten drei Kapitel durchaus als Handbuch begreifen. Dies gilt natürlich nur für erfahrene Benutzer, da der unerfahrene Leser den Text linear wird lesen müssen. Erfahrene Benutzer aber können einzelne Muster aufschlagen und die entsprechende Diskussion nachlesen, da sie über genügend Kontextwissen verfügen.

Zwar *kann* man die nächsten Kapitel als ein (sehr kleines) Handbuch zum „Wissensgebiet“ Werkzeug und Material Metapher begreifen, die Frage aber bleibt, ob man das *will*. Da die

Diskussion zum Thema (Softwarearchitektur-)Handbücher noch nicht abgeschlossen ist, wird die Frage an dieser Stelle nicht beantwortet und auch nicht weiter verfolgt werden.

4 Die Werkzeug und Material Dichotomie

Mit diesem Kapitel beginnen die Muster für die Werkzeug und Material Metapher. Es gibt insgesamt drei Kapitel, welche jeweils eine abgegrenzte Menge von Mustern behandeln. In diesem Kapitel wird das Leitbild des Arbeitsplatzes für qualifizierte menschliche Arbeitstätigkeiten vorgestellt. Es wird von den Interpretations- und Gestaltungsmustern Werkzeug, Material, Verwendungszusammenhang und Umgebung gefolgt, welche es konkretisieren. Das Leitbild legt den Diskursbereich fest, in dem alle vorgestellten Muster zu sehen sind. Es beschreibt den Anwendungsbereich der Muster und verleiht ihnen einen über die technische Konstruktion hinausgehenden Sinn. Die nächsten Kapitel stellen objektorientierte Entwurfsmuster zur detaillierten softwaretechnischen Umsetzung der in diesem Kapitel geschilderten Interpretations- und Gestaltungsmuster vor.

4.1 Überblick über die Muster

Die Softwareentwicklung nach der Werkzeug und Material Metapher basiert auf dem Leitbild *des Arbeitsplatzes für qualifizierte menschliche Arbeitstätigkeiten*. Software wird entwickelt, um diese Tätigkeiten zu unterstützen. Das Leitbild bildet den Kontext für die vorgestellten Interpretations- und Gestaltungsmuster und legt gleichermaßen den Anwendungs- wie auch Diskursbereich des hier vertretenen Ansatzes fest.

Das Leitbild ergibt sich im Kontext aktueller volks- und betriebswirtschaftlicher Diskussionen, welche flexible und leistungsfähige Softwaresysteme verlangen. Diese Systeme sollen die Qualifikation von Benutzern zum Tragen kommen lassen und von ihnen verantwortlich eingesetzt werden können. Sie müssen in ihren Auswirkungen durchschaubar und anpaßbar sein. Die betrachteten Arbeitsplätze und -tätigkeiten sind werkstattartige Umgebungen und Schreibtische in Büros.

In der Ausarbeitung dieses Leitbildes ergibt sich folgende Grundaussage der Werkzeug und Material Metapher: Interpretiere die Anwendungswelt und gestalte die Software durch die Unterscheidung von *Werkzeugen* und *Materialien* und mache ihren *Verwendungszusammenhang* im Rahmen von Arbeitstätigkeiten explizit.

Werkzeuge sind Arbeitsmittel, die zur Bearbeitung von Materialien, den Arbeitsgegenständen, verwendet werden. Der Verwendungszusammenhang eines bestimmten Werkzeugs mit einem bestimmten Material wird mit Hilfe von *Aspekten* beschrieben. Ein Aspekt beschreibt die für eine Arbeitstätigkeit notwendige Funktionalität zur Bearbeitung eines Materials. Er stellt somit eine Perspektive auf ein Material dar, welche Benutzern über das verwendete Werkzeug vermittelt wird. Die Organisation von Werkzeugen und Materialien sowie ihre Zusammenführung wird durch die Ausarbeitung des Umgebungsmusters beschrieben.

Die Muster stellen keine Konzepte zur Modellierung von Arbeitstätigkeiten jenseits des einzelnen Arbeitsplatz zur Verfügung. Computerunterstützte kooperative Arbeit wird nicht themati-

siert. Die konzeptuelle und technische Anbindung der Softwarearbeitsumgebung an Datenbanken oder Informationsdienste wird im Abschnitt über Werkzeugintegration nur zur Abgrenzung behandelt. Ihre detaillierte Ausarbeitung bleibt weiteren Arbeiten vorbehalten.

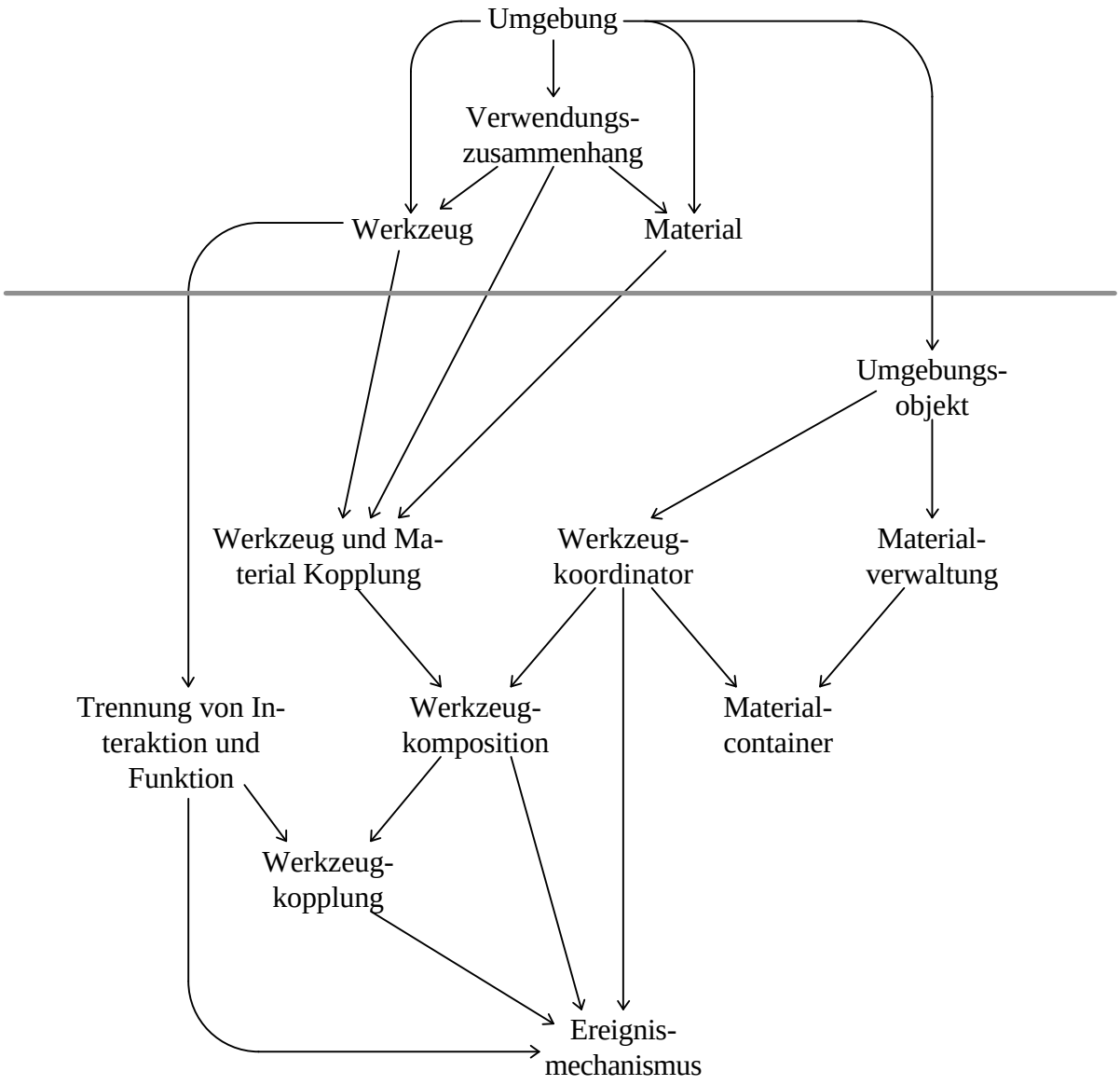


Abb. 4-1: Alle Muster im Überblick. Er faßt die Abb. 4-2, 4-3 und 4-4 zu einem gemeinsamen Schaubild zusammen. Dieses Kapitel erläutert das erste Muster sowie die Interpretations- und Gestaltungsmuster.

Abb. 4-1 gibt einen Gesamtüberblick, welcher drei verschiedene Ebenen aufzeigt. Zuoberst steht das Leitbild, welches den Kontext für die angeführten Muster erläutert. Darunter folgen die Interpretations- und Gestaltungsmuster. Sie werden von den anschließenden objektorientierten Entwurfsmustern in eine konkrete Softwarearchitektur umgesetzt. Ein Pfeil zwischen zwei Mustern bedeutet, daß sich das erste Muster, von dem der Pfeil ausgeht, in seiner Detaillierung und Realisierung auf das zweite Muster abstützt.

Abb. 4-2 bis 4-4 stellen die Zerlegung der Muster in drei Kapitel dar, wie sie in dieser Arbeit vorgenommen wurde. Dieses Kapitel erläutert das Leitbild und die Interpretations- und Gestaltungsmuster für die Werkzeug und Material Metapher, das nächste Kapitel erläutert die objektorientierten Muster für die Konstruktion von Softwarewerkzeugen und ihren Zusammenhang mit Materialien, und das übernächste Kapitel schildert die Integration mehrerer logisch voneinander abhängiger Werkzeuge und Materialien innerhalb einer Umgebung, hier dem einzelnen Arbeitsplatz.

Abb. 4-2: Leitbild und Interpretationsmuster für die Werkzeug und Material Metapher. Sie werden in diesem Kapitel geschildert.

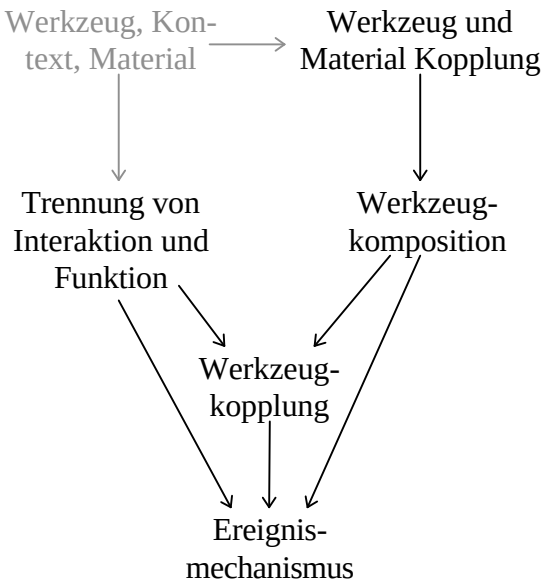


Abb. 4-3: Die Entwurfsmuster für die Konstruktion von Softwarewerkzeugen. Sie werden in Kapitel 5 vorgestellt.

Ist in einer Abbildung ein Schriftzug hellgrau gezeichnet, so bildet er den Anknüpfungspunkt zu einem anderen Teil der Muster und wird im zugehörigen Kapitel erläutert.

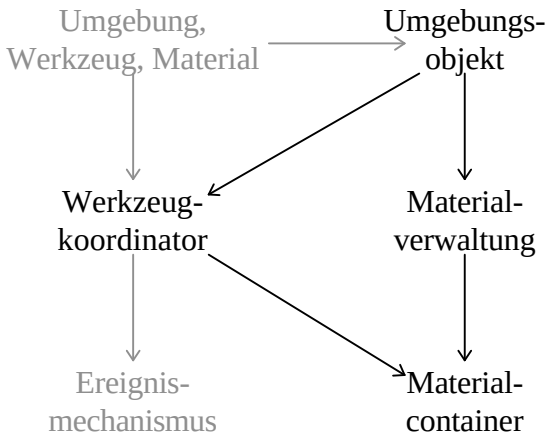


Abb. 4-4: Die Entwurfsmuster von Kapitel 6 ermöglichen die Integration von Werkzeugen und logisch abhängigen Materialien.

4.2 Der Arbeitsplatz für qualifizierte menschliche Arbeit

Dieser Abschnitt schildert das Leitbild des Arbeitsplatzes für qualifizierte menschliche Arbeitstätigkeiten. Es führt die Unterscheidung von *Werkzeugen* und *Materialien* ein, sowohl in der Interpretation der Anwendungswelt als auch der Gestaltung von Software. Diese Unterscheidung ermöglicht es, Software so zu konstruieren, daß sie fachlich qualifizierte Arbeitstätigkeiten effizient unterstützen hilft.

Ein großer Teil der aktuellen volks- und betriebswirtschaftlichen Diskussion dreht sich um neue Organisationsformen von Unternehmen. Die Segmentierung und Sequentialisierung von Arbeit im Sinne tayloristischer Arbeitsstrukturen wird heftig kritisiert und für viele Aufgabenbereiche als nicht mehr adäquat betrachtet [War92, Brö94, Woh94].

Der Dschungel diverser Schlagworte wie „fraktale Fabrik“, „virtuelles Unternehmen“, „Hierarchieverflachung“ und „lernende Organisation“ läßt nur einen begrenzten Konsens aufscheinen, was für die zukünftige Ausrichtung von Unternehmens(re)organisationen entscheidend sein wird. Zu diesem Grundkonsens gehört die Erkenntnis, daß es nie möglich sein wird, Organisationsstrukturen formal und vollständig beschreiben zu können. Im Gegenteil: Viele organisatorische Strukturen werden nur noch von begrenzter Dauer sein und den ständigen Wechsel, die ständige Veränderung zur Regel werden lassen [CP94].

Als Konsequenz dieser Erkenntnis folgt für viele Autoren, wieder die Menschen in den Mittelpunkt ihrer Arbeit zu stellen und ihre Flexibilität und Kreativität zur Geltung kommen zu lassen. Im Rahmen von Organisationskonzepten zur kooperativen Arbeit wird diskutiert, wie voller Einsatz der vorhandenen Qualifikationen, Dezentralisierung von Entscheidungskompetenz und Verantwortung sowie flexible Reaktion auf den stetigen Wandel von Kundenwünschen, Markt und Konkurrenz organisatorisch verankert und umgesetzt werden kann.

Dem Einsatz von Kommunikations- und Informationstechnologie wird hierbei eine bedeutende Katalysatorfunktion beigemessen. Sie kann gleichermaßen die Etablierung neuer Organisationsstrukturen beschleunigen und den beständigen Wandel ermöglichen wie auch ihn effektiv behindern und Umstrukturierungsbemühungen zum Scheitern bringen.

Aus den vielfältigen Konsequenzen dieser Betrachtungen ergeben sich verschiedene Anforderungen an Softwaresysteme, denen eine unterstützende Rolle im stetigen Wandlungsprozeß zukommen soll:

- Software soll die Qualifikation der Anwender zum Tragen kommen lassen und nicht zu ihrer Dequalifikation führen.
- Software soll die Weiterentwicklung von Kompetenz unterstützen, um das fachliche Wissen der Anwender nicht eines Tages obsolet werden zu lassen.
- Der Einsatz von Software muß transparent sein, in den Auswirkungen durchschaubar, um ihren verantwortungsvollen Einsatz zu ermöglichen.
- Soweit wie möglich muß auch auf den Einsatz von Software verzichtet werden können, sollte es die Situation einmal erfordern.
- Beim Einsatz von Software müssen die fachlichen Konzepte erkennbar, transparent und veränderbar sein, so daß Anwender ihre Qualifikation durch die Benutzung in das System hineintragen können.
- Anwender sollen die Software wechselnden Situationen entsprechend adäquat einsetzen können; die Software darf also nicht Arbeitsprozesse unnötig sequentialisieren.
- Die Software muß Anwendern die Möglichkeit geben, ihre Arbeitsumgebung selbst ein-

zurichten und sie wechselnden Gegebenheiten anpassen zu können.

Die Erfahrung zeigt, daß die folgenden Annahmen diese Ziele erreichen helfen: Interpretiere den Anwendungsbereich und konstruiere die Software anhand der *Unterscheidung von Arbeitsmitteln und Arbeitsgegenständen*, den *Werkzeugen* und *Materialien* respektive. Konstruiere Softwarewerkzeuge und Materialien so, daß sie *die fachlichen Konzepte und Gegenstände repräsentieren*, in ihrer Handhabung die *Entfaltung der Qualifikation* der Benutzenden ermöglichen und *keine Arbeitsabläufe festschreiben*. Entwerfe sie so, daß das Zusammenführen eines Werkzeugs mit einem Material den intendierten *fachlichen Arbeitszusammenhang* widerspiegelt.

Werkzeuge sind Arbeitsmittel, die eingesetzt werden um Materialien, die Arbeitsgegenstände zu bearbeiten. Materialien stellen Arbeitsergebnisse dar. Ein Bleistift ist ein Werkzeug, daß auf dem Material Papier arbeitet. Durch die Unterscheidung von Werkzeugen und Materialien wird die Arbeitssituation für Anwendende transparent und ermöglicht die erwünschte flexible Handhabung.

Die flexible Kombination eines Werkzeugs mit einem Material ist nicht beliebig, sondern muß einem fachlichen Arbeitszusammenhang entsprechen, dem *Verwendungszusammenhang*. Ein Werkzeug wird mit einem Material für eine bestimmte Arbeitstätigkeit zusammengeführt. Der resultierende Verwendungszusammenhang von Werkzeug und Material läßt sich durch *Aspekte* beschreiben, welche ein Material einem Werkzeug zur Nutzung anbietet. Ein Aspekt stellt eine Perspektive auf ein Material dar und beschreibt die Funktionalität für unter dieser Perspektive vorgenommene Arbeitstätigkeiten.

Das Schreiben als Verwendungszusammenhang stützt sich auf den Aspekt der Beschreibbarkeit ab. Das zum Schreiben verwendete Werkzeug Bleistift nutzt den vom Material angebotenen Aspekt der Beschreibbarkeit zur Ausführung dieser Arbeitstätigkeit.

Durch die Unterscheidung von Werkzeug- und Materialobjekten und die Herausarbeitung ihrer möglichen Kombinationen als Verwendungszusammenhänge wird die Anwendungssituation durchschaubar. Anwendende können klare Zuordnungen von Verantwortlichkeiten zu ihren Softwareobjekten machen und sie flexibel miteinander kombinieren.

Der Verwendungszusammenhang bestimmt, was ein Materialobjekt und was ein Werkzeugobjekt ist. So ist es möglich, daß ein Werkzeug als Material für ein anderes Werkzeug dient. Ein Bleistift ist zwar ein Werkzeug, wenn es ein Papier beschreibt, aber ein Material, wenn es von einem Anspitzerwerkzeug angespitzt wird. Die Kombination ist allerdings nicht beliebig, da Werkzeuge immer auf Materialien arbeiten und nicht umgekehrt.

Die Organisation von Werkzeugen und Materialien, z.B. auf einem Schreibtisch, wird im Rahmen einer *Umgebung* vorgenommen. Die Umgebung beschreibt räumliche und logische Dimensionen zur Organisation und Strukturierung von andauernden Arbeitstätigkeiten. Sie erlaubt erst die Zusammenführung von Werkzeugen und Materialien für bestimmte Arbeitstätigkeiten und prüft, ob der intendierte Verwendungszusammenhang überhaupt möglich ist, d.h. ob ein Material die von einem Werkzeug geforderten Aspekte überhaupt besitzt. Es macht keinen Sinn, mit einem Anspitzer als Werkzeug ein Material, das nicht angespitzt werden kann, zu bearbeiten.

Aus dieser Betrachtung heraus ergeben sich die Muster Umgebung, Verwendungszusammenhang, Werkzeug und Material. Sie werden in den nächsten Kapitelabschnitten herausgearbeitet.

Zur Erläuterung der Konzepte wird das Beispiel eines Terminkalenders und eines Wochenplaners verwendet. Ein Terminkalender ermöglicht die Eingabe von (einmaligen) Terminen, erlaubt, sie zu kommentieren und listet sie auf. Ein Wochenplaner hält wöchentlich wiederkehrende Termine fest, wie z.B. Seminare oder regelmäßige Treffen. Abb. 4-5 zeigt die Softwareversion des Terminkalenders, Abb. 4-6 den Wochenplaner. Der Entwurf des Kalenders basiert

auf handelsüblichen kleinen Kalendern, der Entwurf des Wochenplaners auf dem aus Schule und Universität bekannten Stundenplan.

Abb. 4-5: Der Terminkalender. Links sind die eingegebenen Termine aufgelistet, rechts die Kommentierung eines ausgewählten Termins.

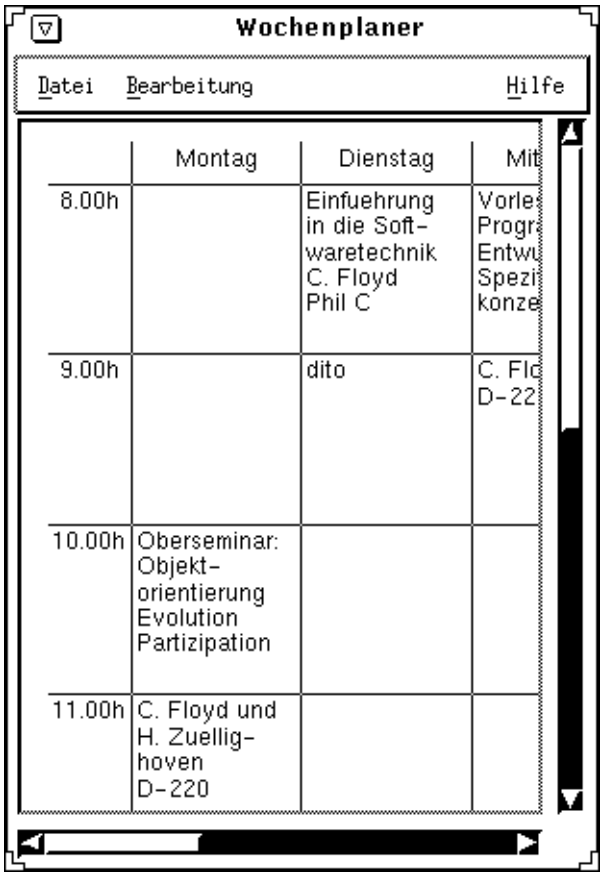


Abb. 4-6: Der Wochenplaner zeigt einen dem Stundenplan nachempfundenen Überblick wöchentlicher Termine an.

Anhand des Terminkalenders wird erläutert, wie ein Werkzeug konzeptuell entworfen und softwaretechnisch gebaut werden kann (dieses und nächstes Kapitel). Mit Hilfe des später hinzukommenden Wochenplaners werden Integrationsprobleme voneinander abhängiger Werkzeuge und Materialien erläutert (Kapitel 6).

Abb. 4-7 zeigt ein vorläufiges Modell der beiden Werkzeuge sowie ihre Einbettung in die Umgebung. Der Wochenplaner wurde nicht ausdetailliert, weil er für die Betrachtung der Konstruktion eines einzelnen Werkzeugs nicht herangezogen wird. Er erhält seine Bedeutung bei den objektorientierten Entwurfsmustern zur Werkzeugintegration.

In Abb. 4-7 erkennbar sind die vier Ebenen Umgebung, Werkzeuge, Aspekte und Materialien. Zuoberst steht die Umgebung, welche alle weiteren Objekte einbettet. Dies sind die Werkzeuge Kalender und Wochenplaner, sowie die Materialien indizierbares Terminbuch und Einzel- sowie regelmäßiger Termin. Das Terminbuch ist eine Liste von Einzelterminen. Der Kalender arbeitet auf den Materialien Terminbuch und Einzeltermin und verwendet dabei ihre Aspekte Indizierbar (TerminBuch) sowie Auflistbar und Editierbar (EinzelTermin).

Die Aspekte haben sich aus der Analyse des Verwendungszusammenhangs des Kalenders mit dem Terminbuch sowie den Einzelterminen selbst ergeben. Einzeltermine werden aufgelistet und editiert. Sie werden vom Kalender in einer Liste, aus der ausgewählt werden kann und die somit indizierbar ist, angezeigt.

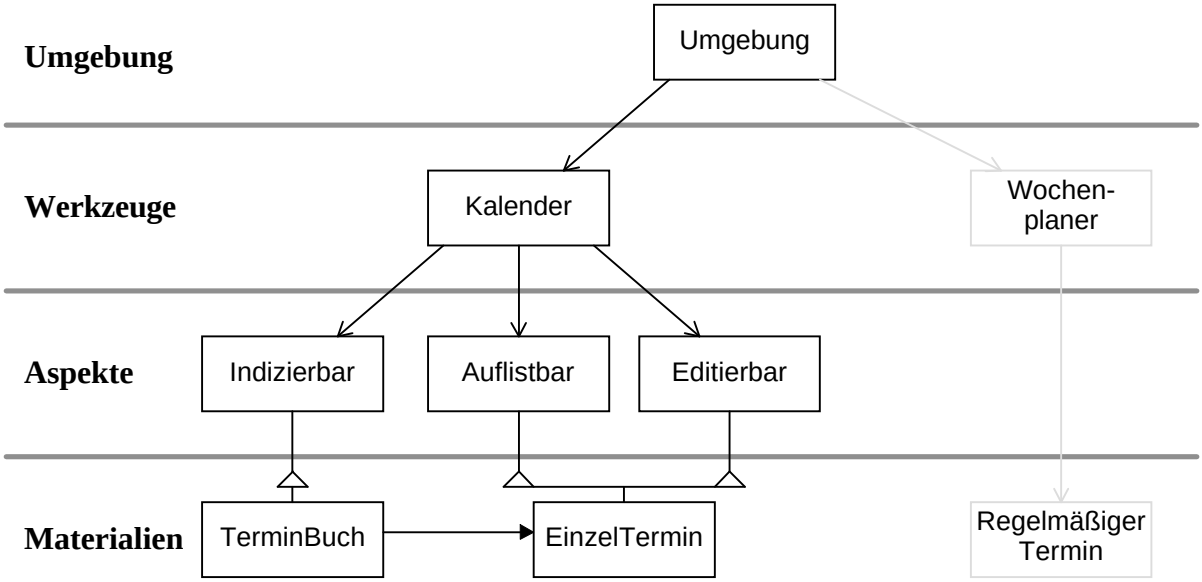


Abb. 4-7: Ein grobes vorläufiges Klassendiagramm des zu entwerfenden Systems. Es wird durch die Muster erklärt werden.

