Domänenanalyse und Analysemuster

Was ist Domänenanalyse?

Das Besondere an Software ist, dass sie für sehr verschiedene Zwecke eingesetzt werden kann: der Typ Maschine: Computer, den die Software steuert, ist nicht von seiner Konstruktion auf eine bestimmte Verwendungsweise festgelegt, er kann jede beliebige Art symbolischer Information verarbeiten. Diese Information kann je nach Anwendung ganz Unterschiedliches sein:

- Daten, die bei rechtswirksamen Geschäften anfallen, wie z.B. Daten einer Bestellung oder Rechnung,
- Informationen, die ein Sensor liefert, z.B. in einem Airbag,
- Informationen, die ein Gerät steuern, wie etwa die Einstellung der Strahlendosis eines medizinischen Bestrahlungsgerätes,
- oder auch Daten, die nur wegen dem Computer selbst existieren, wie etwa der Programmcode für einen Compiler.

Die Beispiele sollen zeigen, dass der Softwareentwickler nur dann eine vernünftige Software entwickeln kann, wenn er über das jeweilige Fachgebiet (vornehm denglisch „Domäne“) jedenfalls soweit Bescheid weiß, dass er Informationen von Fachleuten aufgreifen und in die Steuerung von Computern zweckmäßig und korrekt umsetzen kann. Der Softwareentwickler muss also nicht nur ein Fachmann *seines* Metiers, der Softwareentwicklung, sein, sondern auch ein Stück Experte auf dem Fachgebiet, für das er Software schreibt.

Er muss also in unseren Beispielen verstehen

- wie ein Unternehmen die Auftragsabwicklung gestaltet, welche Schritte durchzuführen sind, welche Dokumente erstellt werden müssen, wohin sie verschickt werden müssen, wie lange sie aufbewahrt werden müssen usw. usw.,
- wie ein Sensor in einem Airbag arbeitet, welche Daten er in welcher Einheit wie oft liefert, wie diese Daten in Bezug darauf zu interpretieren sind, ob sie einen Aufprall signalisieren...
- wie das Bestrahlungsgerät anzusteuern ist, wie die Eingabe eines Therapeuten überprüft werden muss, ob es rechtliche Vorschriften zum Protokoll der Einstellungen gibt etc.,

- in welche Zielsprache der Code übersetzt werden muss, welche Befehle diese Zielsprache kennt und wie die Syntax und Semantik der zu übersetzenden Programmiersprache ist...

Dieses Wissen ist *Voraussetzung* für die Softwareentwicklung. Es ist in vielen Fällen entscheidend für den Erfolg der Entwicklung, in jedem Fall aber für die Qualität des Ergebnisses.

Das Verstehen des Anwendungsgebiets bezeichne ich als „Domänenanalyse“ und sie wird uns in dieser Vorlesung beschäftigen.

Stellung im Softwareentwicklungsprozess

Die verschiedenen Vorgehensmodelle des Softwareentwicklungsprozesses müssen natürlich alle in irgendeiner Weise dafür sorgen, dass die Gegebenheiten des Fachgebiets in der Entwicklung berücksichtigt werden.

In der Vorgehensweise des (Rational) Unified Process hat die Analyse des Fachgebiets eine eigenen Arbeitsschritt, der in das sogenannte „Domänenmodell“ mündet:

> A domain model captures the most important types of objects in the context of the system. The domain objects represent the “things” that exist or events that transpire in the environment in which the system works.
>
> – The Unified Software Development Process S. 119

Wenn es um Geschäftsanwendungen geht, wird auch oft das Domänenmodell als ein Teil der Beschreibung der Geschäftsprozesse gesehen. Man unterscheidet, ob man das Anwendungsgebiet eher unter dem Gesichtspunkt der Abläufe, der Geschäftsprozesse (man sagt: *prozessorientiert*) oder eher unter dem Gesichtspunkt der dabei anfallenden Informationen (man sagt: *datenorientiert*) untersuchen und beschreiben möchte. Der Unified Process kennt beide Vorgehensweisen.

In anderen Vorgehensmodellen wird die Kenntnis des Fachgebiets nicht in einem eigenen Schritt der Entwicklung berücksichtigt, sondern sie wird begleitend mit der Entwicklung der Software erlangt. So sieht etwa das Extreme Programming vor, dass die einzelnen Iterationsschritte durch „User Stories“ getrieben werden und der Kunde und die Anwender ständig an der Entwicklung beteiligt sind und so ihre Kenntnis des Fachgebiets einbringen.

> The customer is the other half of the essential duality of extreme programming. The programmer knows how to program. The customer knows what to programm. Well, not at

> first, of course, but the customer is willing to learn just as much as the programmer is.
> ...
> The best customers are those who will actually use the system being developed, but who also have a certain perspective on the problem to be solved.
>
> – Kent Beck XP explained S. 142f

Ich halte die Untersuchung, die Beschreibung und das Verstehen des Anwendungsgebiets für die Softwareentwicklung für so wesentlich, dass ich dem Thema eine Vorlesung widme.

Ziel

Ziel der Domänenanalyse ist das Verstehen und Beschreiben des Anwendungsgebiets. Das bedeutet

- Maßstab sind die Gegebenheiten des Anwendungsgebiets: man kann demzufolge ein Domänenmodell überprüfen, indem man fragt, ob sich denn die Sache auch tatsächlich so verhält.
- Man kann nicht jedes Detail beschreiben, sondern muss das Wesentliche erfassen, das das Anwendungsgebiet ausmacht in Bezug auf eine zu erstellende Software. Die Beschreibung macht also durchaus eine Abstraktion aus.
- Man muss die Beschreibung so formulieren, dass man sie möglichst leicht später als Basis für die Entwicklung verwenden kann: sie sollte gut strukturiert, präzise und vollständig sein. Dazu ist oft ein gewisser Formalismus nötig, der die Beschreibung des Anwendungsgebiets in eine Sprache bringt, die man für die Softwareentwicklung dann einsetzen kann.
- Welcher Formalismus dafür eingesetzt werden kann, hängt in hohem Maße vom jeweiligen Anwendungsgebiet ab. Manche Anwendungsgebiete haben einen etablierten Formalismus für ihre Beschreibung (z.B. im technischen Umfeld), während man für andere einen Formalismus aus der Informatik einsetzen kann (z.B. Zustandsautomaten, Entity-Relationship-Diagramme, UML) und noch andere Anwendungsgebiete sind in einem so hohen Maße informell, dass man spezielle Notationen einsetzen muss oder eben die Beschreibung in einer natürlichen Sprache verfasst.

Ergebnis

Im Ergebnis sollte eine Beschreibung des Anwendungsgebiets vorliegen, die

- von den Experten der „Domäne“ überprüfbar ist,
- von den Entwicklern gut verstanden wird,
- die für die Entwicklung wesentlichen Informationen enthält,
- zur Grundlage der Validierung der zu konstruierenden Software gemacht werden kann

Wie diese Beschreibung im Einzelnen aussieht, hängt von dem jeweiligen Anwendungsgebiet ab. Im Falle des Airbags gehört etwa die Beschreibung des Verhaltens der beteiligten Sensoren, Explosionsmechanismen usw. auch dazu.

In vielen Fällen, wo Software zur Simulation einer Dienstleistung entwickelt wird, entsteht bei der Domänenanalyse

- ein „Domänenmodell“, besser: Fachmodell – das die „Dinge“, Entitäten, Objekte im Anwendungsgebiet und ihre Struktur und Beziehungen umfasst,
- ein Begriffsverzeichnis, auch Glossar genannt, das diese „Dinge“ definiert und genauer beschreibt.

Dies ist insbesondere dann der Fall, wenn datenorientiert vorgegangen wird.