Die im folgenden beschriebenen Interpretations- und Gestaltungsmuster werden auf die ihnen entsprechende Ebene genauer eingehen und somit das Beispiel detailliert erläutern.

Dieses Leitbild besitzt verschiedene Anknüpfungspunkte zu anderen expliziten oder impliziten Leitbildern. Die meisten aktuellen graphischen Benutzungsschnittstellen sind von der Metapher der direkten Manipulation geprägt. Benutzenden wird der Eindruck vermittelt, direkt auf ihre Arbeitsergebnisse zuzugreifen, sie zu manipulieren [Maa94].

Dabei steht zumeist das Dokument (oder ein vergleichbares Konzept) im Mittelpunkt, zu dessen Bearbeitung eine vom System abhängige Funktionalität angeboten wird. Diese Betrachtung kommt dem hier vertretenen Ansatz entgegen, da sie ebenfalls auf qualifizierte Arbeitstätigkeiten hin ausgerichtet ist und Benutzende weitgehend selbstbestimmt arbeiten läßt.

Die Metapher der direkten Manipulation verfügt allerdings nicht über die hier vorgenommene Unterscheidung von Werkzeugen, welche auf Dokumenten, d.h. den Materialien, arbeiten. Zudem existiert auch in den meisten Systemen kein Begriff des Verwendungszusammenhangs oder der Aspekte.

Dadurch, daß die Begriffe Werkzeug und Verwendungszusammenhang nicht explizit sind, ergeben sich konzeptuell und softwaretechnisch problematische Situationen. Weder kann ein Werkzeug- und Bearbeitungszustand (siehe unten) vom Dokument unabhängig betrachtet werden, noch kann der aktuelle Verwendungszusammenhang explizit gemacht werden. Dies macht ad hoc Lösungen notwendig, welche Benutzer leicht verwirren können. Software, welche „nur“ der Metapher direkter Manipulation folgt, verfügt üblicherweise nicht über die angestrebte flexible Kombinierbarkeit von Werkzeugen und Materialien.

Trotz der Zentriertheit auf einen Dokumentbegriff haben deswegen viele erfolgreiche Anwendungsrahmen den Begriff der vom Dokument getrennten Applikation eingeführt, so z.B. MacApp [App86, App89], ET++ [Wei92, Gam92, WG94] und auch Microsoft Windows [OM95, MO94]. Gerade aber Windows, welches Applikationen als „Behälter“ für Dokumente betrachtet, macht die Notwendigkeit einer höheren Betrachtungsebene und fortgeschrittenen Begriffsbildung unmittelbar deutlich.

4.3 Umgebung

Werkzeuge bearbeiten und verwalten Materialien. Werkzeuge und Materialien befinden sich immer in einer Umgebung. Die Umgebung definiert die zur Organisation von Werkzeugen und Materialien verwendbaren räumlichen und logischen Dimensionen sowie zwischen ihnen geltende Konsistenzbedingungen. Sie stellt für den hier betrachteten einzelnen Arbeitsplatz eine äußere Hülle dar und schirmt die verwendeten Werkzeuge und Materialien nach außen hin ab.

Problem Benutzer organisieren Werkzeuge und Materialien, nehmen sie auf, verwenden sie und legen sie wieder ab. Sie suchen nach Werkzeugen und Materialien und sie kombinieren sie miteinander. Zwischen Materialien müssen Konsistenzbedingungen etabliert und kontrolliert werden.

Kontext An jedem Arbeitsplatz gibt es eine Vielzahl von Werkzeugen und Materialien, welche für verschiedene und zeitweilig schnell wechselnde Arbeitstätigkeiten verwendet werden. Sie zu organisieren ist wichtiger Bestandteil qualifizierter menschlicher Arbeitstätigkeit und muß individuell möglich sein.

Um dies zu tun, verfügen Menschen über diverse historisch, professionell und persönlich gewachsene Werkzeuge, z.B. Ablagen und Ordner. Mit ihrer Hilfe setzen Menschen eine zunächst räumliche Anordnung in eine konzeptuelle Ordnung um, welche ihnen hilft, ihre Werkzeuge und Materialien zu verwalten, und ihren Arbeitstätigkeiten gemäß zu strukturieren.

Softwaresysteme besitzen keinen physikalischen Begriff von Raum. Somit fehlt ihnen die zentrale Grundlage, welche von Menschen zur Strukturierung ihres Arbeitsplatzes eingesetzt wird. Seine Organisation aber ist wesentlicher Bestandteil qualifizierter Arbeit. Deswegen:

Lösung Analysiere die zur Organisation des Arbeitsplatzes verwendeten logischen Dimensionen sowie die zwischen Werkzeugen und Materialien existierenden Abhängigkeiten. Setze das Resultat in Form der *Umgebung* um. Die Umgebung stellt die zur Organisation von Werkzeugen und Materialien verwendeten Dimensionen zur Verfügung.[7]

Der Umgebung kommen somit mehrere verschiedene aber zusammenhängende Aufgaben zu. Zum ersten muß sie eine der menschlichen Wahrnehmung angemessene Umsetzung räumlicher Organisationsformen in logische Dimensionen ermöglichen. Diese können dann in Software realisiert werden. Zum zweiten soll sie fachliche Abhängigkeiten zwischen Materialien sicherstellen und ihre Konsistenz wahren. Und zum dritten muß sie kontrollieren, ob die Anwendung eines bestimmten Werkzeugs auf ein Material überhaupt möglich ist.

Ersteres kann zum Beispiel mit Hilfe der Schreibtischmetapher geschehen. Werkzeuge und Materialien werden als Ikonen repräsentiert und durch direkte Manipulation miteinander in Beziehung gesetzt. Die Umgebung wird durch einen Schreibtisch repräsentiert, dessen Aufgabe es ist, Werkzeugen eine der menschlichen Wahrnehmung angemessene Präsentation der von ihnen realisierten logischen Dimensionen zu ermöglichen. Diese Aufgabe kann heutzutage im Softwareentwurf weitgehend an bereits vorhandene Systemsoftware delegiert werden.

Es ist festzuhalten, daß die meiste Organisationsarbeit ohnehin von Werkzeugen ausgeführt wird. So ist ein Ordner als ein Material zu interpretieren, welches mit Hilfe eines Ordnerbrowsers („Blätterers“) zur Verwaltung der in ihm abgehefteten Dokumente verwendet wird.

---

[7] Diese in folgenden dargestellten unterschiedlichen Aufgaben machen klar, daß es sich bei dem Umgebungsmuster um kein kohärentes Konzept handelt. Es stellt vielmehr die „Welt“ für die weiteren Muster dar und dient zur ihrer Abgrenzung nach außen. Um überhaupt beschreibbar zu sein, muß der Umfang des Umgebungsmusters drastisch reduziert werden. Ausgewählt wurden jene Eigenschaften, welche sich pragmatisch als relevant erwiesen haben.

Der Schreibtisch ermöglicht lediglich die grundlegende Funktionalität, die logische Dimension des Enthaltenseins in eine anschauliche grafische Repräsentation umzusetzen. Dies geschieht zum Beispiel durch Ikonen, welche in Fenstern verwaltet werden.

Die zweite Aufgabe, die Formulierung und Wahrung von Konsistenzbedingungen zwischen Werkzeugen und Materialien, kommt der Umgebung genau dann zu, wenn eine Konsistenzbedingung über ein einzelnes Werkzeug hinausgeht. Auf dem physischen Schreibtisch gibt es üblicherweise kein Pendant zu diesen Konsistenzbedingungen; Benutzende müssen diese zumeist enervierende Fleißaufgabe von Hand ausführen.

Als Beispiel für eine solche Bedingung wird in Kapitel 6 das Überlappungsproblem von Einzelterminen des Kalenders mit regelmäßigen Terminen des Wochenplaners angeführt werden. Wird die zeitliche Überlappungsfreiheit als Konsistenzbedingung formuliert, muß die Umgebung dafür sorgen, daß Materialien und Werkzeuge in einen solchen Zustand versetzt werden, daß das Problem Benutzenden angezeigt wird. Dem Verständnis qualifizierter Arbeit zufolge sollte dies nicht aufdringlich geschehen und auch nicht den Fortgang der Arbeit unterbrechen. Das Problem muß aber sichtbar sein und bis zu seiner Behebung oder bewußten Ignorierung auch angezeigt werden.

Da sich die hier definierte Umgebung auf einen einzelnen Arbeitsplatz beschränkt, gehen auch die Konsistenzbedingungen nicht über diesen hinaus. Sie gelten nur für Materialien, welche aktuell von Werkzeugen bearbeitet werden. Persistente, nicht aktivierte Materialien in Datenbanken unterliegen zwar oft gleichen oder leicht abgewandelten Konsistenzbedingungen, werden hier aber nicht erfaßt. Die Ausarbeitung und Abgrenzung der Umgebung wird in Kapitel 6 vorgenommen.

Die dritte Aufgabe, die Überprüfung, ob ein Werkzeug überhaupt auf ein Material anwendbar ist, wird von der Umgebung mit Hilfe des intendierten *Verwendungszusammenhangs* getroffen. Sie wird im folgenden Muster diskutiert.

4.4 Verwendungszusammenhang

Ein Werkzeug wird mit einem Material immer für eine bestimmte Arbeitstätigkeit zusammenführt. Diese Tätigkeit stellt den *Verwendungszusammenhang* dar, in dem ein Werkzeug mit einem Material verwendet wird. Ein *Verwendungszusammenhang* wird durch die verwendeten *Aspekte* formalisiert, welche ein Material zu seiner Bearbeitung anbietet. Ein Aspekt beschreibt die Funktionalität, welche unter der Perspektive einer bestimmten (Teil-)Tätigkeit benötigt wird, um die beabsichtigte Aufgabe auszuführen.

Problem Die Analyse und der Entwurf von Werkzeugen und Materialien muß den Zusammenhang zwischen ihnen klären, um zu beschreiben, ob sie miteinander kombiniert und wie sie miteinander kombiniert werden können.

Kontext Innerhalb einer Umgebung können Benutzende beliebig versuchen, Werkzeuge auf Materialien anzuwenden. Werkzeuge können aber üblicherweise nicht beliebige Materialien bearbeiten, sondern nur jene, für die sie entwickelt wurden. Es gibt hochspezialisierte Werkzeuge, wie z.B. einen Sechskantschlüssel, der nur auf ein bestimmtes Material, eine Schraube bestimmter Größe, paßt. Und es gibt sehr allgemeine Werkzeuge, so z.B. den Bleistift, der nahezu jedes Material beschreiben kann, das sich nur beschreiben läßt.

In der physikalischen Welt wird dieser Zusammenhang für Benutzende dadurch erfahrbar, daß ein Werkzeug auf ein bestimmtes Material „paßt“ oder eben nicht. In der softwarebasierten Umgebung muß dieses „Passen“ nachempfunden werden, um Benutzende nicht vor überraschende Ergebnisse zu stellen.

Da Werkzeuge zum Bearbeiten bestimmter Materialien entworfen werden, muß das Zusammenführen eines Werkzeugs mit einem Material der beabsichtigen Arbeitstätigkeit entsprechen. Dies ermöglicht ihre effiziente Ausführung und entspricht dem fachlichen Arbeitszusammenhang. Deswegen:

Lösung Analysiere die verschiedenen Arbeitstätigkeiten, welche auf einem Material ausgeführt werden. Erfasse jede Arbeitstätigkeit als *Verwendungszusammenhang* und zergliedere ihn in *Aspekte*, welche die Funktionalität für Teilarbeitstätigkeiten beschreiben. Ordne die gewonnenen Aspekte den Materialien zu und entwerfe Werkzeuge für bestimmte Arbeitstätigkeiten so, daß sie nur diese Aspekte verwenden.

Eine übliche Arbeitstätigkeit im Beispiel ist das Anlegen eines neuen Termins im Kalender. Der *Verwendungszusammenhang* ist das Arbeiten mit und Verwalten von Terminen. Verschiedene Einzeltätigkeiten treten hierbei auf:

- Ein neuer Termin wird erstellt. Dazu wird sein Datum, Titel sowie ein erläuternder Kommentar eingegeben. Datum und Titel sollen nur einmal eingegeben werden, der Kommentar hingegen soll beliebig änderbar sein. Das Ändern des Kommentars wird als Aspekt Editierbar beschrieben, welcher die Funktionalität textuellen Editierens umfaßt.
- Der Termin wird in das Terminbuch eingefügt. Dies führt zum Aspekt Auflistbar des Termins und Indizierbar des Terminbuchs. Der Termin muß auflistbar sein, um ins Terminbuch eingefügt werden zu können. Das Terminbuch muß indizierbar sein, um das Einfügen und Auslesens des Termins an einer bestimmten Stelle zu ermöglichen.

Im Rahmen eines *Verwendungszusammenhang*s formalisiert ein Aspekt eine Teiltätigkeit, in dem er die zur Ausführung dieser Tätigkeiten verwendbaren Operationen beschreibt. Das verwendete Ausdrucksmittel zur Formalisierung ist hierbei noch offengelassen.

Aspekte dienen zur Entkopplung von Werkzeugen und Materialien, da sich Werkzeuge nur auf bestimmte Aspekte zur Bearbeitung eines Materials abstützen müssen und nicht mehr auf das Material selbst. Die funktionale Definition eines Aspekts enthält genau jene Operationen, welche ein Werkzeug benötigt, um seine Aufgaben zu erfüllen. Das Material ist immer nur aus Sicht der aktuellen Tätigkeit interessant und bleibt hinter seinen Aspekten versteckt.

Aspekte stellen den Zugang eines Werkzeugs zu einem Material dar. Ein Werkzeug wird für jedes Material verwendbar, welches diese Aspekte anbietet. In der Umkehrung braucht ein Material nur eine bestimmte Anzahl von Aspekten zur Verfügung zu stellen, um von einem Werkzeug bearbeitet werden zu können.

Es ist wichtig zu betonen, daß hier die Begriffe des *Verwendungszusammenhangs*, von Arbeits- und von Teilarbeitstätigkeiten in einem sozialen Begriffsgefüge eingeordnet und auch so von Entwicklern und Benutzern diskutiert werden. Die Formalisierung des Verwendungszusammenhangs durch Aspekte hingegen ordnet den Aspekt in den Kontext des Softwareentwurfs und der formalen Spezifikation ein. Eine solche formale oder auch nur semi-formale Beschreibung besitzt nicht die Reichhaltigkeit und Ambiguitäten der sozialen Lebenswelt. Sie soll diese auch gar nicht widerspiegeln. Ein Aspekt beschreibt nur das, was als Ergebnis des Diskussionsprozesses formal notiert und im Softwareentwurf verwendet werden kann. Seine Abbildung in eine fachliche Bedeutung wird immer von Menschen vorgenommen.

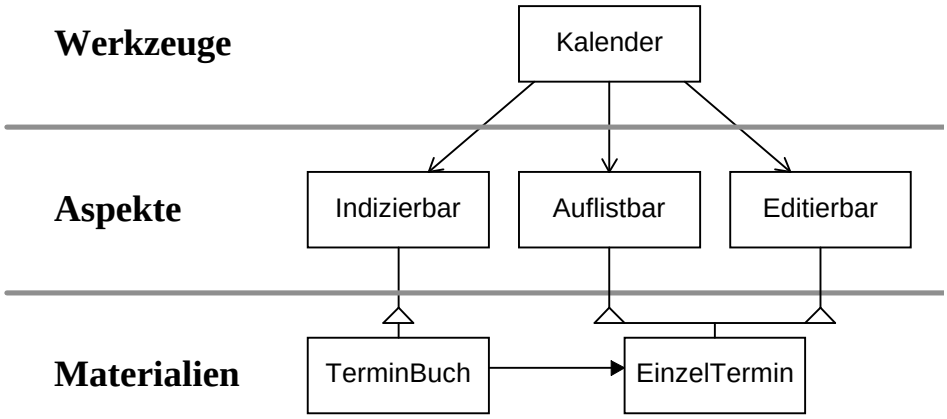


Abb. 4-8: Das Werkzeug Kalender nutzt die Aspekte Indizierbar, Auflistbar und Editierbar der Materialien TerminBuch und EinzelTermin, um auf ihnen zu arbeiten.

Die Umgebung ermittelt, ob ein Werkzeug auf einem Material arbeiten kann, indem sie die vom Werkzeug geforderten Aspekte mit den vom Material angebotenen Aspekten vergleicht. Sind die angebotenen Aspekte eine Obermenge der geforderten Aspekte, so kann das Werkzeug auf dem Material arbeiten. Ihre formale Definition erfahren Aspekte als *Aspektklassen* im objektorientierten Entwurfsmuster *Werkzeug und Material Kopplung* in Kapitel 5.

Vergleiche Dem Aspekt verwandte Konzepte tauchen in verschiedenen Varianten auf benutzungsorientierter oder softwaretechnischer Ebene auf. Arbeitspsychologen verwenden den Begriff der „Usability“, um die Brauchbarkeit eines Objekts im Rahmen einer Aufgabe zu beschreiben [Maa93, DWA93]. Auf softwaretechnischer Ebene wird mit „Traits“ [CA84, RS95] oder „Contracts“ [Mey91, HHG90, Hol92] gearbeitet, um die zwei oder mehr Objekte koppelnde Funktionalität zu formalisieren. Aspekte können technisch mit Hilfe von Traits oder Contracts umgesetzt werden, gehen aber auf der hier vorgestellten konzeptuellen Ebene des Verwendungszusammenhangs über sie hinaus.

4.5 Material

Materialien werden sondiert, manipuliert und in andere Materialien eingebettet. Sie sind Gegenstand und Ergebnis der Arbeitstätigkeiten. Während der Arbeit konzentrieren sich Anwender auf ihre Bearbeitung.

Problem Was sind Materialien und welche Eigenschaften besitzen sie? Wie werden sie bearbeitet und was für eine Rolle spielen sie im Arbeitsprozeß?

Kontext Die Werkzeug und Material Dichotomie sieht die Unterscheidung von Werkzeugen und Materialien vor. Die Betrachtung des *Verwendungszusammenhang*s von Materialien führt zu Aspekten, unter deren Nutzung Werkzeuge auf Materialien arbeiten können.

Zu jeder erfolgreich ausgeführten Arbeit gehören ein oder mehrere Arbeitsergebnisse. Arbeitsergebnisse sind Materialien nach abgeschlossener Bearbeitung. Ihre Gestaltung steht im Mittelpunkt einer jeden Arbeit. Deswegen:

Lösung In der Unterteilung der Objekte der Anwendungswelt in Werkzeuge und Materialien bestimme jene Objekte als Materialien, auf die sich die Gestaltung durch Benutzende und somit ihr Interesse und ihre Konzentration richtet. Untersuche, wie Materialien sondiert, manipuliert oder in andere Materialien eingebettet werden.

Bei der Bearbeitung von Terminen mit Hilfe des Kalenderwerkzeugs konzentrieren Benutzende sich auf die Termine und nicht auf den Kalender. Im Zentrum ihres Interesses stehen die Materialien und nicht die Werkzeuge, mit denen sie bearbeitet werden. Trotzdem wird der Kalender benötigt, da ohne ihn als Werkzeug die Bearbeitung nicht möglich wäre.

Materialien sind weitgehend passive Objekte, welche aufgenommen, bearbeitet und abgelegt werden. Während der Arbeit und der Konzentration auf Materialien entsteht der Eindruck, direkt auf sie zuzugreifen und sie zu manipulieren. Das Werkzeug, welches erst den Zugang und die Interaktion mit dem Material ermöglicht, tritt in den Hintergrund.

Obwohl der Zugang zu einem Material im Computer immer erst über Werkzeuge ermöglicht wird, existieren Materialien unabhängig von diesen. Die vom Kalender verwalteten Termine können auch von anderen Werkzeugen sondiert werden, so z.B. einem einfachen Auflisterwerkzeug, welcher lediglich das Terminbuch als Liste anzeigt.

Materialien werden, wie festgestellt, immer in bestimmten Verwendungszusammenhängen verwendet. Die für das Arbeiten in einem bestimmten Kontext benötigte Funktionalität wird mittels Aspekten beschrieben. Da Werkzeuge aus Gründen der Wiederverwendung nur auf Aspekten aufsetzen, sollte die Schnittstelle eines Materialobjekts ausschließlich aus Aspekten bestehen.

4.6 Werkzeug

Ein Werkzeug bearbeitet Materialien. Es präsentiert sie gemäß ihres Verwendungszusammenhangs und ermöglicht ihre Sondierung und Manipulation. Ein Werkzeug vergegenständlicht immer wiederkehrende Arbeitstätigkeiten, welche an Materialien ausgeführt werden.

Problem Was sind Werkzeuge und welche Eigenschaften besitzen sie? Wie werden sie verwendet und welche Rolle spielen sie im Arbeitsprozeß?

Kontext Zur Bearbeitung von Materialien werden Werkzeuge verwendet. Die Qualität der Arbeitsergebnisse hängt somit entscheidend von der Qualität der zu ihrer Erstellung verwendeten Werkzeuge ab. Diese wiederum hängt von Eigenschaften der Handhabbarkeit, Effizienz und Konsistenz des Werkzeugs ab. Deswegen:

Lösung Interpretiere und entwerfe jene Objekte als Werkzeugobjekte, welche zur Bearbeitung von Materialien verwendet werden, während der Benutzung aber in den Hintergrund treten. Interpretiere sie als vergegenständlichte Unterstützung für immer wiederkehrende aber leicht variierende Arbeitstätigkeiten.

Der Zugang zu Materialien wird ausschließlich über Werkzeuge vermittelt. Werkzeuge präsentieren und ermöglichen die Sondierung und Manipulation von Materialien auf werkzeugspezifische Art und Weise. Gute Werkzeuge sind während der Arbeit „zuhanden“, d.h. sie sind unauffällig und vertraut. Sie werden zwar nicht unsichtbar, treten aber in den Hintergrund.

Im Beispiel ist der Kalender ein Werkzeug, das auf den Materialien Termin und Terminbuch arbeitet. Während der Benutzung des Kalenders stehen die Materialien im Vordergrund. Benutzende konzentrieren sich auf sie und nehmen nur am Rande wahr, daß ein Werkzeug ihren getippten Kommentar an den bearbeiteten Termin weiterleitet.

Werkzeuge erwachsen aus der Erfahrung mit immer wiederkehrenden Tätigkeiten. Sie werden entworfen, um diese Tätigkeiten zu vereinfachen, nicht aber um sie zu automatisieren. Somit vergegenständlichen Werkzeuge die ausgeführten Tätigkeiten. Als Konsequenz besitzen Werkzeuge einen vom Material unabhängigen Werkzeug- und Bearbeitungszustand, welcher sich anhand der Anforderungen der jeweiligen Arbeitstätigkeiten ergibt.

Die Rückmeldungen, welche Benutzende von ihren Materialien erhalten, werden ebenfalls vom Werkzeug kontrolliert: Das Werkzeug entscheidet, welche der über Aspekte zugänglichen Zustandsänderungen aus Sicht des Verwendungszusammenhangs relevant und anzuzeigen sind. Diese Perspektive entspricht der von Benutzenden beabsichtigten Arbeitstätigkeit, welche zur Anwendung eines bestimmten Werkzeugs auf ein Material führte. Für genau diese Verwendungszusammenhänge werden Werkzeuge entworfen.

Vergleiche An Softwaresysteme wird oft der Anspruch gerichtet, daß sie Benutzenden wie ein Werkzeug zur Verfügung stehen sollen. Diese Werkzeugmetapher genannte Perspektive wird als Gegenentwurf zur Maschinen- oder Systemmetapher verwendet, welche Benutzende in eine Maschine einspannen oder zu Bedienern des Systems machen [MO92, Coy95].

Diese (namensgleiche) Metapher basiert wie der hier vertretene Ansatz auf der Voraussetzung, Softwareunterstützung für qualifizierte menschliche Arbeitstätigkeiten entwickeln zu wollen und gleicht ihr weitgehend. In der Literatur taucht sie meist nur abstrakter Form zur Beschreibung von Benutzungsschnittstellenqualitäten auf, während hier diese Qualitäten bis in den detaillierten Softwareentwurf hineingetragen werden. Zudem findet auf dieser abstrakten Ebene nicht die Unterscheidung zwischen Werkzeugen und Materialien statt.

5 Entwurfsmuster zur Werkzeugkonstruktion

In diesem Kapitel werden die objektorientierten Entwurfsmuster zur Konstruktion von Softwarewerkzeugen nach der Werkzeug und Material Metapher geschildert. Mit ihrer Hilfe setzen Entwickler die im vorigen Kapitel geschilderten Prinzipien zur Gestaltung von Werkzeugen und Materialien in einen softwaretechnischen Entwurf um. Zugrunde gelegt werden die Muster Verwendungszusammenhang, Material und Werkzeug. Die Entwurfsmuster greifen ihre Bedeutung auf, gehen aber nicht über den von ihnen aufgespannten Diskursbereich hinaus.

5.1 Überblick über die Muster zur Werkzeugkonstruktion

Der Verwendungszusammenhang klärt, was ein Werkzeug und was ein Material ist. Er wird durch Aspekte formalisiert, welche die zur Bearbeitung eines Materials benötigten Operationen beschreiben. Dies führt zum zentralen objektorientierten Entwurfsmuster für die Konstruktion einzelner Werkzeuge und Materialien, der *Werkzeug und Material Kopplung*: Jeder Aspekt wird durch eine eigene Klassenschnittstelle, die *Aspektklasse*, umgesetzt. Die Schnittstelle einer Materialklasse wird durch Mehrfachvererbung von Aspektklassen zusammengefügt.

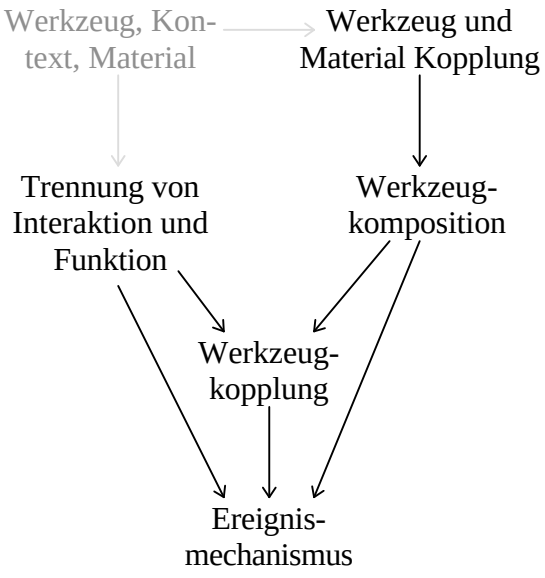


Abb. 5-1: Die Entwurfsmuster dieses Kapitels zur Konstruktion eines einzelnen Werkzeugs.

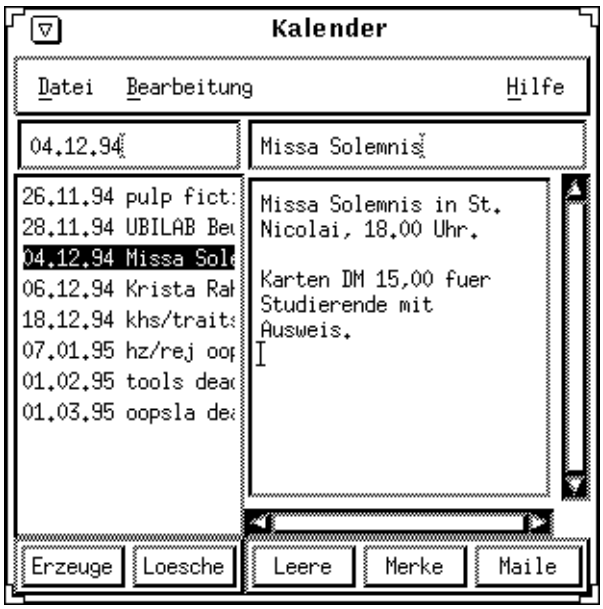


Abb. 5-2: Die Benutzungsschnittstelle des bereits eingeführten Kalenderbeispiels.

Vertikal werden Werkzeuge gemäß des Musters *Werkzeugkomposition* aufgebaut. Dabei bestehen *komplexe Werkzeuge* aus einem Baum von Werkzeugobjekten, welcher die Wiederver-

wendung einfacherer Werkzeuge als Teilbäume oder Blätter ermöglicht. Abb. 5-1 zeigt dieses und die anderen in diesem Kapitel vorgestellten Muster im Überblick.

Horizontal wird ein Werkzeug in einen Interaktions- und einen Funktionsteil aufgliedert, um die Benutzungsschnittstelle eines Werkzeugs von seinem funktionalen Teil zu trennen (dialog independence, separation of concerns). Das Muster *Trennung von Interaktion und Funktion* erläutert und setzt dieses Prinzip konstruktiv um.

Durch die horizontale und vertikale Aufteilung eines Werkzeugs entstehen Kopplungsprobleme, welche mit Hilfe des Musters der *Werkzeug Kopplung* gelöst werden. Die Kopplung von Werkzeugen mit Subwerkzeugen sowie des Interaktions- mit dem Funktionsteils stützt sich auf das Muster des *Ereignismechanismus* ab. Es dient zum softwaretechnisch sauberen Aufbau eine Benutzungshierarchie zwischen Objekten, ohne das für reaktive Systeme notwendige schnelle Feedback von unten nach oben oder außen aufzugeben.

Zur Erläuterung der Muster wird auf das bereits eingeführte Beispiel des Kalenders zurückgegriffen. Abb. 5-2 zeigt noch einmal seine Benutzungsoberfläche und Abb. 5-3 das vorläufige Objektmodel aus Kapitel 4.

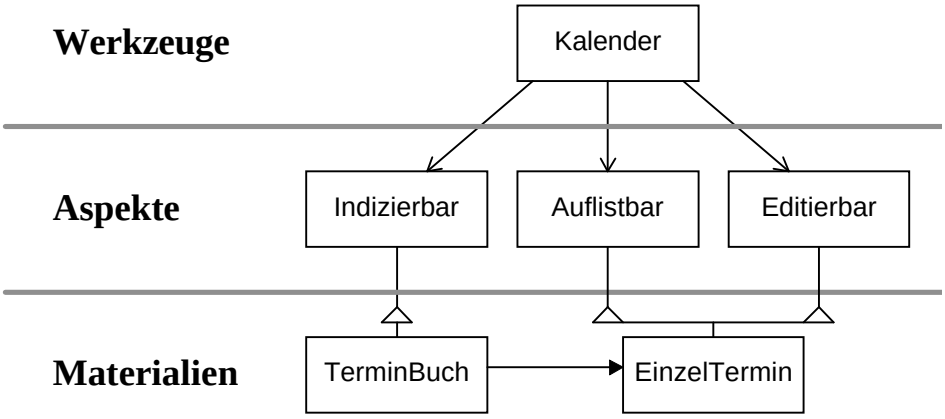


Abb. 5-3: Das Werkzeug Kalender nutzt die Aspekte Indizierbar, Auflistbar und Editierbar der Materialien Terminbuch und Termin, um auf ihnen zu arbeiten.

Im Laufe des Kapitels wird der Kalender in ein Kalenderwerkzeug zergliedert, welches sich eines generischen Auflisters und Editors als Unterwerkzeuge bedient. Diese beiden Werkzeuge arbeiten auf den Aspekten Auflistbar und Editierbar, welche als Aspektklassen Teile der Schnittstelle festlegen, die ein Termin als Material für den Kalender aufweisen muß.

5.2 Werkzeug und Material Kopplung

Werkzeuge werden über Aspekte mit Materialien gekoppelt. Im objektorientierten Entwurf wird ein Aspekt durch eine Klassenschnittstelle, die Aspektklasse, umgesetzt. Die Schnittstelle einer Materialklasse wird mittels Mehrfachvererbung aus Aspektklassen zusammengesetzt. Dies bestimmt, welche Werkzeuge auf ein Material angewendet werden können.

Problem Die Qualität vieler aus dem Handwerk bekannter Werkzeuge ist, vielseitig für mehrere Materialien einsetzbar zu sein. Im Softwareentwurf soll somit ein Werkzeug gleichermaßen auf mehreren Materialien arbeiten können, wie auch ein Material von mehreren Werkzeugen bearbeitet werden können soll.

Kontext Werkzeuge werden auf Materialien angewandt, um eine bestimmte Arbeitstätigkeit auszuführen. Diese wurde als intendierter Verwendungszusammenhang beschrieben,

welcher mit Hilfe von Aspekten formalisiert wurde. Ein Aspekt beschreibt die benötigte Funktionalität, welche die Ausführung einer im Rahmen eines Verwendungszusammenhangs anfallenden Teiltätigkeit ermöglichen soll. Die Art der Spezifikation eines Aspekts und ihre softwaretechnische Umsetzung war dabei offen geblieben.

Zur Vereinfachung der Diskussion werden zuerst nicht der Kalender, sondern zwei einfachere Werkzeuge betrachtet. Abb. 5-4 zeigt ein Auflisterwerkzeug, welches eine Liste von auflistbaren Materialien anzeigen kann. Abb. 5-5 zeigt ein Editorwerkzeug, welches ein beliebiges editierbares Material zu editieren erlaubt. Beide Werkzeuge präsentieren ihre Materialien und ermöglichen ihre Bearbeitung. Der Auflister arbeitet auf zwei verschiedenen Materialien, von denen das eine indizierbar (z.B. das Terminbuch) ist und das andere auflistbar (z.B. ein einzelner Termin). Der Editor arbeitet auf einem editerbaren Material (z.B. einem Terminobjekt). Die genannten Eigenschaften der Materialien wurden bereits als Aspekte Indizierbar, Auflistbar und Editierbar erläutert.

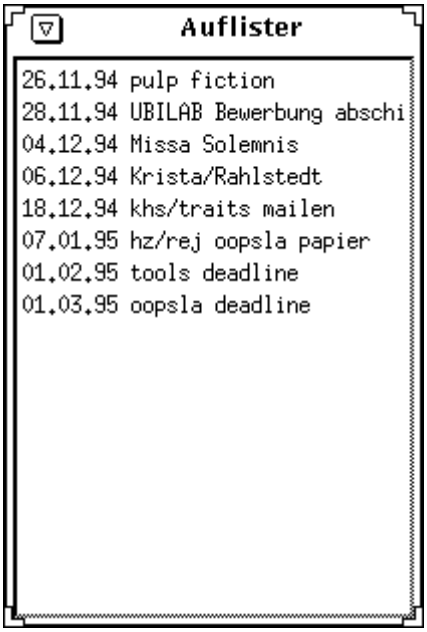


Abb. 5-4: Ein alleinstehendes Auflisterwerkzeug, welches eine Liste von Einträgen anzeigt.

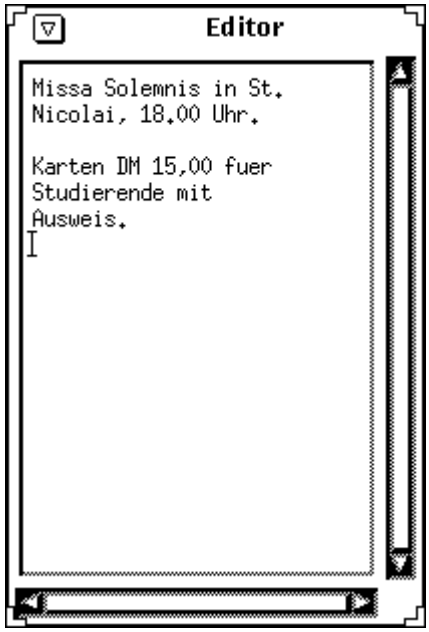


Abb. 5-5: Ein Editorwerkzeug, in dem ein Text eingegeben und editiert wurde.

Aspekte koppeln Werkzeuge mit ihren Materialien. Diese Kopplung muß formal notiert werden, um in einen Softwareentwurf umgesetzt werden zu werden. Sie soll dabei folgende Anforderungen erfüllen:

- Die Funktionalität eines Materials, welche durch den Aspekt beschrieben wird, soll aus Operationen zur Sondierung und Manipulation des Materials bestehen und somit die fachliche Handhabung von Materialien widerspiegeln.
- Soweit dies im Rahmen formaler Spezifikation sinnvoll machbar ist, soll die Semantik der Operationen formal beschrieben werden.
- Die Spezifikation soll programmiersprachlich ausgedrückt werden können, um eine möglichst große Nähe von Entwurf und Programmierung zu ermöglichen.

Zudem muß eine solcherart spezifizierte Schnittstelle softwaretechnisch einfach an Materialklassen anbindbar sein. Deswegen:

Lösung Beschreibe einen Aspekt als eine *Aspektklasse*, welche die möglichen Operationen zur Sondierung und Manipulation eines dahinterstehenden Materials sowie ihre Semantik spezifiziert. Bilde eine Materialklasse als Unterklasse aller Aspektklassen, welche Aspekte ausdrücken, die in den möglichen Verwendungszusammenhängen des Materials benötigt werden. Lasse Werkzeuge ausschließlich Aspektklassen benutzen, um auf Materialien zu arbeiten.

In der Analyse der Anwendungswelt werden Werkzeuge und Materialien in ihren Verwendungszusammenhängen betrachtet. Die Verwendungszusammenhänge bestimmen durch ihre Zergliederung in Teiltätigkeiten und deren Formalisierung durch Aspekte die Aspektklassen, welche die resultierende Klassenschnittstelle eines Materials bilden. Dadurch, daß Materialklassen von Aspektklassen erben, kann ein Materialobjekt von jedem Werkzeugobjekt verwendet werden, welches auf einer vom Material angebotenen Aspektklasse arbeitet.

Soll ein Material von einem Werkzeug aufgelistet werden können, so erbt es von der *Aspektklasse Auflistbar*, soll es editiert werden können, erbt es von *Editierbar*. Die Aspektklassen Auflistbar und Editierbar bieten genau jene Funktionalität, welche zum Auflisten respektive Editieren von Materialien benötigt wird und nicht mehr. Somit bilden Aspektklassen die kleinste sinnvoll wiederverwendbare Klassenschnittstelle. Abb. 5-6 zeigt das resultierende Klassendiagramm für den Auflister und Editor und Abb. 5-7 zeigt die C++ Klassenschnittstelle der Aspektklassen Auflistbar und Editierbar.

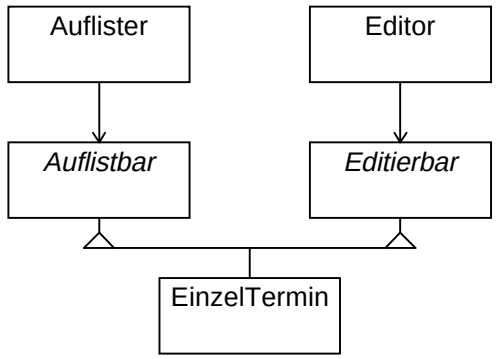


Abb. 5-6: Die Materialklasse EinzelTermin erbt von Auflistbar und Editierbar, welche von den jeweiligen Werkzeugen benutzt werden.

```
class Auflistbar
{
  public:
    String& GibEintragNamen() =0;
};


class Editierbar
{
  public:
    String& GibAbsatz() =0;
    void SetzeAbsatz( String& ) =0;
    void SucheString( String& ) =0;
};
```

Abb. 5-7: Die (etwas vereinfachten) Aspektklassen Auflistbar und Editierbar als C++ Klassenschnittstellen.

Die Angabe einer Klassenschnittstelle stellt nur eine sehr einfache Form der Formalisierung dar. Eine sinnvolle Form der Spezifikation, welche zugleich in Programmiersprachen eingebettet werden kann, sind Vor- und Nachbedingungen, Klasseninvarianten und zeitabhängigen Eigenschaften („history properties“). Möglichkeiten der Spezifikation für objektorientierte Modellierung unter Berücksichtigung von Vererbung respektive Subtyping geben [DT92, LW93, LW93a, PS94]. Die Umsetzung in Programmiersprachen ermöglicht z.B. Eiffel durch das Vertragsmodell („Design By Contract“) [Mey91, Mey92, Mey92a].

Aspektklassen sind definiert als abstrakte Klassen, die nur Funktionalität definieren, ohne sie durch eine Implementation einzulösen. Die Funktionalität wird erst von konkreten erbenden Materialklassen implementiert. Der durch sie formulierte Vertrag zwischen Werkzeug und Material wird und soll von jedem hinter einer Aspektklasse stehenden Material materialspezifisch umgesetzt werden.

Dies ermöglicht die Definition von Werkzeug- und Materialklassen im Kontext einer Aspektklasse α [BCS92]:

Def. 5-1: Werkzeugklasse im Kontext α

Eine Klasse ist eine Werkzeugklasse im Kontext α, wenn sie die Aspektklasse α benutzt.

Def. 5-2: Materialklasse im Kontext α

Eine Klasse ist eine Materialklasse im Kontext α, wenn sie von der Aspektklasse α erbt.

Komplexe Werkzeuge haben in der Regel mehrere Materialien, auf welche sie über unterschiedliche Aspektklassen zugreifen. Der Auflister benötigt zwei Materialien, das eine indizierbar (z.B. eine geordnete Kollektion oder das Terminbuch), das andere auflistbar (z.B. den Einzeltermin). Die auflistbaren Materialien werden vom indizierbaren Material zur Verfügung gestellt.

Die konzeptuelle Einheit Werkzeug–Aspektklasse–Material ist das zentrale Strukturierungsmuster für den Softwareentwurf von Werkzeugen und Materialien. Es lenkt, wie in den Metaphern vorgegeben, den Blick auf den Verwendungszusammenhang.

Durch das zusätzliche Kopplungsglied der Aspektklasse werden Systeme flexibler und grenzen sich von Entwürfen ab, in den Materialien „sich selbst graphisch darstellen“ oder „sich selbst editierbar“ machen. Im letzteren Ansatz fallen Werkzeug und Material zusammen und verhindern die Wiederverwendbarkeit, welcher durch die erhöhte Flexibilität der Entkopplung in Werkzeuge, Aspektklassen und Materialien gewonnen wird.

Zusatz Das hier vorgestellte Konzept der Aspektklassen ist eine Möglichkeit, Sichten auf Materialien zu definieren. Problematisch an diesem Ansatz ist, daß bei komplexen Materialien die Schnittstelle und der Zustand von Objekten zu groß wird. Die Schnittstelle wird zu groß, wenn sie von sehr vielen Aspektklassen erbt. Die Objekte werden „zu dick“, wenn ihre interne Repräsentation, ihr Implementationszustand, sich zu stark ausweitet.

Es gibt verschiedene Antworten auf diese Problematik. Die einfachste Lösung ist, mit Hilfe von Umwicklern die Schnittstelle eines Objekts an einen neuen Kontext anzupassen. In Abb. 5-8 stellt die Klasse AuflistbarerTermin eine Umwicklerklasse für Objekte der Klasse Termin dar. Weil AuflistbarerTermin von Auflistbar erbt, kann jedes umwickelte Terminobjekt unter der Schnittstelle Auflistbar verwendet werden. Somit kann es von einem Auflister angezeigt werden.

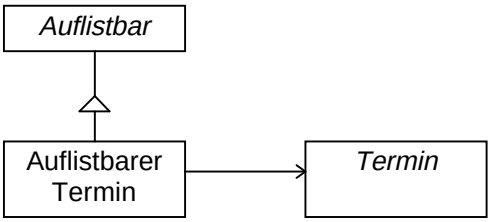


Abb. 5-8: Eine Umwicklerklasse für Objekte der Klasse Termin zur Anpassung an Auflistbar.

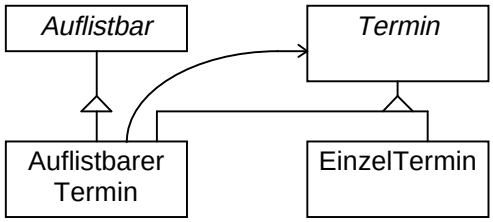


Abb. 5-9: Ein mit der Aspektklasse Auflistbar erweiterter Decorator für Terminobjekte.

Eine Erweiterung des Umwickleransatzes stellt der Kompositionsansatz dar, bei dem die Umwicklerklasse nicht nur von der Schnittstellenklasse erbt, an die das umwickelte Objekte anzupassen ist, sondern von der Klasse des umwickelten Objekts selbst. Der Umwickler kann somit an dessen Stelle verwendet werden. Dies Konzept ist als Decorator Muster [GHJ+95] bekannt. Es ermöglicht es, dynamisch Funktionalität und Zustand eines Objekts zu erweitern.

In Abb. 5-9 ist dargestellt, wie der Kompositionsansatz im Rahmen der Werkzeug und Material Metapher eingesetzt werden kann. Der Umwickler erbt sowohl von der Aspektklasse Auflistbar als auch von der Oberklasse Termin. Der Umwickler kann also gleichermaßen von einem Auflister bearbeitet, als auch unter der Terminschnittstelle weiterverwendet werden. Intern delegiert der Umwickler die meisten Operationsaufrufe an das eingebettete Terminobjekt. Von ihm erfragt es auch die für die Auflistbarkeit relevante Information.

Ein Vorteil dieses Ansatzes ist, daß die Oberklasse Termin nicht mit der zusätzlichen Aspektklassenschnittstelle belastet wird. Weiterhin wird die Erweiterung des Zustands eines Objekts durch den Umwickler entkoppelt, belastet also das eigentliche Objekt nicht. Dies sind aber zugleich auch die Nachteile des Ansatzes: Soll ein Objekt in einem bestimmten Kontext verwendet werden, für welches es nicht die richtige Schnittstelle besitzt, so muß erst ein Umwickler erzeugt werden. Müssen Objekte persistent gehalten werden, so muß zusätzlich zum umwickelten Objekt auch noch der Umwickler gespeichert werden. Es kommt also auf den Kontext an, ob Umwickler oder direktes Erben von Aspektklassen anzuwenden sind.

Auf diese Argumentation aufbauend, lassen sich drei Ebenen der fachlichen und technischen Modellierung erkennen: Die Einfachvererbungshierarchie, die Aspektklassenkomposition und die Umwicklerkomposition.

- Das „Skelett“ einer Klassenhierarchie ist ein Einfachvererbungsbaum, welcher das Ergebnis der Modellierung fachlicher Begriffe der Anwendungswelt darstellt. Er repräsentiert die übliche Klassifikationshierarchie, wie sie nicht nur im objektorientierten Entwurf sondern vielen anderen Wissenschaften auch verwendet wird.
- Aus der Analyse von Arbeitstätigkeiten und Verwendungszusammenhängen ergeben sich verschiedene Anforderungen an die Funktionalität von Materialien. Die Entwickler entscheiden, welche dieser Verwendungszusammenhänge als so grundlegend zu betrachten sind, daß sie unter keinen Umständen von der Klasse abgelöst werden können.[8] Diese Aspekte werden direkt durch Erben von Aspektklassen in die Klassenschnittstelle des Materials übernommen.
- Nicht als grundlegend erachtete aber trotzdem benötigte Verwendungszusammenhänge von Materialien werden als Umwickler realisiert. Umwickler werden gebraucht, um Materialien in Verwendungszusammenhängen zu benutzen, für welche sie nicht die richtige Schnittstelle besitzen. Dies ist dann der Fall, wenn die benötigten Eigenschaften und Attribute nicht in der Materialklasse selbst modelliert wurden.

Dieser Ansatz trägt durchaus für mittelgroße bis große Systeme. Aber auch er kann sich auf lange Sicht als problematisch erweisen, wenn neue Sichten in bereits existierende Systeme eingebracht und diese weiterentwickelt werden müssen. Neuere Forschungsergebnisse weisen darauf hin, daß die Definition von Sichten auf Materialien ein Vorgang ist, der grundlegend für die Evolution von Softwaresystemen ist und dem bereits in der Konstruktion Rechnung getragen werden muß [OH92, HO93].

Vergleiche Eine ähnliche Diskussion, wenngleich auf abstrakterem Niveau, wurde unter der Bezeichnung Schnittstellenklassen geführt [CCH+89]. Im Rahmen des Arbeitens mit CRC-Karten [BC89] können sie auch als Definition von „Contracts“ aufgefaßt werden [WJ90, WWW90]. Weiterhin gibt [RAB+92] eine gute Diskussion der über Schnittstellenklassen definierten Kontexte und Verwendungsmöglichkeiten.

---

[8] Die Frage, welche Funktionalität grundlegend ist, kann nur subjektiv entschieden werden. Insofern stellt die Entscheidung einen willkürlichen Akt dar, der in der Diskussion und Erfahrung der Entwickler und auch Anwender abgesichert werden sollte.

5.3 Werkzeugkomposition

Werkzeuge werden rekursiv aus Subwerkzeugen aufgebaut. Jedem Werkzeugobjekt in der resultierenden Baumstruktur kommt eine Aufgabe zu, welche es durch eigene Leistung sowie Delegation von Teilaufgaben an Subwerkzeuge realisiert.

Problem Es ist mühsam, ein Werkzeug von Grund auf neu zu entwerfen. Besser wäre es, existierende Werkzeuge wiederzuverwenden. Dazu wird eine Strategie benötigt, welche wiederzuverwendende Werkzeuge in einen übergeordneten Zusammenhang einbettet.

Kontext Die allgemeine Wiederverwendungsdebatte ist lang und es lassen sich viele gute Hinweise zum Entwurf von wiederverwendbaren Klassen entnehmen. Hier soll lediglich auf die spezifische Situation der Werkzeugkonstruktion und -wiederverwendung eingegangen werden.