Wenn wir zum Beispiel die Domäne einer Bibliothek analysieren, dann betrachten wir Bücher, Leser, Regelungen der Ausleihe usw. usw. – eben die tatsächlichen Gegebenheiten in einer Bibliothek. Und all dieses wollen wir systematisch aufschreiben. Ein Mittel der Systematisierung ist ein Klassendiagramm der UML, aber es ist nur *ein* Mittel. Wo immer wir Inhalte ausdrücken wollen, die wir in diesem Klassendiagramm nicht unterbringen, schreiben wir sie einfach als Text auf. (Vielleicht können wir auch die OCL *Object Constraint Language* einsetzen, um mit logischen Aussagen präzise zu formulieren, wie sich Sachverhalte verhalten.[1])

Aber nochmals: Es geht dabei bei diesem Vorgehen *nicht* darum, eine Datenbankstruktur oder Klassenstruktur für den Code eines Programms zu entwerfen, sondern die wirklichen Sachverhalte und Gegebenheiten in der Bibliothek zu verstehen und zu beschreiben.

[1] Aber nicht in dieser Vorlesung, weil die Zeit nicht reicht, die OCL einzuführen.

Wenn diese Beschreibung die Sachverhalte korrekt abbildet, wird sie sich in einem späteren Schritt (dem Entwurf) dafür eignen, daraus die benötigten Strukturen der zu konstruierenden Software abzuleiten.

Weshalb der Nachdruck auf diesen Unterschied? Softwareentwickler neigen dazu gleich in den technischen Kategorien ihrer Systeme zu denken und die wirkliche Welt darunter zu subsumieren. Nicht selten entgehen ihnen deshalb Aspekte oder Sachverhalte – und fallen ihnen erst viel später in der Entwicklung auf, möglicherweise erst dann, wenn sie schon mit Mühe eine Software entwickelt haben, von der sich dann herausstellt, dass sie gar nicht die *tatsächlichen* Fragestellungen des Anwendungsgebiets bewältigt.

Produkte und Bestandteile der Domänenanalyse

In der sehr vereinfachten Form, in der wir die Domänenanalyse in der Vorlesung behandeln wollen, beschränken wir uns auf Anwendungen, für die ein Fachmodell in UML zusammen mit einem Begriffsverzeichnis ausreichend ist.

Solche Anwendungsgebiete sind oft die Simulation von Dienstleistungen, bei denen die Struktur der Information das Fachgebiet wesentlich ausmacht – wie etwa das Beispiel der Bibliothek. Solche Gebiete eignen sich auch für die Vorlesung, weil sie keine tiefen Fachkenntnisse erfordern. Aber Achtung: dies trifft nur vordergründig zu. Sobald man tiefer in das Gebiet des Bibiothekswesen etwa einsteigt, stellt man fest, dass sich hinter einer anscheinend simplen Fassade interessante Fragestellungen verbergen (z.B. Ist ein Buch in der zweiten Auflage noch derselbe Titel wie in der ersten Auflage? Wann ändert sich bei einer Neuauflage die ISBN?).

Kurz: in der Vorlesung wird über die Maßen vereinfacht!

Fachmodell

Das Fachmodell ist oft ein Klassendiagramm in UML, das die Objekte (genauer Objekttypen, Klassen) das Fachgebiets, ihre Eigenschaften (Attribute) und ihre Beziehungen darstellt.

An Beziehungen verwendet man

- die Generalisierung/Spezialisierung, die als Substitution im konzeptionellen Sinne verstanden wird (etwa: ein Ausleiher in der Bibliothek kann ein Student oder ein Dozent sein),
- insbesondere die Assoziation mitsamt Multiplizitäten,

- die Aggregation als die Form der Assoziation, die die Beziehung „enthält“ beschreibt.

Methoden (in der UML2: Operationen) wird man im Klassendiagramm für das Fachmodell meist nicht angeben, viele ergeben sich von alleine, von anderen ist noch nicht die Rede. Eine allgemeine Beschreibung der Verantwortlichkeit einer Klasse im Anwendungsgebiet kann hingegen durchaus sinnvoll sein: man schreibt diese Angaben in ein extra Abteil des Klassendiagramms – oder stellt sie im Begriffsverzeichnis dar, siehe unten.

Man kann das Klassendiagramm zum Verständnis mit Objektdiagrammen ergänzen, die einen Schnappschuss der strukturellen Situation verdeutlichen (oft helfen Objektdiagramme auch zum Finden des Klassendiagramms).