Ein Softwareprojekt ist gut beraten, einen Satz einfacher wiederverwendbarer Werkzeuge zu erstellen oder von früheren Projekten zu übernehmen, respektive einen Anwendungsrahmen zu verwenden. Einfache Werkzeuge sind z.B. der Auflister und der Editor, welche mit jedem Material arbeiten können, das auflistbar oder editierbar ist.

Das Kalenderwerkzeug aus Abb. 5-2 soll diese beiden Werkzeuge wiederverwenden, um die Einzeltermine anzuzeigen (linke Hälfte des Kalenders aus Abb. 5-2) und zu editieren (rechte Seite). Abb. 5-10 zeigt das Kalenderwerkzeug, noch ohne den Auflister und Editor wiederverwendet zu haben. Wie an den Aussparungen erkennbar, ist es aber dafür vorbereitet.

Abb. 5-10: Das Kalenderwerkzeug ohne Auflister und Editor. Diese müssen in den Zusammenhang des Kalenders eingefügt werden.

Eine Lösung muß Fragen der Einbettung und Wiederverwendung von Werkzeugen klären, aufzeigen, woran wiederverwendbare Werkzeuge erkennbar und wie sie zu entwerfen sind, eine Richtlinie zum Zusammenbau komplexer Werkzeuge geben und die Anbindung zusammengesetzter Werkzeuge an Aspektklassen und Materialien erläutern. Deswegen:

Lösung Ordne einem Werkzeug eine eindeutig bestimmbare Aufgabe zu. Baue Werkzeuge aus *Subwerkzeugen* zusammen, so daß sich ein Werkzeugbaum ergibt. Nimm die Zerlegung anhand der den Werkzeugen zugeordneten Aufgabe vor: Eine komplexe Aufgabe wird von einem *zusammengesetzten Werkzeug* erledigt, Teilaufgaben der komplexen Aufgabe von Subwerkzeugen, die für diese Teilaufgabe entworfen wurden.

Ein zusammengesetztes Werkzeug delegiert Teile der ihm zugedachten Aufgabe an Subwerkzeuge und interpretiert deren Ergebnisse in Hinblick auf seine eigene Aufgabe. Es bildet den *Kontext* für seine Subwerkzeuge. Subwerkzeuge können selbst zusammengesetzte Werkzeuge sein, welche Teilaufgaben an ihre Subwerkzeuge delegieren, oder sie können *einfache Werkzeuge* sein, welche in der Lage sind, die ihnen zugedachte Aufgabe ohne weitere Hilfe auszuführen.

Zusammengesetzte Werkzeuge sind Knoten im Werkzeugbaum, einfache Werkzeuge sind die Blätter des Werkzeugbaums. Ein Werkzeug, welches Subwerkzeuge verwendet, heißt *Kontextwerkzeug für die Subwerkzeuge*. Die Aufgabenverteilung innerhalb der resultierenden Baumstruktur entspricht dem bekannten Bürokratiemodell [Web72].

Ein zusammengesetztes Werkzeug versteckt seine Subwerkzeuge zwar nicht, tritt aber nach außen hin als ein Werkzeug auf. Ein zusammengesetztes Werkzeug erzeugt, verwendet und löscht ein Subwerkzeug nach eigenem Gutdünken und der Aufgabe entsprechend. Die Klassenbaumstruktur ist in Abb. 5-11 dargestellt und entspricht dem Entwurfsmuster *Composite* [GHJ+95], welches die Erstellung rekursiver Baumstrukturen von Objekten beschreibt.

Das Kalenderwerkzeug ist ein zusammengesetztes Werkzeug, welches Teilaufgaben an die zwei Subwerkzeuge Auflister und Editor delegiert (Abb. 5-12). Auflister und Editor sind einfache Werkzeuge. Der Kalender stellt ihr Kontextwerkzeug dar.

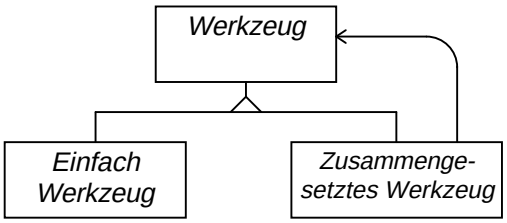


Abb. 5-11: Klassenstruktur für einfache und zusammengesetzte Werkzeuge.

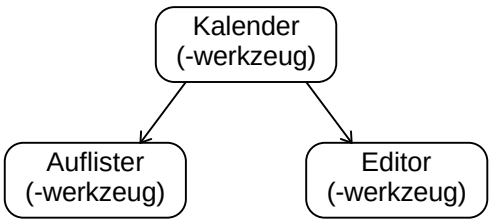


Abb. 5-12: Der Kalender benutzt die Subwerkzeuge Auflister und Editor als ihr Kontextwerkzeug.

Jedes Werkzeug dient zur Bearbeitung einer wohldefinierten Aufgabe. Die dazu benötigten Daten oder Materialien erhält ein Werkzeug vom Benutzer, von seinen Subwerkzeugen oder von seinem Kontextwerkzeug. Daten sind hier zumeist einfache Werte [Mac82, EK95], welche nicht die Komplexität von Materialien besitzen. Jeder dieser drei Arten kommt eine unterschiedliche Bedeutung zu:

- Gemäß der Werkzeugmetapher wartet ein Werkzeug auf Eingaben von Benutzenden, um diese zu interpretieren und umzusetzen. Es macht keine Annahmen über ihren Zeitpunkt und möglichst keine über Reihenfolgen von Eingaben.
- Ein Werkzeug holt sich von seinen Subwerkzeugen die benötigten Daten und Materialien nach Belieben. Es legt Zeitpunkt und Reihenfolge fest und es interpretiert und manipuliert sie in seinem Sinne.
- Ein Werkzeug erhält von seinem Kontextwerkzeug Daten und Materialien, welche es seinem gegenüber dem Kontext eingeschränkten Verständnis gemäß interpretiert. Es macht keine Annahmen über Reihenfolgen, solange sie nicht in seiner eigenen technischen Schnittstelle vertraglich festgelegt wurden.

Wesentlicher Aspekt der eben geführten Diskussion ist, daß Werkzeuge die ihnen zur Verfügung stehenden Daten und Materialien nur im Rahmen der ihnen gegebenen Möglichkeiten interpretieren. Damit ist klar, daß ein Werkzeug keine Annahmen über seinen Kontext und seine Verwendung machen darf, da andernfalls seine Wiederverwendung auf jene Kontexte respektive Werkzeuge eingeschränkt wird, auf die es sich bezieht.

Zusatz Die geschilderte Aufgabenverteilung darf nicht mit funktionaler Dekomposition im Sinne von Structured Design zu verwechseln. Es wird kein strenger Kontrollfluß von der Wurzel in die Blätter des Werkzeugbaums festgelegt, sondern Benutzer können jederzeit mit Werkzeugen in der Hierarchie frei interagieren. Werkzeuge sind nicht eine Ansammlung von Funktionen sondern besitzen einen eigenen Zustand unabhängig von Materialien, welcher

die Arbeitssituation und den Werkzeugzustand als wichtige Information für die Benutzung vermerkt.

Vergleiche Das geschilderte Prinzip der Aufgabenverteilung und Delegation stammt nicht aus der Informatik, sondern ist schon lange als Prinzip zur Strukturierung von Behörden und anderen hierarchischen Organisationen bekannt [Divide et Impera!, Web47, Web72]. Für die Wiederverwendungsdebatte auf Klassen und Komponenten Ebene siehe unten anderem [MM92, Riz93, TGP89].

5.4 Trennung von Interaktion und Funktion

Ein Werkzeug besteht aus mindestens zwei Objekten: einer Interaktionskomponenten und einer Funktionskomponente. Die Interaktionskomponente präsentiert Werkzeug und Material und ermöglicht seine Handhabung. Die Funktionskomponente realisiert die vom Werkzeug angebotene Funktionalität unabhängig von einer gewählten Benutzungsschnittstelle.

Problem Ein Werkzeug muß gleichermaßen sich selbst und seine *Materialien präsentieren*, wie auch die zur Bearbeitung der Materialien benötigte *Funktionalität realisieren*. Wird Präsentation mit Funktionalität vermischt, so ergeben sich Abhängigkeiten, welche die Weiterentwicklung und Anpassung erschweren. Somit sollten beide Bereiche getrennt werden.

Kontext Die Präsentation eines Werkzeugs sowie die Interaktion mit ihm hängen von der gewählten Benutzungsumgebung ab (Terminals, graphische Benutzungsschnittstellen, fachspezifische Eingabegeräte). Die Realisierung der Funktionalität eines Werkzeugs ist davon weitgehend unabhängig und nur dadurch bestimmt, was konzeptuell mit Materialien gemacht werden soll.

Somit kann jedes Werkzeug in zwei Teile aufgespalten werden: In einen Teil, der Interaktion und Präsentation ermöglicht und einen anderen Teil, der die dahinterstehende Funktionalität realisiert. Diese sogenannte Trennung von Interaktion und Funktion ist wohlbekannt und soll auch für den Entwurf von Werkzeugen nach der Werkzeug und Material Metapher gelten.

Die Benutzungsschnittstelle eines Werkzeugs muß dabei verschiedenen Anforderungen genügen, um fachlich qualifizierte Arbeit zu ermöglichen:

- Sie muß reaktiv sein, d.h. einem Benutzer adäquates Feedback über die Auswirkungen seiner Handlungen und den Fortgang seiner Arbeit geben.
- Sie soll außer in Ausnahmefällen nicht modal sein, d.h. der Benutzer muß seine Arbeitsabläufe selbst bestimmen können.
- Sie muß die Werkzeugfunktionalität wahlfrei, sofern nicht durch fachliche Randbedingungen eingeschränkt, zur Verfügung stellen. So wird Sequentialisierung verhindert.
- Sie muß den Werkzeugzustand nicht aufdringlich anzeigen und die Präsentation und Handhabung der Materialien in den Mittelpunkt stellen.

Die Funktionalität eines Werkzeugs entspricht der Arbeitsaufgabe, für die das Werkzeug entworfen wurde. Die Realisierung der Aufgabe stützt sich auf den durch Aspektklassen für Materialien definierten Verwendungszusammenhang ab.

Eine Unterteilung des Werkzeugs in eine Benutzungsschnittstelle und einen die Funktionalität realisierenden Teil muß diesen Anforderungen genügen. Deswegen:

Lösung Baue ein Werkzeug aus einer oder mehreren *Interaktionskomponenten* und einer *Funktionskomponente* zusammen. Lasse die Interaktionskomponente (Ik) die Präsentati-

on und Handhabung des Werkzeugs erledigen und die Funktionskomponente (Fk) die Funktionalität des Werkzeugs realisieren. Entwerfe die Ik so, daß sie den oben genannten Forderungen entspricht, also reaktiv und nicht modal ist.

Die Interaktionskomponente steuert die Präsentation und Handhabung des Werkzeugs. Die Funktionskomponente eines Werkzeugs bewahrt den fachlichen, nicht handhabungsgebundenen Zustand des Werkzeugs und ermöglicht die Sondierung und Manipulation von Materialien. Die von der Ik realisierten mitunter sehr komplexen Handhabungsformen werden in Benutzungsaufrufe der Fk umgesetzt, welche die übergebenen Daten interpretiert und anwendet.

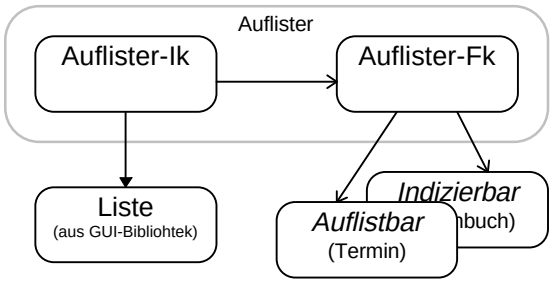


Abb. 5-13: Der Auflister aus Ik und Fk mit List-It für die Oberfläche sowie den Aspektklassen zu Verabredungen und dem Container.

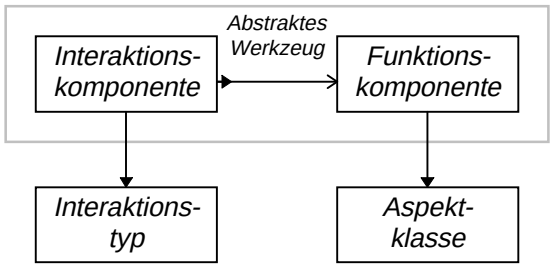


Abb. 5-14: Schematische Struktur eines Werkzeugs samt Abstützung auf Interaktionstypen und Aspektklassen.

Als einfaches Beispiel sei die Operation `NeueVerabredung` genannt, welche die Fk des Kalenders möglichen Ik's anbietet. Diese fachliche Operation kann von einer Ik über einen Menüpunkt im üblichen Bearbeitungsmenü (Abb. 5-10, Menüleiste, Menü Bearbeitung), durch einen Knopf (Abb. 5-10, Knopfleiste links unten) oder durch das für Termine vorgesehen Datumsfeld (Abb. 5-10, links oben unter der Menüleiste) dem Benutzer präsentiert werden. Im Fall der Auswahl eines Menüpunkts oder der Betätigung des Knopfs ist die Aktivierung dieser Operation für den Benutzer explizit, im Fall der Eingabe in das Datumsfeld mit anschließender Bestätigung ist sie implizit. Das Kalenderbeispiel bietet somit in der Benutzungsschnittstelle drei redundante Möglichkeiten, welche mit Hilfe einer einzelnen Fk Operation realisiert sind.

Abb. 5-13 zeigt das Beispiel des Auflisters, der aus einer Ik und einer Fk besteht. Erkennbar ist, daß die Ik ihre Fk benutzt, also von ihr abhängig ist. Die Umkehrung gilt nicht. Somit kann die Interaktionskomponente an wechselnde Benutzungsschnittstellen angepaßt werden, ohne daß die Funktionskomponente verändert werden müßte.

Interaktionskomponenten verwenden sogenannte Interaktionstypen, welche die in einer Umgebung möglichen Benutzungsschnittstellenelemente kapseln. In graphischen Benutzungsoberflächen sind dies z.B. Fenster, Knöpfe, Menüleisten usw., mit deren Hilfe die Benutzungsschnittstelle zusammengebaut wird. Der schematische Zusammenhang von Ik, Fk, Interaktionstypen (It) und Aspektklassen ist in Abb. 5-14 noch einmal dargestellt.

Zusatz Mit Einführung dieses Entwurfsmusters verschwindet das Werkzeugobjekt, von dem bisher die Rede war. Es wird durch eine Gruppe mehrerer Objekte ersetzt, minimal einer Interaktions- und Funktionskomponente. Konzeptuell ist es sinnvoll, weiterhin von einem Werkzeug(objekt) zu sprechen. Muß ein Werkzeug durch ein technisches Objekt repräsentiert werden, so sollte dazu die Funktionskomponente verwendet werden. Sie ist der Ort, an dem die Funktionalität zur Strukturierung eines komplexen Werkzeugs angesiedelt wird und deswegen gut als Vertreter des Werkzeugs fungieren kann. In Situationen, in denen die Trennung von Interaktion und Funktion zu schwergewichtig wird, ist es denkbar, die Interaktions- und Funktionskomponenten wieder zu einem Objekt zusammenfallen zu lassen.

Vergleiche Die Trennung von Interaktion und Funktion ist wohl bekannt. Die Grundlagen sind in [DHM89] gut dargestellt. Spezifische und interessante Ausformungen sind MVC [KP88], ALV [Hil92] und CommonInteract [SP93].

5.5 Werkzeugkopplung

Besitzt ein Werkzeug Subwerkzeuge, so werden die Ik's des Werkzeugs zu Kontext-Ik's, welche auf den Ik's der Subwerkzeuge arbeiten, und die Fk des Werkzeugs zur Kontext-Fk, welche auf der Fk des Subwerkzeugs arbeitet. Auf- und Abbau von Subwerkzeugen wird von der Kontext-Fk entschieden, softwaretechnisch aber von ihr entkoppelt.

Problem Ein Werkzeug besteht immer aus einer oder mehreren Interaktionskomponenten und einer Funktionskomponente. Besitzt ein Werkzeug gemäß des Musters der *Werkzeugkomposition* Unterwerkzeuge, welche ihrerseits aus Ik's und Fk's bestehen, so müssen diese an die Ik's und Fk ihres Kontextwerkzeugs angebunden werden.

Kontext Werkzeuge werden vertikal nach dem Entwurfsmuster der Werkzeugkomposition in Kontext- und Subwerkzeuge aufgeteilt und horizontal nach dem Muster der Trennung von Interaktion und Funktion in Interaktions- und Funktionskomponenten aufgespalten.

Geklärt werden muß, wie die Interaktions- und Funktionskomponenten von Kontextwerkzeugen mit denen ihrer Subwerkzeuge verbunden werden, wie sie erzeugt und gelöscht werden und wo die Verfügungsgewalt über solche Entscheidungen liegt. Deswegen:

Lösung Mache die Funktionskomponente eines Werkzeugs zum Stellvertreter dieses Werkzeugs. Übertrage ihr die Entscheidungsgewalt zum Erzeugen und Löschen von Subwerkzeugen. Bilde den konzeptuellen Werkzeugbaum isomorph auf einen Funktionskomponentenbaum ab. Ordne die Interaktionskomponenten ihren Fk's zu. Falls nötig, verbinde die Ik's untereinander zu einem Ik-Baum, welcher den Fk-Baum nachvollzieht.

Die Werkzeugstruktur ergibt sich als die Fk-Baumstruktur, in welcher eine Fk als logischer Vertreter ihres Werkzeugs zu verstehen ist. Sie fällt die Entscheidung, ob ein Subwerkzeug zu erzeugen oder zu löschen ist. Sie führt dies aus, in dem sie die Fk des Subwerkzeugs erzeugt respektive löscht.

Abb. 5-15 zeigt schematisch die resultierende Objektbeziehungsstruktur. Die Ik und Fk eines Kontextwerkzeugs heißt Kontext-Ik und -Fk respektive, die Ik und Fk eines Subwerkzeugs Sub-Ik und -Fk. Anders als eine Kontext-Fk muß eine Kontext-Ik nicht zwingend mit ihren Sub-Ik's arbeiten. Wenn es keine handhabungs- und präsentationsbedingten Abhängigkeiten gibt, wird die Kontext-Ik ihre Sub-Ik's nicht weiter benutzen. Eine Kontext-Fk und ihre Ik's *beobachten* die Sub-Fk. Dies geschieht mit Hilfe des in 5.6 erläuterten Musters des Ereignismechanismus.

Das Erzeugen der Ik's eines Werkzeugs geschieht nach dem Erzeugen der Fk und wird innerhalb des Anwendungsrahmens mit Hilfe später Erzeugung anonym und automatisch ausgeführt [RZ94]. Die Verwaltung von Ik's und Fk's und ihre Zusammengehörigkeit ist als generische Funktionalität in den Oberklassen des Anwendungsrahmens realisiert. Das Löschen der Ik's zu einer Fk geschieht mit Hilfe dieser Verwaltungsinformationen und passiert, bevor die Fk gänzlich gelöscht wird. Dabei wird geprüft, ob der aufgrund graphischer Benutzungsschnittstellen invertierte Kontrollfluß durch bereits gelöschte Objekte zurückkehrt, so daß Laufzeitfehler auf jeden Fall verhindert werden.

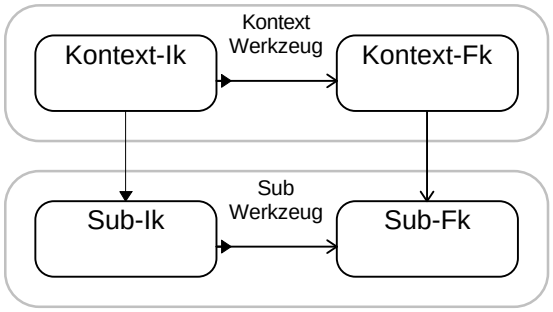


Abb. 5-15: Schematische Darstellung der Kontext- zu Subwerkzeugbeziehung auf Objektebene.

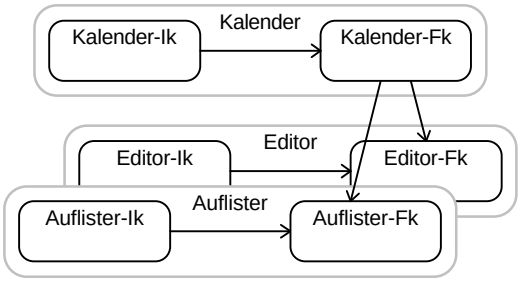


Abb. 5-16: Objektstruktur des Kalenders mit den Unterwerkzeugen Auflister und Editor.

Die Betrachtung des Kalenderbeispiels in Abb. 5-16 zeigt diese Struktur auf: Die Kalender-Fk des Kalenderwerkzeugs besitzt die zwei Sub-Fk's Editor- und Auflister-Fk. Da jedes Werkzeug genau eine Fk besitzt, bilden sie die Werkzeugkomposition aus dem Kontextwerkzeug Kalender und den Subwerkzeugen Auflister und Editor nach. Zu jeder Fk eines Werkzeugs gehört im Beispiel genau eine Interaktionskomponente. Die Ik's sind nicht untereinander verbunden. Der relevante Informationsfluß findet über die Fk's statt, welche Benutzungsaktionen wie die Listenauswahl und das Editieren eines Texts fachlich umsetzen.

Auch wenn Interaktionskomponenten auf die Benutzung von Sub-Ik's verzichten mögen, so bilden sie doch einen Ik-Baum. Dies wird daran deutlich, daß eine Ik immer eine eigenständige Benutzungsschnittstelle besitzt, welche in die Benutzungsschnittstelle einer umgebenden Kontext-Ik eingebettet werden muß.

Die konkreten Benutzungsschnittstellen einer Ik werden mit Hilfe von Interaktionstypen (It's) realisiert, welche die Benutzungsschnittstellenelemente des zugrundeliegenden Fenstersystems kapseln. Betrachtet man die Benutzungsschnittstelle einer Ik als Teiloberfläche, so ergibt sich aufgrund der Werkzeugkomposition auch ein Teiloberflächenbaum, welcher dem Ik-Baum entspricht.

Soll die Wiederverwendbarkeit eines Werkzeugs nicht eingeschränkt werden, so darf die Ik und ihre Teiloberfläche keine Annahmen über den Kontext ihrer Verwendung machen. Dazu sollte ein Ik/Fk Paar immer als Werkzeugkomponente und nicht als ein vollständiges Werkzeug entworfen werden. Eine Werkzeugkomponente wird zum Werkzeug, in dem es in einen entsprechenden Rahmen (und sei dies nur ein einzelnes Fenster) gepackt wird. Folgende Richtlinien haben sich als hilfreich erwiesen:

- Eine Ik sollte bestimmte Benutzungsschnittstellenelemente, z.B. Fenster, als oberstes Benutzungsschnittstellenelement nicht selbst erzeugen. Fenster schränken die Wiederverwendungsmöglichkeit von Ik's drastisch ein, da sie nicht mehr beliebig eingebettet werden können. Statt dessen sollte eine Ik sich darauf verlassen, von ihrer Kontext-Ik ein Fenster zu erhalten, in dem sie ihre Teiloberfläche aufbauen kann.
- Somit muß jede Ik ein ausgewiesenes Benutzungsschnittstellenelement besitzen, innerhalb dessen die Ik's der Subwerkzeuge ihre Teiloberfläche aufbauen können. Im Anwendungsrahmen ist dies insbesondere die Klasse `ItBord`, aber auch `ItHBox`, `-VBox` usw. Interaktionstypen dieser Klassen können beliebige weitere It's enthalten, welche die Wurzel für eine eigenständige Teiloberfläche darstellen.
- Beim Layout der Oberfläche eines zusammengesetzten Werkzeugs dürfen die Benutzungsschnittstellenelemente der Subwerkzeuge nicht auf Elemente des Kontextwerkzeugs zurückgreifen, weil sie damit den Kontext der Verwendung des Werkzeugs einschränken.

- Jeder Interaktionstyp erhält zur Laufzeit von seiner Ik einen eindeutigen Namen, welcher sich aus dem Werkzeugnamen und einem von der Ik vorgesehenem Kürzel zusammensetzt. Der Name wird dazu benutzt, aus einer sogenannten Ressourcendatenbank weitere Informationen über das konkrete Aussehen des It zu erfahren. Somit kann jeder von zwei unterschiedlichen Ik's derselben Klasse erzeugte It ein eigenständiges Aussehen besitzen.

Ergänzend muß dazu gesagt werden, daß jede Fk zur Laufzeit einen eigenständigen Namen besitzt, der nicht von der Klasse vorgegeben sein muß. So ist es möglich, von derselben Klasse `FkGlossar` zur Laufzeit mehrere Objekte zu haben, welche einem ihrer Aufgabe entsprechenden Werkzeugnamen, z.B. Objektlexikon oder Funktionslexikon, tragen.

Vergleiche Die unter Trennung von Interaktion und Funktion genannten Strukturierungskonzepte (MVC, ALV, CommonInteract) besitzen ebenfalls eigene Kompositionsstrategien. Im Falle von Smalltalk sind dies z.B. *CompositeViews* oder proprietäre Entwicklungen, d.h. Entwicklungen der jeweiligen Hersteller, z.B. *Pluggable Views*, *Value Models* und *Application Models* [Par94a, Par94b, Woo95] von ParcPlace Systems oder *PARTS* von DigiTalk.

5.6 Ereignismechanismus

Kontext-Fk und Sub-Ik *benutzen* die Sub-Fk und bleiben dabei für sie anonym. Da sie aber vom Zustand der Sub-Fk und ihres Materials abhängig sind, *beobachten* sie die Sub-Fk zudem und werden von ihr über etwaige Zustandsänderungen *benachrichtigt*.

Problem Die Präsentation von Werkzeug und Materialien durch eine Ik hängt vom Zustand ihrer Fk und ihrer Materialien ab. Der Zustand einer Kontext-Fk hängt ebenfalls vom Zustand ihrer Sub-Fk's ab. Die abhängigen Ik's und die Kontext-Fk müssen deswegen über relevante Zustandsänderungen benachrichtigt werden. Der für diese Benachrichtigung notwendige Datenfluß läuft der existierenden Benutzungsbeziehung zuwider.

Kontext Wie in den Entwurfsmustern Werkzeugkomposition und Trennung von Interaktion und Funktion ausgeführt, ist eine Werkzeug-Fk von ihren Ik's und ihrer Kontext-Fk unabhängig. Sie darf also nicht explizit auf sie Bezug nehmen und ihre konkreten softwaretechnischen Schnittstellen nicht kennen.

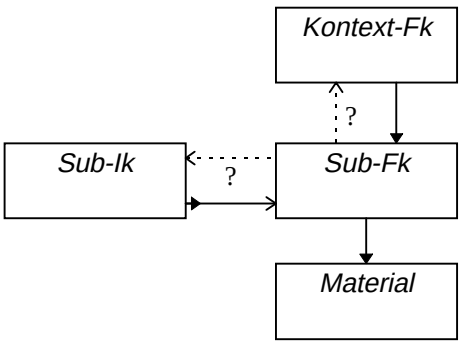


Abb. 5-17: Der problematische Kontroll- und Datenfluß ist gestrichelt und mit einem Fragezeichen versehen gezeichnet.

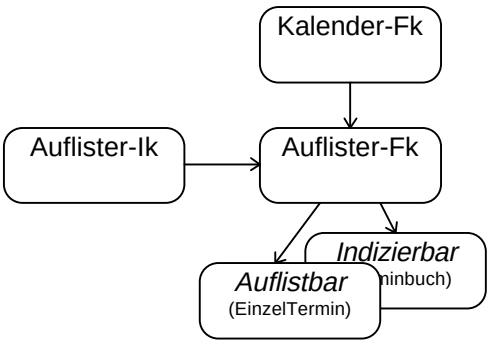


Abb. 5-18: Kalender-Fk und Auflister-Ik hängen vom Zustand der Auflister-Fk und ihren Materialien ab.

Gleichwohl hängen Kontext-Fk und die Ik's einer Fk von ihrem Zustand und dem ihrer Materialien ab (Abb. 5-17). Die Ik's benötigen das Wissen um den Zustand ihrer Fk zur adäquaten Präsentation und Handhabung des Werkzeugs; die Kontext-Fk benötigt dieses Wissen, um

über den Zustand der an die Sub-Fk delegierten Aufgabe informiert zu sein und ihren eigenen Zustand danach auszurichten.

Abb. 5-18 zeigt dies am Beispiel der Kalender/Auflister Beziehung: Die Auflister-Ik muß von ihrer Fk benachrichtigt werden, wenn eine neue Verabredung erzeugt und als auflistbares Material in die Liste der anzuzeigenden Materialien eingefügt wurde. Die Kalender-Fk muß informiert werden, wenn über die Auflister-Ik ein neues auflistbares Material ausgewählt wurde, um sich und den Editor in den neuen Bearbeitungszustand zu versetzen. In beiden Fällen widerspricht der notwendige Daten- und Kontrollfluß der Benutzungsbeziehung von Auflister-Ik und Kalender-Fk zur Auflister-Fk.

Um den beiden zuerst genannten Entwurfsmustern weiterhin zu genügen, darf die Benutzungsbeziehung von Ik zu Fk und Kontext-Fk zu Sub-Fk nicht durch ihre Umkehrung bidirektional werden, da sonst die Wiederverwendungsmöglichkeiten drastisch eingeschränkt würden. Es muß eine anonymer Mechanismus gefunden werden, welcher es erlaubt, die abhängigen Ik's und die Kontext-Fk über Zustandsänderungen zu benachrichtigen ohne die Benutzungshierarchie aufzubrechen. Deswegen:

Lösung Bilde zu jeder relevanten Zustandsänderung genau ein *Ereignisobjekt* und bette es in der Schnittstelle der Fk ein. Interpretiere das Eintreten der Zustandsänderung als *Ereignis*. Das Ereignisobjekt muß über eine Operation `Verkünde` verfügen, welche von der Fk aufzurufen ist, wenn die mit dem Objekt assoziierte Zustandsänderung eintritt. Sorge dafür, daß jede Ik und jede Kontext-Fk, welche an einem bestimmten Ereignis der Fk interessiert ist, sich bei dem Ereignisobjekt über die Operation `MeldeAn` als zu informierender Interessent anmeldet. Sorge weiterhin dafür, daß in der `Verkünde` Operation diese Interessenten auch informiert werden.

Die am Eintreten eines Ereignisses interessierten Objekte heißen Beobachter. Sie melden sich über die Operation `MeldeAn` mit ihrer Objektreferenz sowie einer im Falle des Ereignisses aufzurufenden Operation an. Tritt das Ereignis ein und ruft die Fk die Operation `Verkünde` des Ereignisses auf, so wird das Melden des Ereignisses durch Aufrufe der für die Beobachter registrierten Operationen umgesetzt.

Für die Fk bleiben die Beobachter, also ihre Ik's und die Kontext-Fk, anonym. Die Ereignisobjekte verfügen nur über eine anonymisierte Objekt- und Operationsreferenz, so daß keine Gefahr besteht, die erwünschte Benutzungshierarchie aufzulösen.

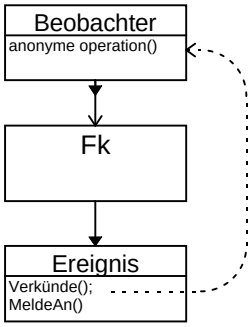


Abb. 5-19: Schematische Darstellung: Eine Fk hat mindestens einen Beobachter und bietet keines, eins, oder mehrere Ereignisse an.

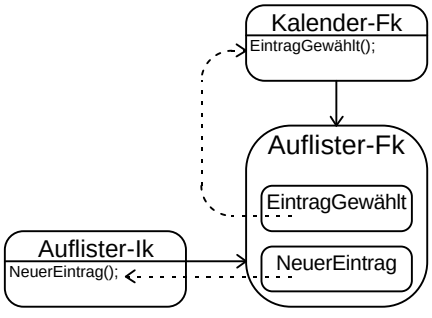


Abb. 5-20: Die Kalender-Fk läßt sich für das Ereignis EintragGewählt und die Auflister-Ik für NeuerEintrag registrieren.

Abb. 5-19 zeigt die generische Beziehung zwischen Fk, Ereignisobjekten und Beobachtern auf. Jede Fk kann mehrere Beobachter haben (mindestens eine Ik), welche sich bei den von der Fk angebotenen Ereignisobjekten registrieren lassen. Eine Fk kann je nach Komplexität ihres ab-

strakten Zustands beliebig viele Zustandsänderungen als Ereignisobjekte anbieten. Diese Ereignisobjekte sind Teil der öffentlichen Schnittstelle einer jeden Fk und stehen somit beliebigen Beobachtern zur Verfügung.

In Abb. 5-20 sind die Ereignisse `EintragGewählt` und `NeuerEintrag` für den Auflister eingeführt, bei denen sich die Auflister-Ik respektive Kalender-Fk registrieren lassen. Wird z.B. von der Auflister-Ik die Fk-Operation `WähleAus(Index)` aufgerufen, so aktiviert die Fk das Ereignis `EintragGewählt`, welches in diesem Fall die Kalender-Fk benachrichtigt. Eine Diskussion verschiedener Realisierungsmöglichkeiten („Design Space“) führen Garlan et al. (Implicit Invocation [GS93]) und Gamma et al. (Observer [GHJ+95]).

Zu bemerken ist, das der Begriff des *Ereignisses* anders gebraucht wird als in den meisten anderen Ereignismechanismen [GR83, WGM89, Rei90]. Ereignisse sind nicht Objekte oder Konstanten, welche alle über dieselbe Operation an abhängige Beobachter geschickt werden. Sie besitzen konzeptuell keinen Zustand und haben keine Identität.

Ereignisse finden spontan statt. Sie signalisieren Zustandsübergänge einer Fk. Aus Implementationsgründen werden sie zumeist durch Ereignisobjekte vergegenständlicht. Die Ereignisobjekte sind in der Schnittstelle einer Fk angesiedelt sind und rufen, wenn sie eintreten, *spezifische Operationen* von Beobachtern auf, welche sich explizit dafür angemeldet haben.

Trotz der genannten Vorteile ist der Einsatz des Ereignismechanismus in zweierlei Hinsicht problematisch. Zum einen kann es passieren, daß sich Beobachter und Fk in einer Endlosschleife verfangen, weil eine Änderung zu einem Ereignis führt, welches wiederum eine Veränderung bewirkt, die zu einem Ereignis führt usw. Zum anderen besteht die Gefahr, implizite Abhängigkeiten einzuführen, wenn es z.B. notwendig wird, neue Ereignisse in der Fk-Schnittstelle einzuführen, um neuen Anforderungen der Ik oder Kontext-Fk zu genügen. Das erste Problem läßt sich durch Einhalten der folgenden Richtlinien[9] weitgehend vermeiden:

Richtlinie zum Entwurf von Beobachtern:

Entwerfe einen Beobachter immer so, daß er im Falle eines Ereignisses nie verändernd auf den Urheber reagiert, sondern immer nur sondierend.

Richtlinie zum Entwurf von beobachtbaren Objekten (Seiteneffektfreiheit):

Entwerfe die Klassenschnittstelle und implementiere die Klasse so, daß klar zwischen verändernden und sondierenden Operationen unterschieden werden kann.

Das zweite Problem impliziter Abhängigkeiten ist eine Frage des guten Entwurfs. Ereignisse werden aufgrund relevanter Zustandsänderungen des abstrakten Zustandes einer Klasse entworfen. Da der abstrakte Zustand Teil des durch die Klassenschnittstelle angebotenen Vertrags mit Klienten ist [Mey88, Mey91], reduziert sich das Problem auf die sorgfältige Analyse und Umsetzung der geforderten Funktionalität einer Fk. Zu unternehmende Schritte sind:

- Sorgfältige Analyse des abstrakten Zustands der Fk und der vermittelten Zugriffsmöglichkeit auf Materialien;
- Ausarbeitung der am abstrakten Zustand orientierten möglichen Zustandsänderungen, z.B. durch Beschreibung mit Hilfe von endlichen Automaten oder fortgeschrittenerer

[9] Dem Wissen des Autors zufolge wurden diese Richtlinien in Bezug auf beobachtende und beobachtete Objekte das erste Mal von Karl-Heinz Sylla formuliert. Eine Veröffentlichung ist nicht bekannt.

Beschreibungsmittel [Boo94, RBP+91, CHB92, BS95];

- Auswahl der relevanten Zustandsänderungen für Verwendungszusammenhänge und ihre Vergegenständlichung als Ereignisobjekte.

Trotz der Beherrschbarkeit der zwei angeführten Probleme sollte der Ereignismechanismus nur kontrolliert und so sparsam wie möglich verwendet werden, weil durch die Umkehrung des Kontroll- und Datenflusses die Komplexität des Softwaresystems zunimmt und seine intellektuelle Beherrschbarkeit erschwert wird. Aus diesem Grunde wurde der im Prinzip allgemeine Ereignismechanismus nur für Fk's eingesetzt.

Vergleiche Aus Smalltalk ist der Vorfahre dieses Konzepts als Change-Update Mechanismus (Change Propagation) [GR83] bekannt. In [Rie93a, Rie93b] wurde er als Beobachterkonzept bezeichnet. In [GHJ+95] wird er als Observer/Subject Muster noch einmal detailliert ausgearbeitet. Von [SN92, NGG+93] wurden schließlich Ereignisobjekte eingeführt, welche der Ausgangspunkt des hier geschilderten Ereignismechanismus sind.

6 Entwurfsmuster zur Werkzeugintegration

Dieses Kapitel stellt Entwurfsmuster zur Integration mehrerer Werkzeuge an einem einzelnen Arbeitsplatz vor. Integration bedeutet hierbei, Konsistenzbedingungen zwischen Materialien zu wahren und dafür zu sorgen, daß Werkzeuge immer einen aktuellen Materialzustand anzeigen. Die betrachteten Konsistenzbedingungen gelten nur für am Arbeitsplatz bearbeitete Materialien und nicht für archivierte, d.h. bei externen Diensten lagernde Materialien. Für diese gelten andere Anforderungen, welche durch weitere, nicht in dieser Arbeit beschriebene Muster umgesetzt werden müssen.

6.1 Überblick über die Muster zur Werkzeugintegration

Ausgehend vom Umgebungsmuster wird als erstes Muster das *Umgebungsobjekt* eingeführt. Das Umgebungsobjekt realisiert den Schreibtisch, bindet den Arbeitsplatz an externe Dienste zur Materialverwaltung (z.B. Datenbanken) an, und erzeugt und koordiniert Werkzeuge. Das Umgebungsobjekt übernimmt diese Aufgaben, um gleichzeitig eine Abgrenzung nach außen zu vollziehen, für deren Beschreibung weitere Muster, insbesondere zur Materialverwaltung, benötigt werden.

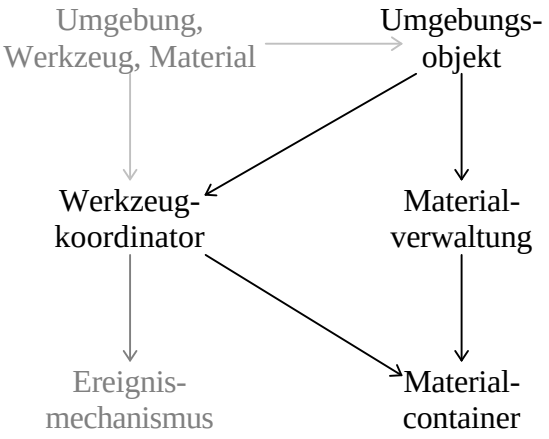


Abb. 6-1: Die vier Entwurfsmuster dieses Kapitels.

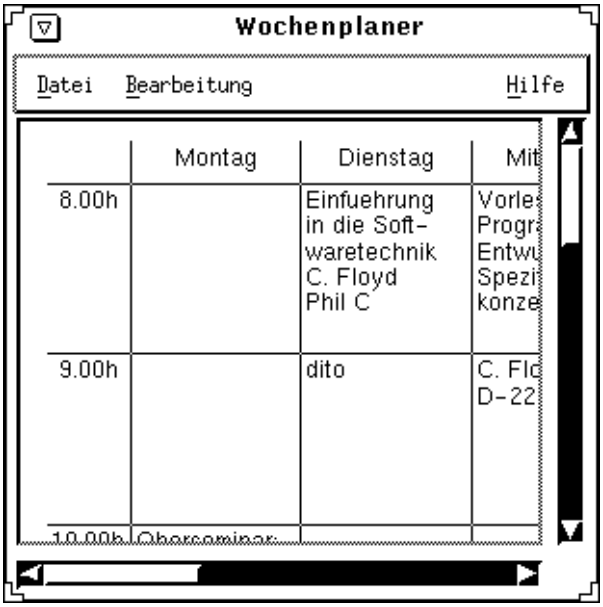


Abb. 6-2: Die Benutzungsschnittstelle des Wochenplaners.

Wie in Abb. 6-1 gezeigt, stützt sich das Umgebungsobjekt auf zwei weitere Muster ab. Das Muster des *Werkzeugkoordinators* übernimmt die Aufgabe, Werkzeuge zu aktualisieren, die

über voneinander abhängige Materialien gekoppelt sind. Ändert ein Werkzeug ein Material und ist dies für andere Materialien und andere Werkzeuge relevant, erfährt der Werkzeugkoordinator davon und informiert die abhängigen Werkzeuge.

Die *Materialverwaltung* ermöglicht mit Hilfe von *Materialversorgern* die Aktivierung und Passivierung von Materialien, sichert also ihre Persistenz. Sie bietet eine Anfrageschnittstelle und liefert Ergebnisse in *Materialsammlungen* und *Materialcontainern* zurück.

Materialcontainer enthalten logisch voneinander abhängige Materialien, welche aktiviert sind und von Werkzeugen benutzt werden. Diese Materialien werden mit Hilfe von Konsistenzbedingungsobjekten (*Constraints*) zusammengebunden. Konsistenzbedingungen werden als eigenständige Objekte formuliert und sorgen dafür, daß die von ihnen zusammengebundenen Materialien immer im konsistenten Zustand sind.

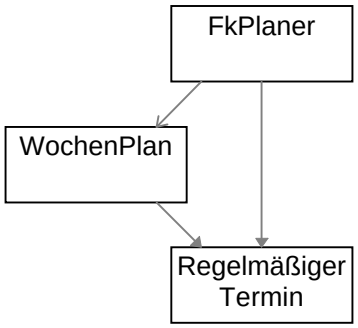


Abb. 6-3: Klassendiagramm für den Wochenplaner. Er arbeitet mit einem Wochenplan und Terminen.

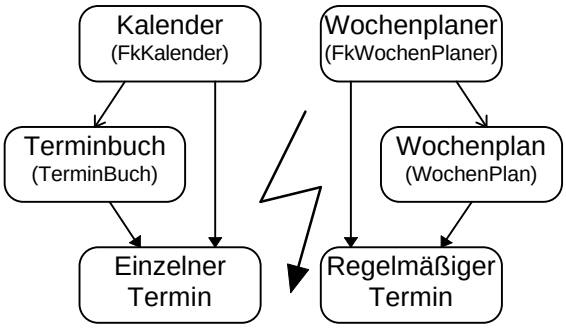


Abb. 6-4: Überblick über den Zusammenhang von Kalender und Wochenplaner.

Das Beispielszenario für dieses Kapitel ist wie folgt: Normale, d.h. einmalige, vom Kalender bearbeitete und verwaltete Termine können mit den regelmäßigen Terminen des Wochenplaners (Abb. 6-2) zeitlich überlappen. Für den Benutzer ist diese Überlappung eine relevante Information, da sie bedeutet, daß er einen Terminkonflikt zu klären hat.

Zur Vereinfachung der Diskussion wird der Terminkonflikt durch eine boolsche Variable ausgedrückt, welche in beiden Objekten gesetzt wird und einen Konflikt signalisiert. Ist sie im einmaligen Termin wie auch im regelmäßigen Termin gesetzt, so liegt ein Konflikt vor. Die Variable wird erst dann wieder zurückgesetzt, wenn der einmalige Termin verlegt wird.

6.2 Umgebungsobjekt

Das *Umgebungsobjekt* dient zur Umsetzung des Interpretationsmusters Umgebung. Es übernimmt mehrere Aufgaben. Das Umgebungsobjekt realisiert mit Hilfe eines *Schreibtischobjekts* das Anwenden von Werkzeugen auf Materialien, mit Hilfe der *Materialverwaltung* die Anbindung des Systems an externe Dienste und mit Hilfe von *Werkzeugkoordinatoren* die Koordinierung von Werkzeugen.

Problem Wie können die vom Umgebungsmuster formulierten Anforderungen an einen Softwarearbeitsplatz in einen konkreten Entwurf überführt werden?

Kontext Die Analyse der Einbettung und Handhabung von Werkzeugen und Materialien führte in Kapitel 4 zur Einführung des Umgebungsmuster. Sie bestimmt folgende Anforderungen, welche ein das Umgebungsmuster umsetzendes objektorientiertes Muster berücksichtigen muß:

- Werkzeuge und Materialien müssen räumlich und logisch den jeweiligen Verwendungszusammenhängen entsprechend anordbar sein, aufeinander angewendet und auch wieder weggelegt werden können.
- Die Umgebung muß die Anwendung von Werkzeugen auf Materialien auf ihre Ausführbarkeit hin kontrollieren.
- Ebenso soll die Umgebung als Hülle für den einzelnen Arbeitsplatz dienen, d.h. sie muß die Anbindung nach außen, an Archive respektive Datenbanken ermöglichen.

Hinzu kommt, daß Werkzeuge koordiniert werden müssen. Deswegen:

Lösung Konstruiere ein *Umgebungsobjekt*, welches die genannten Aufgaben an weitere Objekte delegiert. Dies sind: eine *Materialverwaltung*, ein *Schreibtischobjekt* sowie *Werkzeugkoordinatoren*. Verteile die Aufgaben nach den im folgenden genannten Richtlinien und integriere sie über das Umgebungsobjekt (Abb. 6-5).

Aufgabe der Materialverwaltung ist es, eine Anfrageschnittstelle für Materialien zur Verfügung zu stellen, welche von jedem Werkzeug genutzt werden kann. Dabei versteckt die Materialverwaltung vergleichbar einem Dienstvermittler („Servicebroker“) die Delegation der Anfrage an weitere *Materialversorger*, welche jeweils einen externen Dienstanbieter wie z.B. eine Datenbank kapseln. Die in einer Umgebung zur Verfügung stehenden Materialversorger werden vom Umgebungsobjekt ermittelt und als Objekte erzeugt. Sie werden der Materialverwaltung zur Systeminitialisierungszeit übergeben (Abb. 6-6).

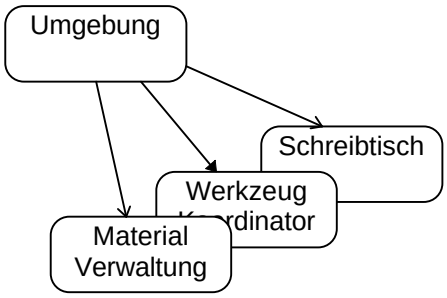


Abb. 6-5: Das Umgebungsobjekt delegiert Aufgaben an folgende Objekte: den Schreibtisch, Werkzeugkoordinatoren sowie die Materialverwaltung.

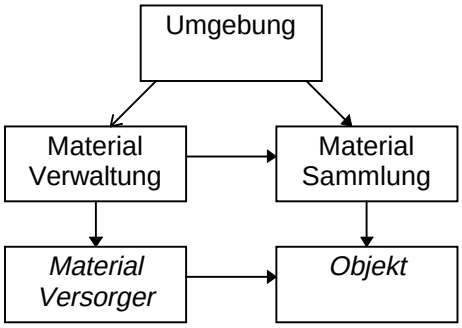


Abb. 6-6: Die Umgebung versieht die Materialverwaltung mit Materialversorgern. Sie nimmt Anfragen entgegen und setzt sie auf Materialversorger um.

Aufgabe des Schreibtischobjekts ist es, auf dem Bildschirm einen graphischen Schreibtisch mit Ikonen und direkter Manipulation darzustellen, welcher die Anwendung von Werkzeugen auf Materialien ermöglicht (Abb. 6-7).

Erfährt das Umgebungsobjekt vom Schreibtisch von der Anwendung eines Werkzeugs auf ein Material, so überprüft es dessen Ausführbarkeit mittels der vom Werkzeug geforderten und vom Material gebotenen Aspektklassen. Diese Überprüfung geschieht mit Hilfe eines hier nicht weiter diskutierten Metaobjekt-Protokolls, welches über ausreichende Metainformation verfügt, um eine solche Abfrage zu ermöglichen. Ist die Anwendung möglich, so erzeugt die Umgebung das Werkzeug.

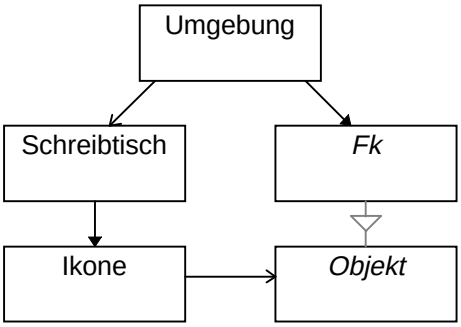


Abb. 6-7: Vom Schreibtisch erfährt die Umgebung, welches Objekt auf ein anderes anzuwenden ist und versucht, dies umzusetzen.

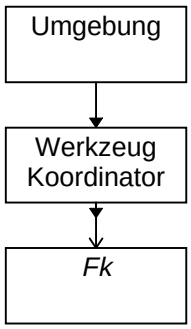


Abb. 6-8: Das Umgebungsobjekt delegiert die Koordination voneinander abhängiger Werkzeuge an Werkzeugkoordinatoren.

Ein Werkzeug wird einem Werkzeugkoordinator zugeordnet, der entweder bereits besteht oder neu erzeugt wird (Abb. 6-8). Aufgabe des Werkzeugkoordinators ist die Information abhängiger Werkzeuge von Zustandsänderungen der von ihnen bearbeiteten Materialien. Diese können ohne Wissen des Werkzeugs eintreten, weil entweder mehrere Werkzeuge dasselbe Material bearbeiten oder Konsistenzbedingungen zwischen Materialien vorerst ohne Wissen des Werkzeugs zur Änderung der Materialien führen.

Werkzeuge sind genau dann voneinander abhängig und werden vom selben Werkzeugkoordinator verwaltet, wenn sie voneinander abhängige Materialobjekte bearbeiten. Die Materialklassen bestimmen, ob Objekte dieser Klassen voneinander abhängig *sein können*. Aber erst konkrete Exemplare dieser Klasse, d.h. Objekte, *sind* voneinander abhängig. Konzeptuell sind die Klassen für einmalige Termine und regelmäßige Termine voneinander abhängig. Aber erst die konkreten Objekte im Terminbuch können von den Objekten im Wochenplan abhängig sein. Im Beispiel wird die Zuordnung über den Besitzer, d.h. den Benutzer vollzogen (Konsistenzbedingungen zwischen den Terminen zweier unterschiedlicher Benutzer werden also nicht betrachtet).

Die Spezifikation dieser Abhängigkeiten kann auf verschiedene Weisen geschehen; so ist etwa ein einfaches deklaratives Schema denkbar. Die im Anwendungsrahmen gewählte Lösung ist, eine Analyse der ins System eingebundenen Konsistenzbedingungsobjekte vorzunehmen und Materialien dem Analyseergebnis zufolge zu gruppieren. Werkzeugkoordinatoren gruppieren dann Werkzeuge nach den vollzogenen Materialgruppierungen. Die nächsten Muster schildern dies im Detail.

Vergleiche Viele der aufgeworfenen Fragen werden im Rahmen der Standardisierung von Werkzeug- und Datenintegrationsdiensten, so z.B. PCTE [Tho89, TN92, WJ93], behandelt. Eine Integration der verschiedenen Aspekte findet notwendig Anwendung im Entwurf von Softwareentwicklungsumgebungen [Bis92, BKS94, BKM+94, MNL+93].

6.3 Materialverwaltung

Die Materialverwaltung bietet eine objektorientierte Anfrageschnittstelle für zu verwaltende und zu aktivierende Materialien. Die Anfragen werden mit Hilfe von Materialversorgern umgesetzt, welche jeweils einzelne externe Dienste kapseln. Anfrageergebnisse werden an den Anfragenden, üblicherweise ein Werkzeug, in einer Materialsammlung zurückgegeben. Werkzeuge können dann explizit Materialien entnehmen oder fallen lassen.

Problem Softwarewerkzeuge werden nicht nur auf einmal ausgewählte Materialien angewandt, sondern müssen auf Benutzungswunsch hin oder auch eigenständig weitere Materialien von externen Diensten, z.B. Datenbanken, aktivieren und bearbeiten.

Kontext In jedem Büro und erst recht in der durch EDV ermöglichten Massendatenverarbeitung werden Materialien nicht nur am einzelnen Arbeitsplatz gelagert, sondern in externen Archiven respektive Datenbanken. Hieraus folgen mehrere Anforderungen an die einfache Umsetzung eines Datenbankbegriffs im Rahmen der Werkzeug und Material Metapher:

- Materialien müssen effizient verwaltet werden, so daß sie persistent gehalten und wiedergefunden werden können.
- Materialien müssen über eine ausreichend mächtige Anfrageschnittstelle spezifiziert und aktiviert werden können.
- Die Anfrageschnittstelle muß möglichst einheitlich sein und, sofern nicht explizit anders gewünscht, die Delegation der Aufgabe an verschiedene Dienste verstecken.
- Die unterschiedlichen Dienste müssen einheitlich genutzt werden können. Relationale, objektorientierte oder hostbasierte hierarchische Datenbanken müssen zugreifbar sein.

Hierbei wird die Einschränkung vorgenommen, die Inanspruchnahme spezifischer Dienste nicht explizit zu machen, sondern alle gleich zu behandeln. Somit ist folgende Lösung möglich:

Lösung Erstelle zur Systeminitialisierungszeit genau ein Objekt, welches als einheitliche Schnittstelle zu den externen Diensten fungiert und Anfragen erlaubt: die *Materialverwaltung*. Übergebe diesem Objekt mehrere *Materialversorger*, welche jeweils einen einzelnen externen Dienst verbergen. Liefere als Ergebnis von Anfragen *Materialsammlungen* zurück, welche alle verfügbaren und der Anfrage entsprechenden Materialien enthalten.

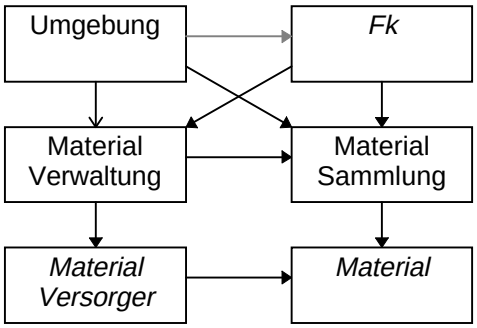


Abb. 6-9: Die Umgebung versieht die Materialverwaltung mit Materialversorgern. Sie erhält Materialien in Materialsammlungen zurück.

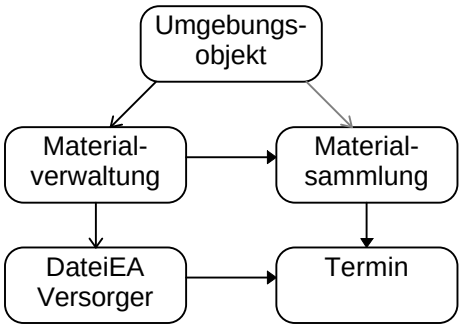


Abb. 6-10: Im Anwendungsrahmen dient ein einfacher Datei-E/A basiert Materialversorger zum Aktivieren von Objekten.

Das Umgebungsobjekt erzeugt gleichermaßen die Materialverwaltung als auch die Materialversorger. Letztere übergibt sie der Materialverwaltung. Damit bleibt das Wissen um die Außenwelt beim Umgebungsobjekt und hinter den Materialversorgern gekapselt.

Jedes neue Werkzeug erhält einen Verweis auf die zu verwendende Materialverwaltung. Es kann dann Materialanfragen an diese richten. Die Ergebnisse erhält es in einer Materialsammlung zurück, welche jene aktivierten Materialien enthält, die auf die Anfrage passen. Die Materialverwaltung übernimmt alle Verwaltungsaufgaben und entlastet somit die Werkzeuge. Sie delegiert die Verwaltungsaufgaben, sofern möglich, an die Materialversorger.

Die Materialsammlung ermöglicht das Sondieren der Materialien. Will ein Werkzeug dann mit einem Material arbeiten, muß es dieses aus der Materialsammlung explizit entnehmen („auschecken“). Dies geschieht über die Materialverwaltung, welche das Material entweder als Original (mit dem Recht zu ändern) oder als Kopie (nur zur Sondierung) herausreicht.

Vergleiche Anfragesprachen, wie sie die Materialverwaltung ermöglichen und umsetzen muß, werden im Rahmen von objektorientierten Datenbanken diskutiert [KL89, Ban93, Cat93]. Die Verteilung der verschiedenen Dienste auf Materialversorger entspricht dem „Service“ Gedanken, ihre Koordinierung durch die Materialverwaltung dem „Servicebroker“ oder auch „Object Request Broker“ des CORBA Standards [OMG91, OMG92]. Das hier vorgestellte Muster der Materialverwaltung für die Werkzeug und Material Metapher wird detailliert in [WW94] behandelt.

6.4 Materialcontainer

Ein zur Bearbeitung aktiviertes Material befindet sich in einem oder mehreren *Materialcontainern* und wird von *Konsistenzbedingungsobjekten* überwacht. Die Konsistenzbedingungsobjekte führen bei Veränderung eines Materials zu Veränderungen abhängiger Materialien.

Problem Materialien hängen voneinander ab. Um die Konsistenz eines laufenden Systems zu wahren, müssen Änderungen eines Materials zu kontrollierten Änderungen der abhängigen Materialien führen.

Kontext An jedem Softwarearbeitsplatz gibt es mehrere gleichzeitig bearbeitete Materialien, welche Werkzeuge von der Materialverwaltung erfragen und explizit entnehmen. Zwischen diesen Materialien kann es Abhängigkeiten geben, welche es notwendig machen, Kontrollinstanzen für diese Abhängigkeiten einzuführen.

Wie im Überblick geschildert, enthalten sowohl einzelne als auch wöchentliche Termine eine boolsche Variable, welche darüber Auskunft gibt, ob sich zwei Termine überlappen. Können mehrere Termine konfligieren, welche von verschiedenen Werkzeugen bearbeitet werden, so liegt die Verantwortung zur Feststellung und (möglichen) Auflösung des Konflikts außerhalb eines einzelnen Werkzeugs.

Da Materialien unabhängig von spezifischen Werkzeugen zu denken sind, muß eine Kontrollinstanz für Materialien unabhängig von Werkzeugen und unter ausschließlicher Bezugnahme auf Materialien entworfen werden. Deswegen:

Lösung Führe für jede Kombination unterschiedlicher voneinander abhängiger Materialien ein *Konsistenzbedingungsobjekt* ein. Seine Aufgabe ist, die Konsistenzbedingungen zwischen Materialien aufrecht zu erhalten. Gruppiere die abhängigen Materialien in einem *Materialcontainer* und parametrisiere diesen mit dem Konsistenzbedingungsobjekt. Informiere den Materialcontainer darüber, daß Änderungen stattgefunden haben, so daß er das Konsistenzbedingungsobjekt aktivieren kann.

Wenn ein Werkzeug ein neues Material von der Materialverwaltung verlangt und erhalten hat, muß dieses in einen oder mehrere Materialcontainer eingefügt werden. Ein Materialcontainer gruppiert abhängige Materialien. Sind bereits Materialien in aktiver Bearbeitung, welche vom neuen Material abhängig sind (und umgekehrt), so wird das neue Material in die bereits existierenden Materialcontainer eingefügt. Bestehen keine Abhängigkeiten zu aktivierten Materialien, so wird ein neuer Materialcontainer erzeugt.

Die Ermittlung von Abhängigkeiten zwischen Materialien und die Sicherstellung ihrer Konsistenz geschieht mit Hilfe von Konsistenzbedingungsobjekten. Die Materialverwaltung sammelt

alle Klassen für jene Konsistenzbedingungen ein, welche sich auf das zu aktivierende Material beziehen. Dann findet ein Abgleich mit bereits existierenden Konsistenzbedingungsobjekten statt, welcher zur Einordnung des Materials in einen existierenden Container oder zur Erzeugung eines neuen führt. Ein neuer Container wird mit dem Konsistenzbedingungsobjekt parametrisiert, welches für die in ihm zu gruppierenden Materialien zuständig ist.

Wird ein Material von einem Werkzeug verändert, so wird über den Materialcontainer das Konsistenzbedingungsobjekt aktiviert, welches das veränderte Material analysiert und die abhängigen Materialien in einen konsistenten Zustand versetzt. Die Benachrichtigung des Materialcontainers über Änderungen eines in ihm enthaltenen Materials ist auf verschiedene Arten ausführbar. Folgende Varianten sind technisch möglich:

- Das Werkzeug kann nach Änderung des Materials dies dem Container explizit mitteilen.
- Der Materialcontainer kann die Zugriffe auf Materialien überwachen.
- Die Materialien können via Ereignismechanismus den Container selbst benachrichtigen.

Die Erfahrung hat gezeigt, daß der Einsatz des Ereignismechanismus so sparsam wie möglich geschehen sollte. Deswegen wurde die erste Alternative gewählt. Jeglicher Einsatz des Ereignismechanismus bedeutet immer die Gefahr, Anforderungen des Kontexts in die beobachteten Objekte hineinzutragen. Deswegen wurde, wie bereits erläutert, der Ereignismechanismus nur für Werkzeuge eingesetzt.

Abb. 6-11 zeigt die allgemeine Situation, in dem die Funktionskomponente eines Werkzeugs ein von Konsistenzbedingungsobjekten kontrolliertes Material benutzt. In der Abbildung zusätzlich dargestellt ist der Materialcontainer, der von der Fk über Materialänderungen explizit benachrichtigt wird.

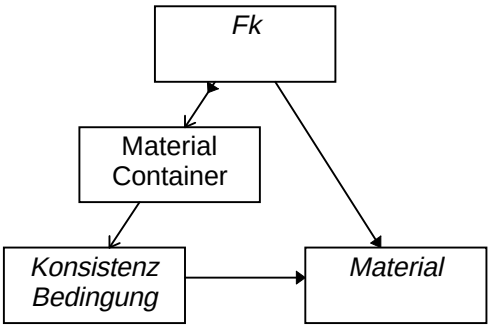


Abb. 6-11: Das allgemeine Schema für Fk's und Materialien, welche in Materialcontainern gehalten und von Konsistenzbedingungen überwacht werden.

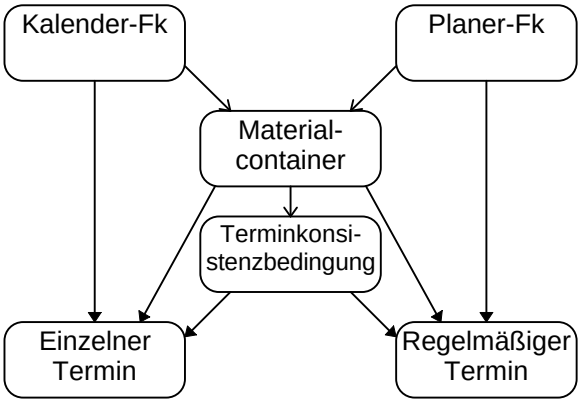


Abb. 6-12: Im Kalender und Wochenplaner Beispiel sorgt eine Konsistenzbedingung für Termine für die Überprüfung von Terminkonflikten.

Für die im Kalender und im Wochenplaner zu bearbeitenden Termine wird ein Terminkonsistenzbedingungsobjekt eingeführt (Abb. 6-12). Es überwacht normale und regelmäßige Termine, die aus voneinander abhängigen Terminbüchern und Wochenplänen (desselben Benutzers) stammen.

6.5 Werkzeugkoordinator

Zu jedem Materialcontainer mit Konsistenzbedingungsobjekt gehört ein Werkzeugkoordinator. Wird ohne Wissen des bearbeitenden Werkzeugs ein Material geändert, so wird es vom zum Materialcontainer gehörenden Werkzeugkoordinator benachrichtigt und kann seinen Zustand am neuen Materialzustand ausrichten.

Problem Der Zustand und die Materialpräsentation eines Werkzeugs kann inkonsistent werden, wenn die von ihm bearbeiteten Materialien aufgrund einer Konsistenzbedingung ohne sein Wissen geändert werden.

Kontext Durch Einführung des Materialcontainers und der Konsistenzbedingungsobjekte wurde die Konsistenz von Abhängigkeiten zwischen Materialien sichergestellt. Wird eines mehrerer voneinander abhängiger Materialien geändert, so sorgt das überwachende Konsistenzbedingungsobjekt dafür, daß die abhängigen Materialien ebenfalls geändert werden.

Ändert sich ein von einem Werkzeug bearbeitetes Material, dann kann der Werkzeugzustand und somit auch seine Benutzungsschnittstelle inkonsistent werden. Ebenso wird die Präsentation des Materials aktualisiert werden müssen. Deswegen:

Lösung Erzeuge zu jedem Materialcontainer mit Konsistenzbedingungsobjekt einen Werkzeugkoordinator, welcher die über ihre Materialien gruppierten Werkzeuge koordiniert. Wird ein Material geändert, so sorge mit Hilfe des Werkzeugkoordinators dafür, daß abhängige Werkzeuge informiert werden.

Wird von der Materialverwaltung ein neuer Materialcontainer angelegt, so erzeugt das Umgebungsobjekt automatisch einen dazugehörigen Werkzeugkoordinator. Dieser analysiert die Zuordnung von Werkzeugen zu Materialien und erstellt eine einfache Abhängigkeitstabelle. Erfährt der Werkzeugkoordinator von Änderungen eines Materials, informiert er auf Basis dieser Tabelle die abhängigen Werkzeuge.

Alternativ verzichtet der Werkzeugkoordinator auf eine Analyse der Werkzeug/Material Beziehungen und leitet lediglich Aktualisierungsereignisse zwischen allen Werkzeugen weiter, welche auf einem bestimmten Materialcontainer arbeiten. Der Werkzeugkoordinator würde dann zu einem reinen Ereignisverteiler („Event Dispatcher“) degenerieren.

Wiederum sind mehrere Varianten denkbar, den Informationsfluß zwischen Werkzeug und Werkzeugkoordinator bzgl. Materialzustandsänderungen zu realisieren:

- Das Werkzeug teilt Änderungen dem Materialcontainer explizit mit. Dann kann es auch via Ereignismechanismus den Werkzeugkoordinator informieren.
- Der Materialcontainer kann den Werkzeugkoordinator über den Ereignismechanismus benachrichtigen.
- Und ebenfalls kann der Werkzeugkoordinator zugleich die Materialien beobachten, um über allgemeine Änderungen sofort informiert zu werden.

Wie bereits erläutert, ist auch hier die erste Variante zu befürworten, weil sie den Ereignismechanismus am sparsamsten einsetzt.

Abb. 6-13 zeigt die allgemeine Struktur sowie die 1 zu 1 Zuordnung von Werkzeugkoordinatoren zu Materialcontainern auf. Abb. 6-14 zeigt das Objektdiagramm für das Beispiel des Terminkalenders und Wochenplaners: Beide Werkzeuge werden aufgrund der Gruppierung ihrer Materialien in einen Materialcontainer auch unter einen Werkzeugkoordinator eingeordnet. Wird z.B. im Kalender ein neuer Einzeltermin eingegeben, welcher mit einem wöchentlichen Termin überlappt, so sorgt das in der Abbildung nicht angezeigte Terminkonsistenzbedin-

gungsobjekt für das Setzen des Konfliktflags in den beiden Terminobjekten. Der Werkzeugkoordinator erfährt von der Änderung der beiden Terminobjekte und informiert die Werkzeuge. Diese sorgen dann für eine entsprechende Aktualisierung ihres Zustands und ihrer Benutzungsschnittstelle.

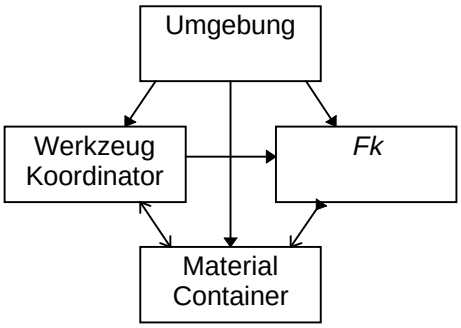


Abb. 6-13: Die Umgebung ordnet jedem Materialcontainer einen Werkzeugkoordinator zu, welcher die auf den Materialien arbeitenden Werkzeuge koordiniert.

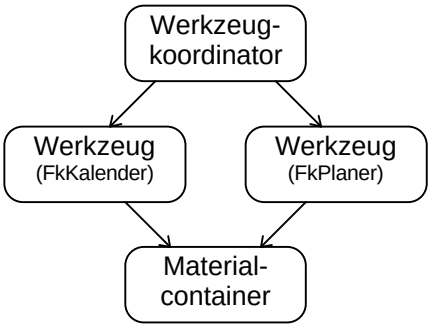


Abb. 6-14: Der Werkzeugkoordinator erhält Änderungsereignisse (von Werkzeugen oder vom Materialcontainer) und verteilt sie auf die abhängigen Werkzeuge.

Vergleiche Der Materialcontainer nimmt mit Hilfe des Konsistenzbedingungsobjekts notwendig gewordene Änderungen an Materialien vor, während der Werkzeugkoordinator die Vermittlung des Änderungsereignisses übernimmt. Das Konzept des *Mediators* [SN92, GHJ+95] dient zur Integration voneinander unabhängiger Objekte, z.B. von Werkzeugen, und ähnelt somit dem hier vorgestellten Konzept. Er übernimmt beide Aufgaben, indem er bei abhängigen Objekten die notwendig gewordene Änderung selbst vornimmt.

Für den hier vertretenen Ansatz wäre dies problematisch, da Mediatoren nicht nur für bestimmte Materialien, sondern auch für die sie bearbeitenden Werkzeuge maßgeschneidert werden müßten, da sie nur mit ihrer Hilfe die notwendig gewordenen Änderungen an Materialien ausführen lassen können. Dies macht sie schwer wiederverwendbar, da sie gleichermaßen auf Werkzeug- wie Materialstrukturen aufsetzen müssen.

Die Entzerrung des Problems in Werkzeugkoordinatoren, welche lediglich Änderungsereignisse verteilen, sowie Konsistenzbedingungsobjekte, die unabhängig von Werkzeugen abhängige Materialien konsistent halten, lagert das Wissen um diese Abhängigkeiten bei den betroffenen Objekten selbst an. Das Konsistenzbedingungsobjekt kontrolliert Materialien und nur Materialien. Der Werkzeugkoordinator verteilt lediglich anonyme Änderungsereignisse.

7 Diskussion und Abschluß

Dieses Kapitel versucht zu klären, ob die Verwendung von Mustern zur Dokumentation und Kommunikation von Softwareentwürfen ihrem Anspruch gerecht wird. Dazu wird das Zeitplanungssystembeispiel noch einmal aufgearbeitet und das Ergebnis kritisch betrachtet. Im Anschluß wird die Arbeit als Ganzes reflektiert. Offengebliebene Fragen und Wege der Fortführung werden aufgezeigt. Ein Resümee schließt die Arbeit ab.

7.1 Anwendung der Muster

Dieser Abschnitt greift Abschnitt 2.3 wieder auf, in dem ein Szenario und eine Handhabungsvision für das Zeitplanungssystem gegeben wurden. Hier wird die Handhabungsvision erneut niedergeschrieben, diesmal aber unter Verwendung der durch die Muster etablierten Terminologie. Es folgt eine ebenfalls auf den Mustern basierende Entwurfsbeschreibung, welche die beispielhaften Auszüge der letzten Kapitel zu einem kohärenten Ganzen zusammenfaßt.

7.1.1 Handhabungsvision des Kalenders

Der Benutzer möchte einen Konzerttermin in sein Terminbuch eingeben. Er findet sein Terminbuch für private Termine im Zeitplanungsordner. Durch Doppelklick auf das Material startet er das für das Material vordefinierte Werkzeug, den Kalender. Das Kalenderwerkzeug zeigt somit zu Beginn das private Terminbuch an.

Der Benutzer tippt in das vom Kalender dafür vorgesehene linke obere Eingabefeld das Datum für den Termin. Er weiß, daß das Werkzeug diese Eingabe in ein neues Material, einen einmaligen Termin, umsetzen wird. Mittels Tabulator setzt der Benutzer den Eingabefokus auf das nächste Feld, in das er einen Titel für den neuen Termin tippt. Daraufhin setzt der Benutzer den Eingabefokus mittels Mausklick auf den sich unterhalb des Titeleingabefelds befindlichen Editor. Dies wird vom Kalender als Bestätigung des neuen Termins gewertet.

Die Bestätigung eines neuen Termins führt zu seiner Erzeugung durch das Werkzeug. Der Termin erscheint zur Linken des Editors in einem Auflister, welcher alle Termine des Terminbuchs chronologisch sortiert anzeigt. Ein kurzer Blick des Benutzers genügt ihm, um sich zu vergewissern, daß das eingegebene Datum, Uhrzeit und Titel stimmen und richtig einsortiert wurden.

Der Benutzer gibt nun im Editor seine Erläuterungen zum Termin ein: Ort, Anlaß sowie einen Hinweis, den Studierendenausweis nicht zu vergessen, um an günstige Eintrittskarten zu gelangen. Mit Verlassen des Editierfeldes des Editors wird der eingegebene Text im eigentlichen Termin gesetzt.

Da keine weiteren Aufgaben zu erledigen sind, beendet der Benutzer die Arbeit durch Doppelklick auf den Schließenknopf des Kalenderfensters. Er bestätigt die Rückversicherung, den

neu eingegebenen Termin zu sichern, positiv. Kalender samt Fenster verschwindet und der Benutzer sieht sich wieder dem Schreibtisch gegenüber.

7.1.2 Entwurf des Zeitplanungssystems

Zur Beschreibung des Zeitplanungssystems wird die statische Struktur vom dynamischen Aufbau und Verhalten getrennt.

7.1.2.1 Struktur des Zeitplanungssystems

Das Zeitplanungssystem besteht aus zwei Werkzeugen, einem Kalender und Editor, sowie verschiedenen Materialien, einem Terminbuch mit Einzelterminen und einem Stundenplan mit regelmäßigen Terminen. Aspekte des Arbeitens mit den Materialien sind:

- Das Terminbuch muß über einen Index verfügen und besitzt deswegen den Aspekt Indizierbar. Über den Index können auflistbare Materialien erfragt werden.
- Auch der Wochenplan ist indizierbar und liefert auflistbare Materialien zurück. Die Ordnung des Index basiert auf der zeitlichen Ordnung zwischen den Terminen.
- Einzeltermine wie auch regelmäßige Termine können als Einträge in Listen verwendet werden und besitzen somit den Aspekt Auflistbar. Zudem können sie textuell editiert werden, besitzen also den Aspekt Editierbar.

Jeder Teilarbeitstätigkeit, jedem Aspekt, wird eine Aspektklasse zugeordnet. Für jede Aspektklasse gibt es ein einfaches Werkzeug, welches elementare Handhabungen ermöglicht und im Rahmen eines Kontextwerkzeugs wiederverwendet werden kann. Das einfache Werkzeug Auflister stellt die auflistbaren Materialien eines indizierbaren Materials (Einzeltermine im Terminbuch) in einer Liste zur Auswahl bereit. Der Editor präsentiert ein editierbares Material zum möglichen textuellen Editieren (Abb. 7-1).

Auflister und Editor sind Subwerkzeuge des Werkzeugs Kalender (Abb. 7-2). Sie bestehen jeweils aus genau einer Interaktions- und einer Funktionskomponente. Die Interaktionskomponente des Auflisters präsentiert eine Liste auf dem Bildschirm, die des Editors ein mehrzeiliges Editierfeld. Das Kalenderwerkzeug bietet weitere Benutzungsschnittstellenelemente zur Eingabe von Terminen an, welche es wie auch die Benutzungsschnittstelle der Subwerkzeuge innerhalb eines Hauptfenster mit Menüleiste anordnet (Abb. 7-3).

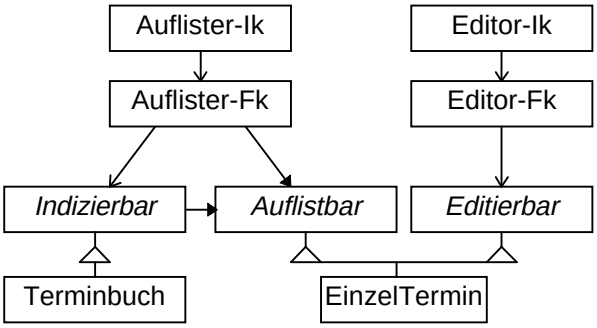


Abb. 7-1: Die Struktur der Einfachwerkzeuge Auflister und Editor sowie ihre Aspekte.

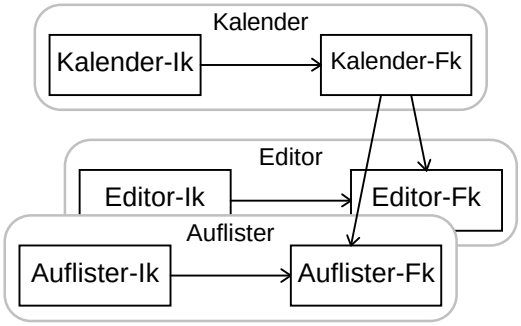


Abb. 7-2: Die Kalenderwerkzeugstruktur mit den Subwerkzeugen Auflister und Editor.

Der Wochenplaner wird ebenfalls aus den Subwerkzeugen Auflister und Editor aufgebaut, nur mit dem Unterschied, daß diese eine andere (kompliziertere) Benutzungsschnittstelle besitzen (Abb. 7-4). Bei der Konstruktion des Wochenplaners werden deswegen die Interaktionskom-

ponenten der Subwerkzeuge durch solche ersetzt, welche die Auswahl und das Editieren innerhalb einer Zeichenfläche ermöglichen. Ein Feld auf der Zeichenfläche entspricht einem Eintrag im Wochenplan. Das Editieren des Feldes entspricht dem textuellen Editieren eines regelmäßigen Termins.

Die Funktionskomponenten von Auflister und Editor können wiederverwendet werden. Die Klassenstruktur entspricht den Abb. 7-1 und 7-2, nur mit dem Wochenplanerwerkzeug anstelle des Kalenders sowie neuen Auflister- und Editorinteraktionskomponenten.

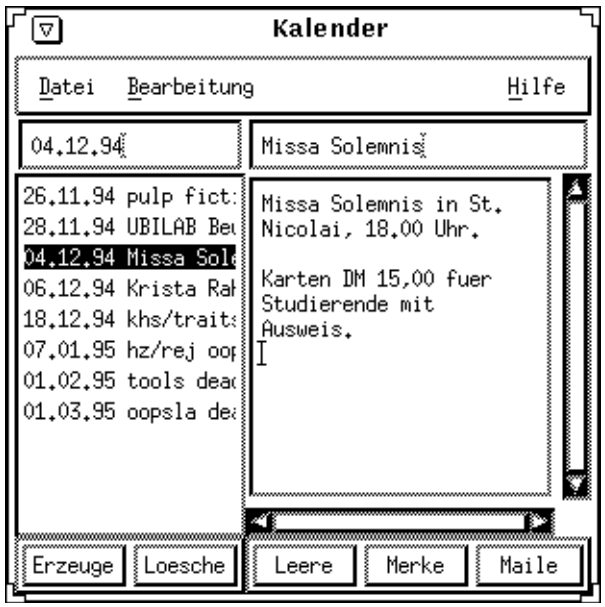


Abb. 7-3: Die Benutzungsschnittstelle des Kalenderwerkzeugs.

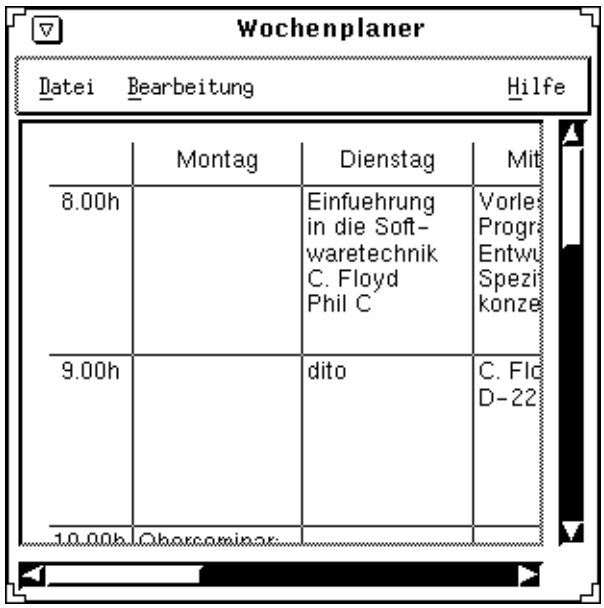


Abb. 7-4: Die Benutzungsschnittstelle des Wochenplaners.

Beide Werkzeuge, Kalender wie Wochenplaner, sowie die Materialien, Terminbuch wie Wochenplan, stehen Benutzern auf ihrem softwaretechnischen Schreibtisch zur Verfügung. Der Schreibtisch präsentiert sie als Ikonen, welche Benutzer bewegen und aktivieren können.10 Er ermöglicht die Anwendung von Werkzeugen auf Materialien.

Die Umgebung erzeugt den Schreibtisch und überprüft die Anwendung von Werkzeugen auf Materialien. Sie verwaltet aktivierte Werkzeuge und erzeugt Werkzeugkoordinatoren und Konsistenzbedingungsobjekte, sofern diese benötigt werden. Die Klassenstruktur ist in Abb. 7-5 dargestellt. Materialverwaltung und Materialversorger wurden in der Abbildung nicht aufgeführt.

Sind beide Werkzeuge aktiviert, so werden sie von einem Werkzeugkoordinator überwacht, welcher sie über Änderungen an ihrem Material informiert. Der Werkzeugkoordinator arbeitet nur mit abstrakten Oberklassen, ist also unabhängig von spezifischen Werkzeugen und Materialien. Materialcontainer und Konsistenzbedingungsobjekt übernehmen die Aufgabe, abhängige Materialien zusammenzufassen und ihre Konsistenz zu sichern.

---

[10] Ikonen werden als Umwickler realisiert. Es wäre zwar denkbar gewesen, Werkzeuge und Materialien direkt mit einer weiteren Aspektklasse Ikonifizierbar zu versehen, hätte aber deren Klassenschnittstelle zu sehr aufgeblasen. Die Unterscheidung Aspektklassenkomposition versus Umwicklerkomposition wird pragmatisch getroffen (siehe 5.3): Auflistbarkeit und Editierbarkeit sind intrinsische, unabdingbare Eigenschaften des Materials EinzelTermin, seine Darstellung auf einem Schreibtisch als Ikone ist es nicht. Die Position, Auswahlzustand und graphische Repräsentation des Materials als Ikone braucht nicht im Material gespeichert zu werden.

Die Konsistenzbedingung im Zeitplanungssystem besteht darin, daß ein Flag in Einzelterminobjekten gesetzt werden muß, wenn der Termin zeitlich mit einem regelmäßigen Termin überlappt. Beide Materialtypen werden von unterschiedlichen Werkzeugen bearbeitet und können deswegen nicht von einem einzelnen Werkzeug koordiniert werden.

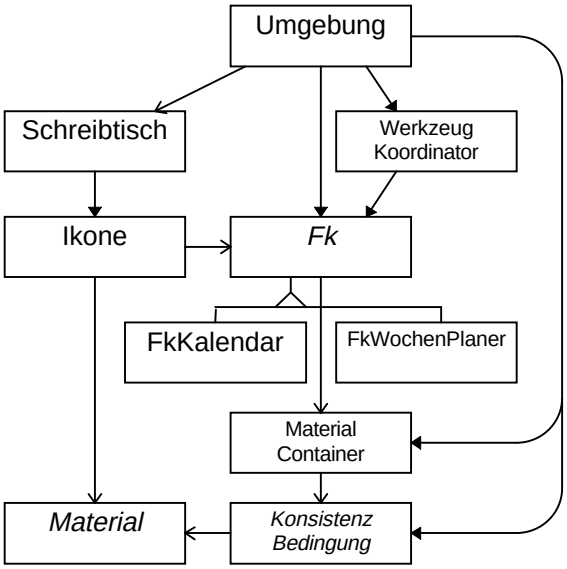


Abb. 7-5: Die Umgebung kontrolliert die Systemstruktur im Großen. Sie arbeitet nur mit abstrakten Oberklassen.

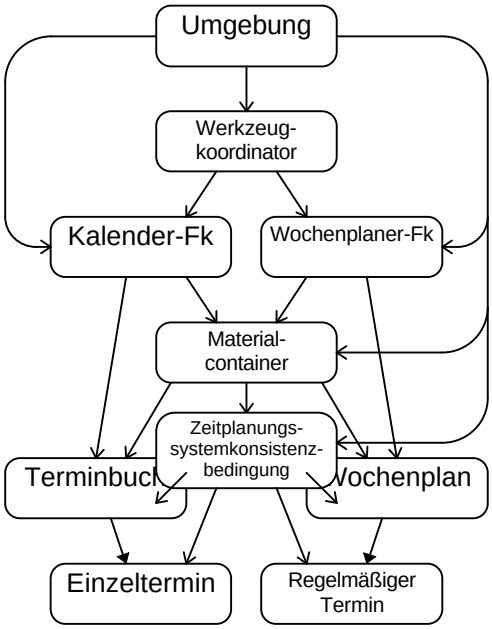


Abb. 7-6: Objektstruktur des Zeitplanungssystems bei gleichzeitig aktiviertem Kalender und Wochenplaner.

7.1.2.2 Dynamik des Zeitplanungssystems

Wird der Kalender vom Benutzer aktiviert, so erzeugt die Umgebung das Werkzeug. Wird nun der Wochenplaner aktiviert, so muß zusätzlich zum Werkzeug noch ein Werkzeugkoordinator sowie ein Konsistenzbedingungsobjekt erzeugt werden.[11] Die Umgebung baut die in Abb. 7-6 gegebene Objektstruktur auf.

Das Konsistenzbedingungsobjekt des Zeitplanungssystems überwacht das Terminbuch und den Wochenplan sowie die in ihnen enthaltenen Termine. Wird ein neuer Termin im Kalender eingegeben, so erzeugt die Kalender-Fk ein neues Einzelterminobjekt und fügt es in das Terminbuch ein. Gleichzeitig stößt es den Materialcontainer an, welcher seinerseits die Konsistenzbedingung aktiviert. Diese überprüft das übergebene Material und führt möglicherweise notwendig gewordene Änderungen an den Materialien durch. Die Konsistenz ist somit immer sichergestellt. Der Materialcontainer merkt sich die in Änderungen involvierten Materialien.

Die Fk, welche ursprünglich Auslöser der ersten Änderung war, signalisiert dies durch ein einfaches Ereignis an den Werkzeugkoordinator. Dieser erfragt vom Materialcontainer die veränderten Materialien und ermittelt anhand von Material zu Werkzeug Zuordnungstabellen jene Fk's, die auf Materialien arbeiten, welche gerade geändert wurden. Er informiert diese dann über die Änderungen, so daß sich das betroffene Werkzeug in seinem sichtbaren und unsichtbaren Zustand am neuen Materialzustand ausrichten kann.

---

[11] Wie in Kapitel 6 geschildert, wird hierbei davon ausgegangen, daß aktivierte Materialien zu Beginn einer Sitzung immer konsistent sind. Abhängigkeiten zwischen aktivierten und passivierten Materialien werden in dieser Arbeit nicht betrachtet.

7.2 Diskussion der Musteranwendung

Im folgenden wird die Anwendung der Muster für das Zeitplanungssystem betrachtet. Dabei ist die Handhabungsvision vom Entwurfsdokument zu unterscheiden. Zu betrachten ist, in welchem Maße neue Begriffe in die Dokumente Eingang gefunden haben und ob es relevante strukturelle Veränderungen in den sprachlichen Beschreibungsformen gegeben hat.

7.2.1 Diskussion der Handhabungsvision

Der Einsatz von Mustern für die Handhabungsvision hat zu keinen augenscheinlichen Veränderungen geführt. Obwohl sie zeitlich unabhängig voneinander geschrieben wurden, sind sich beide Texte sehr ähnlich. Die zweite Niederschrift der Handhabungsvision in diesem Kapitel läßt kaum die inzwischen erarbeitete Terminologie erkennen. Der Sprachgebrauch betont zwar etwas mehr den Werkzeug- und Materialcharakter der diskutierten Begriffe, sprachliche Veränderungen selbst aber scheint es nicht gegeben zu haben.

Nach Einführung der verschiedenen an den Arbeitstätigkeiten orientierten Muster wäre anzunehmen gewesen, daß Begriffe wie Verwendungszusammenhang, Aspekt und Werkzeug stärker aufgegriffen worden wären. Da eine Handhabungsvision aber auf den Umgang mit einer Benutzungsschnittstelle ausgerichtet ist und nicht so sehr auf die Struktur der Arbeitstätigkeiten, war dies vielleicht dann doch nicht zwingend.

Zu bemerken ist noch, daß an jenen Stellen, wo von einem „indizierbaren Material, welches weitere auflistbare Materialien enthält“ hätte gesprochen werden können, der sehr viel einfachere (und weniger künstliche) Begriff der Liste verwendet wird. Dies betont die in dieser Arbeit nicht weiter thematisierten Interaktionstypen, welche durch die ihnen eigene Darstellung und Handhabung bereits die intendierten Umgangsformen und Aspekte der präsentierten Materialien reifizieren. Hierzu sind weitere Arbeiten im Gange [Stu95, Frö95].

7.2.2 Diskussion des Entwurfsdokuments

Der Einsatz der durch die Muster etablierten Terminologie im Entwurfsdokument ist durchgängig. Es können folgende Beobachtungen gemacht werden:

- Die neu eingeführten Begriffe fügen sich als Eigennamen naht- und problemlos in den Text ein.
- Die verwendeten Begriffe sind sehr viel präziser geworden und ihr Umfang hat beträchtlich zugenommen.
- Es findet ein ständiger Wechsel zwischen den allgemeinen Begriffen (Werkzeug) und ihren spezifischen Ausprägungen statt.
- Die eigentlichen Musternamen tauchen im Text nur dann auf, wenn sie eine Komponente bezeichnen (Werkzeug, Materialcontainer) aber nicht, wenn sie eine Beziehung benennen (Trennung von Interaktion und Funktion, Werkzeug Material Kopplung).
- Taucht ein Beziehungsverhältnis doch einmal auf, so wurde das die Beziehung ausdrükkende Substantiv (Kopplung, Trennung, Unterscheidung) in das passende Verb (koppeln, trennen, unterscheiden) umgesetzt und in den Satzbau integriert.
- Es entsteht der Eindruck des Arbeitens mit einem wohldefinierten *Begriffsgerüst*, nicht aber einer *Sprache*.
- Während des Schreibens und Lesens wird implizit auf Konzepte verwiesen, die sofern nicht weiter ausgeführt, trotzdem sehr präzise hätten niedergeschrieben werden können.

Das entscheidende Wort lautet „Begriffsgerüst“. Augenfällig ist die Durchsetzung des Dokuments mit Begriffen, welche komponentenartigen Charakter besitzen. Durch die detailliert ausgearbeiteten Muster wurden diese Begriffe etabliert und die Beziehungen zwischen ihnen definiert. Diese Komponenten lassen sich zumeist direkt auf Klassen und Objekte abbilden, ihre Beziehungsstruktur in ein Klassendiagramm.

Praktisch nicht zum Tragen gekommen ist der Beziehungscharakter von Mustern. Betonte ein Muster in seinem Namen einmal eine Beziehung (Trennung von ..., Kopplung von ...), so tauchte diese gelegentlich als Verb wieder auf, hatte aber keinen großen Einfluß auf die sich im Kopf entfaltende Struktur des diskutierten Entwurfs. Wollen die in dieser Arbeit beschriebenen Muster eines Tages Sprachcharakter gewinnen, so müssen sie zumindest so umgeschrieben werden, daß sie ihre Beziehungsstrukturen in den Vordergrund stellen. Ein Gerüst aufeinander Bezug nehmender Begriffe ist nicht ausreichend, den Begriff Sprache zu rechtfertigen.

Einer der großen Vorteile, welcher sich durch die herausgearbeiteten Muster und durch die von ihnen etablierten Begriffe (Interaktionskomponenten, Ereignisse, Konsistenzbedingungsobjekte usw.) ergibt, besteht in der konzisen Kommunikationsmöglichkeit des Entwurfs. Die wohldefinierte Terminologie, wie sie hier angewandt wurde, ermöglicht nahezu reibungslos die Fortführung des Entwurfs in nicht explizit diskutierte Bereiche. Vieles, was nicht angesprochen wurde, läßt sich aufgrund der Terminologie und ihrer strukturierten Anordnung unproblematisch ableiten, so daß der mit ihr vertraute Leser niemals das Gefühl gewinnt, das Entwurfsdokument würde durch Auslassungen schwammig werden.

7.2.3 Zusammenfassung der Diskussion

Die beispielhafte Anwendung der Muster für eine Handhabungsvision und ein Entwurfsdokument hat gezeigt, daß die etablierte Terminologie für ersteres kaum eine Bedeutung, für letzteres aber sehr wohl ein große Bedeutung besitzen kann.

Nun ist dies aber keine empirische Untersuchung, sondern eine exemplarische Betrachtung gewesen. Sie darf deswegen nicht als Bestätigung oder gar Nachweis der erhofften Qualitäten mißverstanden werden. Sie dient vielmehr als Orientierungshilfe für weitere Arbeiten. Ein expliziter Nachweis steht noch aus und wird sicherlich nicht ohne größere Anstrengungen erfolgen können. Eine einzelne Arbeit wäre damit auf jeden Fall überfordert.

7.3 Abschluß

Zentraler Teil dieser Arbeit war die Herausarbeitung eines für die Softwaretechnik konstruktiv verwendbaren Musterbegriffs. Hierbei wurden viele Fragen aufgeworfen und in der Tiefe oder auch nur oberflächlich diskutiert. Im einzelnen:

Der Musterbegriff. Auf Basis einer umfangreichen Literaturrezeption wurde ein allgemeiner Musterbegriff definiert und pragmatisch diskutiert. Dieser Begriff wurde in Interpretations- und Gestaltungsmuster, Entwurfsmuster und Programmiermuster unterschieden. Diese Unterscheidung hat sich im praktischen Einsatz bewährt und als fruchtbar erwiesen.

Zusammenhänge von Mustern. Ebenfalls auf Basis der Literaturrezeption sowie eigener Erfahrung wurde ein Darstellungsschema für die Anordnung und Verweisung von Mustern untereinander entwickelt und auch angewendet. Der hierbei kritische Begriff Mustersprache wurde abgelehnt und nicht weiter thematisiert.

Muster für die Werkzeug und Material Metapher. Diese Arbeit hat im Anschluß an die Definitionen ein größeres Beispiel für die Anwendung der definierten Begriffe gegeben. Es wurden das Erfahrungswissen aus drei Jahren Arbeit an einem Anwendungsrahmen für die Werkzeug

und Material Metapher sowie die grundlegenden Konzepte der Werkzeug und Material Metapher selbst als Muster formuliert. Die Muster sind detailliert herausgearbeitet worden und besitzen eine große Stabilität.

Nachweis der erhofften Qualitäten. Die Muster wurden sowohl zum Zwecke ihrer Erläuterung als auch zu ihrer Erprobung in einem kleinem Beispiel angewendet. Diese Probe bestätigte weitgehend die kritische Haltung gegenüber einem Mustersprachenbegriff. Für einen Nachweis der von Mustern erhofften Qualitäten war das Beispiel nicht ausreichend. Ein empirischer Nachweis steht weiterhin aus.

Zum gegenwärtigen Zeitpunkt (März 1995) erfreuen sich „design patterns“ (Entwurfsmuster) in Journalen und auf Konferenzen großer Beliebtheit. Diese Arbeit hat eine detaillierte Antwort darauf gegeben, was ein Muster ist. Sie hat Muster nach einem vernünftigen Schema in den Musterzusammenhang gestellt, eine größere Menge von Mustern vorgestellt und sie an einem mittelgroßen Beispiel erprobt. Die diskutierte Literatur sowie weitere Veröffentlichungen und Diskussionen unter Wissenschaftlern lassen vermuten, daß die gewählte Struktur tragfähig ist und in ähnlicher Form auch von anderen Wissenschaftlern verwendet werden wird.

Offen aber bleibt, ob die von Mustern erhofften Vorteile in der Softwareentwicklung auf Dauer erreicht werden können. Werden Softwareentwickler effizient ihr Erfahrungswissen mit Hilfe von Mustern kodifizieren und kommunizieren können? Werden Muster die grundlegenden Bausteine für Softwarearchitekturhandbücher sein? Und werden Muster vielleicht doch eines Tages „Sprachqualität“ erreichen werden?

Diese Fragen zu beantworten lag jenseits der Möglichkeiten dieser Arbeit. Muster basieren auf Erfahrungswissen und werden sich auch nur in der Erfahrung bewähren können. Sie sind kein theoretisches und formal ableitbares Konstrukt, sondern bedürfen des empirischen Nachweises. Die relevanten Erfahrungen zu sammeln, empirisch zu bestätigen, auszuwerten und wieder konstruktiv umzusetzen wird die Herausforderung der nächsten Jahre sein.

Literaturverzeichnis

AG92 Robert Allen and David Garlan. “A Formal Approach to Software Architectures.” IFIP ’92, *Conference Proceedings*, Volume 1. Amsterdam, North-Holland: Elsevier Science Publishers, 1992. 134-141.

AIS77 Christopher Alexander, Sara Ishikawa and Murray Silverstein, with Max Jacobson, Ingrid Fiksdahl-King and Shlomo Angel. *A Pattern Language*. New York: Oxford University Press, 1977.

Ale64 Christoper Alexander. *Notes on the Synthesis of Form*. Cambridge, Massachusetts: Harvard University Press, 1964.

Ale79 Christopher Alexander. *The Timeless Way of Building*. New York: Oxford University Press, 1979.

And93 Bruce Anderson. “Workshop Report: Towards an Architecture Handbook.” OOPSLA ’92 Addendum, *OOPS Messenger* 4, 2 (April 1993): 109-114.

And94 Bruce Anderson. “Patterns: Building Blocks for Object-Oriented Architectures.” OOPSLA ’93 Workshop Report, *ACM SIGSOFT Software Engineering Notes* 19, 1 (January 1994): 47-49.

App86 Apple Computer. *MacApp Programmer’s Manual*. Cupertino, California: Apple Computer Inc., 1986.

App89 Apple Computer: *MacApp II Programmer’s Manual*. Cupertino, California: Apple Computer Inc., 1989.

Ban93 François Bancilhon. “Object Database Systems: Functional Architecture.” ISOTAS ’93, LNCS 742, *Object Technology for Advanced Software*. Edited by Shojiro Nishio and Akinori Yonezawa. New York: Springer-Verlag, 1993. 163-175.

BC89 Kent Beck and Ward Cunningham. “A Laboratory For Teaching Object-Oriented Thinking.” OOPSLA ’89, *ACM SIGPLAN Notices* 24, 10 (October 1989).

BCS92 Reinhard Budde, Marie-Luise Christ-Neumann and Karl-Heinz Sylla. „Tools And Materials: An Analysis and Design Metaphor“. TOOLS-7, *Technology of Object-Oriented Languages and Systems, Europe ’92*. Edited by G. Heeg, B. Magnusson and B. Meyer. Prentice-Hall, 1992. 135-146.

Bec94 Kent Beck. “Patterns and Software Development.” *Dr. Dobbs Journal* 02/94 (February 1994): 18-22.

Bis92 Walter R. Bischofberger. “Sniff – A Pragmatic Approach to a C++ Programming Environment.” Usenix ’92, *USENIX 1992 C++ Conference Proceedings*.

BJ94 Kent Beck and Ralph Johnson. “Patterns Generate Architectures.” ECOOP ’94, LNCS 821, *Object Oriented Programming*. Edited by Mario Tokoro and Remo Pareschi. Berlin, Heidelberg: Springer-Verlag, 1994. 139-149.

BKK+92 Reinhard Budde, Karlheinz Kautz, Karin Kuhlenkamp and Heinz Züllighoven. *Prototyping*. Berlin, Heidelberg: Springer-Verlag, 1992.

BKM+84 Reinhard Budde, Karin Kuhlenkamp, Lars Mathiassen and Heinz Züllighoven. *Approaches to Prototyping*. Berlin, Heidelberg: Springer-Verlag, 1984.

BKM+94 Walter R. Bischofberger, Thomas Kofler, Kai-Uwe Mätzel and Bruno Schäffer. “Computer Supported Cooperative Software Engineering: The Beyond-Sniff Approach.” *Computer Science Research at UBILAB*. Edited by Walter R. Bischofberger and Hans-Peter Frei. Konstanz: Universitätsverlag Konstanz, 1994. 32-47.

BKS94 Walter R. Bischofberger, Thomas Kofler and Bruno Schäffer. “Object-Oriented Programming Environments: Requirements and Approaches.” *Software—Concepts and Tools* 15, 2 (June 1994): 49-60.

Boe88 Barry W. Boehm. “A Spiral Model of Software Development and Enhancement.” *IEEE Computer* 21, 5 (May 1988): 61-72.

Boo94 Grady Booch. *Object-Oriented Analysis and Design with Applications*. 2nd Edition. Redwood City, California: Benjamin/Cummings, 1994.

BP92 Walter R. Bischofberger and Gustav Pomberger. *Prototyping-Oriented Software Development*. Berlin, Heidelberg: Springer-Verlag, 1992.

Bra72 Stein Braten. “Model Monopoly and Communication: Systems Theoretical Notes on Democratization.” *Acta Sociologica* 16, 2 (February 1973): 98-107.

Bra88 Stein Braten. “Between Dialogical Mind and Monological Reason. Postulating the Virtual Other.” *Between Rationality and Cognition*. Edited by Miriam Campanella. Turin: Albert Meynier, 1988. Chapter 8.

Bro93 F.A. Brockhaus. *Brockhaus Enzyklopädie*, 19te Auflage. Mannheim: F.A. Brockhaus, 1993.

Brö94 Peter Brödner. „Gestaltung von Arbeit, Technik und Organisation zur Rückgewinnung von Wettbewerbsfähigkeit im Maschinenbau“. *Der GMD-Spiegel* 3/94 (September 1994): 20-22.

BS95 Reinhard Budde und Karl-Heinz Sylla. „Objektorientierte Echtzeitanwendungen auf Grundlage perfekter Synchronisation“. *OBJEKTspektrum* 02/95 (März/April 1995): 54-60.

BW81 F.A.Brockhaus. *Deutsches Wörterbuch*. Herausgegeben von Gerhard Wahrig, Hildegard Krämer und Harald Zimmermann. Wiesbaden: F.A. Brockhaus, 1981.

BZ90 Reinhard Budde und Heinz Züllighoven. *Softwarewerkzeuge in einer Programmierwerkstatt*. München, Wien: R. Oldenbourg Verlag, 1990.

BZ92 Reinhard Budde and Heinz Züllighoven. “Software Tools in a Programming Workshop.” *Software Development and Reality Construction*. Edited by Christiane Floyd, Heinz Züllighoven, Reinhard Budde and Reinhard Keil-Slawik. Berlin, Heidelberg: Springer-Verlag, 1992. 252-268.

CA84 Gael A. Curry and Robert M. Ayers. “Experience with Traits in the Xerox Star Workstation.” *IEEE Transactions on Software Engineering* 10, 5 (September 1984): 519-527.

Cat93 Rick G. Cattell et al. *The Object Database Standard: ODMG '93*. New York: Morgan Kaufmann Publishers, 1993.

CCH+89 Peter S. Canning, William R. Cook, Walter L. Hill and Walter G. Olthoff. "Interfaces for Strongly-Typed Object-Oriented Programming." OOPSLA '89, *ACM SIGPLAN Notices* 24, 10 (October 1989): 457-467.

CHB92 Derek Coleman, F. Hayes and S. Bear. "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design." *IEEE Transactions on Software Engineering* 18, 1 (1992).

Cle94 Andrew Clement. "Computing At Work: Empowering Action By 'Low-level Users'." *Communications of the ACM* 37, 1 (January 1994): 53-63.

CM93 Peter Coad and Mark Mayfield. "Workshop Report: Patterns." OOPSLA '92 Addendum, *OOPS Messenger* 4, 2 (April 1993): 93-95.

Coa92 Peter Coad. "Object-Oriented Patterns." *Communications of the ACM* 35, 9 (September 1992): 152-159.

Cop92 James O. Coplien. *Advanced C++: Programming Styles and Idioms*. Reading, Massachusetts: Addison-Wesley, 1992.

Coy92 Wolfgang Coy (Hrsg.). *Sichtweisen der Informatik*. Braunschweig, Vieweg, 1992.

Coy95 Wolfgang Coy. „Automat – Werkzeug – Medium". *Informatik Spektrum* 18, 1 (Februar 1995): 31-38.

CP94 Armin B. Cremers und Michael Paetau. „Informationstechnische Voraussetzungen für neue Organisationsstrukturen". *Der GMD-Spiegel* 3/94 (September 1994): 12-13.

CS95 James O. Coplien and Douglas C. Schmidt (Editors). *Pattern Languages of Program Design*. Reading, Massachusetts: Addison-Wesley, 1995.

DHM89 Mahesh H. Dodani, Charles E. Hughes and J. Michael Moshell. "Separation of Powers." *Byte* (März 1989): 255-262.

DL95 Yvonne Dittrich and Carola Lilienthal. "Designing a Help Tool for Transparency - a Language Based Approach." CiC '95, *Conference Proceedings*, 1995. Submitted for Publication.

DT92 Mahesh H. Dodani and Chung-Shin Tsai. "ACTS: A Type System for Object-Oriented Programming Based on Abstract and Concrete Classes." ECOOP '92, LNCS 615, *Object-Oriented Programming*. Edited by O. Lehrmann Madsen. Berlin, Heidelberg: Springer-Verlag, 1992. 309-328.

DWA93 Wolfgang Dzida, Marion Wiethoff and Albert G. Arnold. *ERGOguide – The Quality Assurance Guide to Ergonomic Software*. GMD, Schloß Birlinghoven, Germany, 1993.

EK95 Gabriel Eckert and Magnus Kempe. "Modeling with Objects and Values." *Report on Object Analysis and Design* 1, 5 (January 1995): 20-27.

Flo84 Christiane Floyd. "A Systematic Look At Prototyping." *Approaches to Prototyping*. Edited by Reinhard Budde, Karin Kuhlenkamp, Lars Mathiassen and Heinz Züllighoven. Berlin, Heidelberg: Springer-Verlag, 1984. 1-18.

Flo94 Christiane Floyd. „Evolutionäre Systementwicklung und Wandel in Organisationen". *Der GMD-Spiegel* 3/94 (September 1994): 36-40.

Frö95 Frank Fröse. *Entkopplung von Interaktionstypen und Oberflächenbibliotheken am Beispiel StarView*. Studienarbeit. Hamburg: Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, 1995.

Gab93 Richard P. Gabriel. "The Quality Without a Name." *Journal of Object-Oriented Programming* 6, 5 (September 1993): 86-89.

Gab94 Richard P. Gabriel. "The Failure of Pattern Languages." *Journal of Object-Oriented Programming* 5, 9 (February 1994): 84-88.

Gam92 Erich Gamma. *Objektorientierte Software-Entwicklung am Beispiel von ET++*. Berlin, Heidelberg: Springer-Verlag, 1992.

GAO94 David Garlan, Robert Allen and John Ockerbloom. "Exploiting Style in Architectural Design Environments." SIGSOFT '94, *Software Engineering Notes* 19, 5 (December 1994): 175-188.

GHJ+93 Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. "Design Patterns: Abstraction and Reuse of Object-Oriented Design." ECOOP '93, LNCS 707, *Conference Proceedings*. Berlin, Heidelberg: Springer-Verlag, 1993. 406-431.

GHJ+95 Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns: Elements of Reusable Design*. Reading, Massachusetts: Addison-Wesley, 1995.

GK94 Michael R. Genesereth and Steven P. Ketchpel. "Software Agents." *Communications of the ACM* 37, 7 (July 1994): 48-53.

Gol90 Adele Goldberg. "Information Models, Views and Controllers." *Dr. Dobb's Journal* (July 1990): 54ff.

GR83 Adele Goldberg and David Robson. *Smalltalk-80: The Language and Its Implementation.* Reading, Massachusetts: Addison-Wesley, 1983.

GS93 David Garlan and Curtis Scott. "Adding Implicit Invocation to Traditional Programming Languages." ICSE-15, *Proceedings*. Los Alamitos, IEEE Computer Society Press, 1993. 447-455.

GS93 David Garlan and Mary Shaw. "An Introduction to Software Architecture." *Advances in Software Engineering and Knowledge Engineering.* Edited by V. Ambriola and G. Tortora. New Jersey, World Scientific Publishing Company, 1993.

GZ92 Guido Gryczan und Heinz Züllighoven. „Objektorientierte Systementwicklung: Leitbild und Entwicklungsdokumente". *Informatik-Spektrum* 15, 5 (Oktober 1992): 264-272.

HBR+94 Ralph D. Hill, Tom Brinck, Steven L. Rohall, John F. Patterson and Wayne Wilner. "The Rendezvous Architecture and Language for Constructing Multiuser Applications." *ACM Transactions of Computer-Human Interaction* 1, 2 (June 1994): 81-125.

HHG90 Richard Helm, Ian M. Holland and Dipayan Gangopadhyay. "Contracts: Specifying Behavioral Compositions in Object-Oriented Systems." OOPSLA '90, *SIGPLAN Notices* 25, 10 (October 1990): 169-180.

HHM+92 Richard Helm, Tien Huynh, Kim Marriott and John Vlissides. "An Object-Oriented Architecture for Constraint-Based Graphical Editing." *Research Report RC 18524 (79392)*. IBM Research Division, T.J. Watson Research Center, 1992.

Hil92 Ralph D. Hill. "The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications." CHI '92, *SIGCHI Conference Proceedings*.

Edited by Penny Bauersfeld, John Bennet and Gene Lynch. Reading, Massachusetts: Addison-Wesley, 1992. 335-342.

Hil93 Ralph D. Hill. “The Rendezvous Constraint Maintenance System.” UIST ’93, *Proceedings of UIST ’93*. New York: ACM Press. 225-234.

HO93 William Harrison and Harold Ossher. “Subject-Oriented Programming (A Critique of Pure Objects).” OOPSLA ’93 *ACM SIGPLAN Notices* 28, 10 (October 1993): 411-428.

Hol92 Ian M. Holland. “Specifying Reusable Components Using Contracts.” ECOOP ’92, LNCS 616, *Conference Proceedings*. Edited by O. Lehrmann Madsen. Berlin, Heidelberg: Springer-Verlag, 1992. 287-308.

JNZ+93 Jeff A. Johnson, Bonnie A. Nardi, Craig L. Zarmer and James R. Miller. “ACE: Building Interactive Graphical Applications.” *Communications of the ACM* 36, 4 (April 1993): 41-55.

Joh92 Ralph E. Johnson. “Documenting Frameworks using Patterns.” OOPSLA ’92, *ACM SIGPLAN Notices* 27, 10 (October 1992): 63-70.

Kay90 Alan Kay. “User Interface: A Personal View.” *The Art of Human Computer-Interface Design*. Edited by Brenda K. Laurel. Reading, Massachusetts: Addison-Wesley, 1990. 191-207.

Kei89 Reinhard Keil-Slawik. „Systemgestaltung mit Aufgabennetzen“. *Software Ergonomie ’89*. Hrsg. Susanne Maaß und Horst Oberquelle. Stuttgart: Teubner, 1989. 123-133.

KGZ93 Klaus Kilbert, Guido Gryczan und Heinz Züllighoven. *Anwendungsorientierte Softwareentwicklung*. Vieweg, 1993.

KL89 Won Kim and Frederick H. Lochovsky. *Object-Oriented Concepts, Databases and Applications*. Reading, Massachusetts: Addison-Wesley, 1989.

KP88 Glenn E. Krasner and Stephen T. Pope. “A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80.” *Journal of Object-Oriented Programming* 1, 3 (August/September 1988): 26-49.

Lau90 Brenda K. Laurel. *The Art of Human-Computer Interface Design*. Reading, Massachusetts: Addison-Wesley, 1990.

Lea94 Doug Lea. “Christopher Alexander: An Introduction for Object-Oriented Designers.” *ACM SIGSOFT Software Engineering Notes* 19, 1 (January 1994): 39-46.

Lil95 Carola Lilienthal. *Konstruktion und Realisierung eines an der Anwendungssprache orientierten Hilfesystems nach der Werkzeug und Material Metapher*. Diplomarbeit. Hamburg: Universität Hamburg, Fachbereich Informatik, 1995.

Lis88 Barbara Liskov. “Data Abstraction and Hierarchy.” OOPSLA ’87 (Addendum), *ACM SIGPLAN Notices* 23, 5 (Mai 1988): 17-34.

LVC89 Mark A. Linton, John M. Vlissides and Paul R. Calder. “Composing User Interfaces with InterViews.” *IEEE Computer* 22, 2 (February 1989): 8-22.

LW93 Barbara Liskov and Jeannette Wing. “Specifications and Their Use in Defining Subtypes.” OOPSLA ’93, *ACM SIGPLAN Notices* 28, 10 (October 1993): 16-28.

LW93a Barbara Liskov and Jeannette Wing. “A New Definition of the Subtype Relation.” ECOOP ’93, LNCS 707, *Conference Proceedings*. Berlin, Heidelberg: Springer-Verlag, 1993. 118-141.

Maa93 Susanne Maaß. “Software Ergonomie. Benutzer- und Aufgabenorientierte Systemgestaltung.” *Informatik Spektrum* 16, 4 (August 1993): 191-205.

Maa94 Susanne Maaß. *Transparenz – Eine Zentrale Software-Ergonomische Forderung.* Technischer Report, Universität Hamburg, Fachbereich Informatik. 1994.

Mac82 B. J. MacLennan. “Values And Objects In Programming Languages.” *ACM SIGPLAN Notices* 17, 12 (December 1982): 70-79.

Mae94 Pattie Maes. “Agents that Reduce Work and Information Overload.” *Communications of the ACM* 37, 7 (July 1994): 31-41.

Mey88 Bertrand Meyer. *Object-Oriented Software Construction.* Englewood-Cliffs, New Jersey: Prentice-Hall, 1988.

Mey91 Bertrand Meyer. “Design by Contract.” *Advances in Object-Oriented Software Engineering*. Edited by Dino Mandrioli und Bertrand Meyer. New York, London: Prentice-Hall, 1991. 1-50.

Mey92 Bertrand Meyer. *Eiffel. The Language*. New York, London: Prentice-Hall, 1992.

Mey92a Bertrand Meyer. “Applying Design By Contract.” *IEEE Computer* 25, 10 (October 1992): 40-51.

MO92 Susanne Maaß and Horst Oberquelle. “Perspectives and Metaphors for Human-Computer Interaction.” *Software Development and Reality Construction*. Edited by Christiane Floyd, Heinz Züllighoven, Reinhard Budde and Reinhard Keil-Slawik. Berlin, Heidelberg: Springer-Verlag, 1992. 233-251.

MO94 Hanns-Martin Meyer und Karl Obermayr. *Objekte integrieren mit OLE2*. Berlin, Heidelberg: Springer-Verlag, 1994.

MM92 James C. McKim, Jr. and David A. Mondou. “Class Interface Design.” Tools-8, *Technology of Object-Oriented Languages and Systems.* Edited by Raimund Ege, Madhu Singh and Bertrand Meyer. New York, London: Prentice-Hall, 1992. 151-161.

MNL+93 Kin’ichi Mitsui, Hiroaki Nakamura, Theodore C. Law and Shahram Javey. “Design of an Integrated and Extensible C++ Programming Environment.” ISOTAS ’93, LNCS 742, *Object Technology for Advanced Software*. Edited by Shojiro Nishio and Akinori Yonezawa. New York: Springer-Verlag, 1993. 95-109.

NGG+93 David Notkin, David Garlan, William G. Griswold and Kevin Sullivan. “Adding Implicit Invocation to Languages: Three Approaches.” ISOTAS ’93, LNCS 742, *Object Technology for Advanced Software*. Edited by Shojiro Nishio and Akinori Yonezawa. New York: Springer-Verlag, 1993. 489-510.

Nor94 Donald A. Norman. “How Might People Interact with Agents.” *Communications of the ACM* 37, 7 (July 1994): 68-71.

OM95 Gerd Oslowsky-Klein und Eduardo Mendoza. „Client/Server-Entwicklung durch Microsoft Solutions Framework und OLE“. *OBJEKTspektrum* 6 (Januar/Februar 1995): 40-49.

OMG91 Object Management Group. *The Common Object Request Broker: Architecture and Specification*. Revision 1.1 (OMG Document 91.12.1), 1991.

OMG92 Object Management Group. *Object Management and Architecture Guide*. 2nd Edition (OMG Document 92.11.1), 1992.

OH92 Harold Ossher and William Harrison. “Combination of Inheritance Hierarchies.” OOPSLA ’92, *ACM SIGPLAN Notices* 27, 10 (October 1992): 25-40.

Par94a Parcplace Systems. *VisualWorks User’s Guide*. Revision 1.1. Parcplace Systems, Inc., 1994.

Par94b Parcplace Systems. *VisualWorks Cookbook*. Revision 1.0. Parcplace Systems, Inc., 1994.

PC86 David L. Parnas and Paul C. Clements. “A Rational Design Process: How and Why to Fake It.” *IEEE Transactions on Software Engineering*, 12, 2 (February 1986): 251-257.

PML94 Diskussion auf der Patterns-Mailinglist (patterns@cs.uiuc.edu). Subskription via patterns-request@cs.uiuc.edu. Archiv als ftp://st.cs.uiuc.edu:/pub/patterns/ verfügbar. 1994.

Pre94 Wolfgang Pree. “Meta Patterns – A Means for Capturing the Essentials of Reusable Object-Oriented Design.” ECOOP ’94, LNCS 821, *Object Oriented Programming*. Edited by Mario Tokoro and Remo Pareschi. Berlin, Heidelberg: Springer-Verlag, 1994. 150-160.

Pri93 Rubén Prieto-Díaz. “Status Report: Software Reusability.” *IEEE Software* 10, 3 (May 1993): 61-66.

PS94 Jens Palsberg and Michael I. Schwartzbach. *Object-Oriented Type Systems*. New York: John Wiley & Sons Ltd., 1994.

RAB+92 Trygve Reenskaug, Egil P. Andersen, Arne Jorgen Berre, Anne Hurlen, Anton Landmark, Odd Arild Lehne, Else Nordhagen, Eirik Nêss-Ulseth, Gro Oftedal, Anne Lise Skaar and Pâl Stenslet. “OORASS: Seamless Support for the Creation and Maintenance of Object-Oriented Systems.” *Journal of Object-Oriented Programming* 5, 6 (October 1992): 27-41.

RBP+91 James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy and William Lorensen. *Object-Oriented Modeling and Design.* London: Prentice-Hall, 1991.

Rei90 Steven P. Reiss. “Connecting Tools Using Message Passing in the Field Environment.” *IEEE Software* (July 1990): 57-66.

Rie93 Dirk Riehle. *Objektorientierte Architektur nach der Werkzeug – Material Metapher am Beispiel eines grafischen Aufgabennetzeditors.* Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1993.

Rie93a Dirk Riehle. *Dokumentation zur IATMotif-Bibliothek.* Arbeitsbereich Softwaretechnik, Fachbereich Informatik, Universität Hamburg, 1993.

Rie93b Dirk Riehle. *Dokumentation zur FIAK-Bibliothek.* Arbeitsbereich Softwaretechnik, Fachbereich Informatik, Universität Hamburg, 1993.

Rie95 Dirk Riehle. “How and Why to Encapsulate Class Trees.” OOPSLA ’95, *Conference Proceedings*. To appear, 1995.

Riz93 Krista Rizman. “Building More Structured, Understandable, Reusable and Changeable Object-Oriented Software.” *Structured Programming* 14, 3 (1993): 102-109.

Rol93 Arno Rolf. „Informatik und Gestaltung“. *InfoTech* 5, 4 (Dezember 1993). 16-22.

Roy70 W. W. Royce. “Managing the Development of Large Software Systems.” *Proceedings of IEEE WESCON*. IEEE, 1970. 1-9. Reprinted in: *IEEE Tutorial on Software Engineering Project Management*. Edited by R. H. Thayer. IEEE, 1988.

RS95 Dirk Riehle and Karl-Heinz Sylla. “The Traits Pattern Language of Object Coupling.” Unpublished Manuscript.

Rum95 James Rumbaugh. “OMT: The Object Model.” *Journal of Object-Oriented Programming* 7, 8 (January 1995): 21-27.

RZ94 Dirk Riehle und Heinz Züllighoven. „Späte Erzeugung“. *39. Internationales Wissenschaftliches Kolloquium*. Thüringen: Technische Universität Ilmenau (Thüringen), 1994. 3-9.

RZ95 Dirk Riehle and Heinz Züllighoven. “A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor.” *Pattern Languages of Programs*. Edited by James O. Coplien and Douglas C. Schmidt. Reading, Massachusetts: Addison-Wesley, 1995. 9-42.

SDK+95 Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, Daivd M.Young and Gregory Zelesnik. “Abstractions for Software Architecture and Tools to Support Them.” *IEEE Transactions on Software Engineering*. To appear.

SN92 Kevin J. Sullivan and David Notkin. “Reconciling Environment Integration and Software Evolution.” *ACM Transactions on Software Engineering and Methodology* 1, 3 (July 1992): 229-268.

SP93 Bernhard Strassl and Franz Penz. “CommonInteract: An Object-Oriented Architecture for Portable Direct Manipulative User Interfaces.” *Journal of Object-Oriented Programming* 6, 3 (June 1993): 33-39.

Stu95 Thorsten Sturm. *Unbekannter Titel*. Studienarbeit. Hamburg: Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, 1995.

TGP89 David Taenzer, Murthy Ganti and Sunil Podar. “Object-Oriented Software Reuse: The Yoyo Problem.” *Journal of Object-Oriented Programming* 2, 3 (September/ October 1989): 30-35.

Tho89 Ian Thomas. “PCTE Interfaces: Supporting Tools in Software-Engineering Environments.” *IEEE Software* 6, 6 (November 1989): 15-22.

TN92 Ian Thomas and Brian A. Nejmeh. “Definitions of Tool Integration for Environments.” *IEEE Software* 9, 2 (March 1992): 29-35.

Tra88 Will Tracz. “Software Reuse Myths.” *ACM SIGSOFT Software Engineering Notes* 13, 1 (January 1988): 17-21.

Tra92 Will Tracz. “Domain Analysis Working Group Report – First International Workshop on Software Reusability.” *ACM SIGSOFT Software Engineering Notes* 17, 3 (July 1992): 27-33.

Vli90 John M. Vlissides. *Generalized Graphical Object Editing*. Technical Report: CSL-TR-90-427, Stanford University, 1990.

Vli95 John Vlissides. “Reverse Architecture.” Dagstuhl Seminar 9508, *Position Paper*, 1995: 11 pages.

War92 Hans-Jürgen Warnecke. *Die Fraktale Fabrik: Revolution in der Unternehmenskultur*. Berlin, Heidelberg: Springer-Verlag, 1992.

Web47 Max Weber. *The Theory of Social and Economic Organization*. New York: Oxford University Press, 1947.

Web72 Max Weber. *Wirtschaft und Gesellschaft*. 5te Auflage, Tübingen, 1972.

Web86 Merriam Webster, Inc. *Webster's Third New International Dictionary*. Merriam Webster, Inc., 1986.

Wei92 André Weinand. *Objektorientierte Architektur für Grafische Benutzungsoberflächen*. Berlin, Heidelberg: Springer-Verlag, 1992.

Wes91 Uwe Westphal. *The Bauhaus*. London: Gallery Books, 1991.

WG94 André Weinand and Erich Gamma. "ET++ – a Portable, Homogenous Class Library and Application Framework." *Computer Science Research at UBILAB*. Edited by Walter R. Bischofberger and Hans-Peter Frei. Konstanz: Universitätsverlag Konstanz, 1994. 66-92.

WGM89 André Weinand, Erich Gamma and Rudolf Marty. "Design and Implementation of ET++, a Seamless Object-Oriented Application Framework." *Structured Programming* 10, 2 (Juni 1989): 63-87.

WJ90 Rebecca Wirfs-Brock and Ralph E. Johnson. "Surveying Current Research in Object-Oriented Design." *Communications of the ACM* 33, 9 (September 1990): 104-124.

WJ93 Lois Wakemann and Jonathan Joweit. *PCTE: The Standard For Open Repositories*. Englewood Cliffs, New Jersey: Prentice-Hall, 1993.

Woh94 Gerhard Wohland. „Jenseits von Taylor – Irritation als Methode". *Der GMD-Spiegel* 3/94 (September 1994): 22-26.

Woo95 Bobby Woolf. "Making MVC Code More Reusable." *The Smalltalk Report* 4, 4 (January 1995): 15-18.

WW94 Martina Wulf und Dirk Weske. *Konzepte zur Materialversorgung verteilter Werkzeugumgebungen am Beispiel der Anbindung einer objektorientierten Datenbank*. Studienarbeit. Hamburg: Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, 1994.

WWW90 Rebecca Wirfs-Brock, Brian Wilkerson und Lauren Wiener. *Designing Object-Oriented Software*. Englewood Cliffs, New Jersey: Prentice-Hall, 1990.