Oft gibt es Eigenschaften, die nicht graphisch im Klassendiagramm der UML dargestellt werden können. In diesem Fall kann man Notizen im Diagramm anbringen und dort die Sachverhalte beschreiben. Noch besser ist es, das Ergebnis der Domänenanalyse in einem Text darzustellen, in dem dann auch die entwickelten Klassendiagramme erläutert werden.

Begriffsverzeichnis

Das Begriffsverzeichnis ist ein Glossar, das die Fachbegriffe erläutert. Dabei wird natürlich das Vokabular des Fachgebiets verwendet. Oft werden Synonyme mit angegeben.

Es dient zwei Zwecken:

- Definition und Erläuterung von Begriffen.
- Vereinheitlichung der Sprechweise, so dass dieselbe Sache auch stets mit demselben, möglichst eindeutigen Begriff bezeichnet wird.

Da viele Anwender und Kunden die UML nicht kennen oder nicht jedes graphische Konstrukt der UML korrekt interpretieren, ist das Begriffsverzeichnis für die Diskussion der Ergebnisse der Domänenanalyse sehr wichtig.

Beispiel

Analysiert man die Gegebenheiten in einer Bibliothek, stößt man auf interessante Punkte:

Man spricht von einem Buch und kann dabei verschiedene Dinge meinen. Man denkt vielleicht an ein konkretes Exemplar eines Buchs, das

gestern von Petra Obermeyer ausgeliehen wurde und mittlerweile auf Seite 12 einen Kaffeefleck hat. Oder man denkt an das Buch über „Objektorientierte Analyse und Design“ von Grady Booch und meint damit nicht das Exemplar, das Petra Obermeyer hat, sondern den Inhalt – also das was durch die ISBN gekennzeichnet wird, oder was im Katalog des Buchhändlers steht.

Dieses Beispiel zeigt, dass die Frage der *Identität* und der *Gleichheit* eine wichtige Rolle in der Domänenanalyse spielt.

Man muss also Bücher im Sinne von Exemplaren und von Titel unterscheiden. Ausgeliehen werden konkrete Exemplare von Büchern (Buchexemplar), während bei der Vormerkung der Inhalt, der Titel des Buchs (Buchtitel) gemeint ist. Denn wer sich ein Buch zur Ausliehe vormerken lässt, will nicht ein ganz bestimmtes Exemplar, sondern das erste wieder verfügbare Exemplar.

Die beiden Begriffe Buchtitel und Buchexemplar stehen in der Beziehung der Beschreibung: der Buchtitel beschreibt den Inhalt des Buches und damit etwas, was allen Exemplaren desselben Buchtitels gemeinsam ist.

Beim Ausliehen oder Vormerken von Büchern werden Informationen festgehalten, die weder zu den Eigenschaften des Buchs noch zu den Merkmalen des Ausleihenden gehören, z.B. das Ausleihdatum. Deshalb treten neue „Dinge“ in der Beschreibung des Fachgebiets auf, die die Vorgänge beschreiben z.B. die Ausleihe – und die all die Attribute enthalten die durch die Beziehung des Ausliehenden zu dem entliehenen Buchexemplar entstehen.

Das Beispiel zeigt also auch, dass die Zuordnung von Information zu den Objekten bzw. Klassen wichtig ist, um die *Struktur* des Fachgebiets zu erfassen. Je besser die Struktur erfasst wird, umso weniger zusätzliche Erläuterungen sind notwenig, um den Kern der Sache zu erfassen.

Aus der Situation im Beispiel

- Das Buch „Objekt-orientierte Analyse und Design“ von Grady Booch mit der Signatur „605-dav-02“ hat Hans Klein ausgeliehen.
- Ein weiteres Exemplar des Buches ist von Petra Obermeyer ausgeliehen, sie muss das Buch bis zum 30. Mai zurückgeben.
- Peter Lustig ist auch an diesem Buch interessiert und hat es am 24.5. vormerken lassen.

kann man folgende Begriffe herausdestillieren:

Buchtitel (BuchTitel) Der Buchtitel ist die Bezeichnung und Beschreibung des Inhalts eines Buches, also Autor(en), Titel, Verlag, Erscheinungsjahr usw. In der Bibliothek kann es zu einem Buchtitel mehrere Exemplare geben. Jeder Buchtitel hat eine eindeutige ISBN.

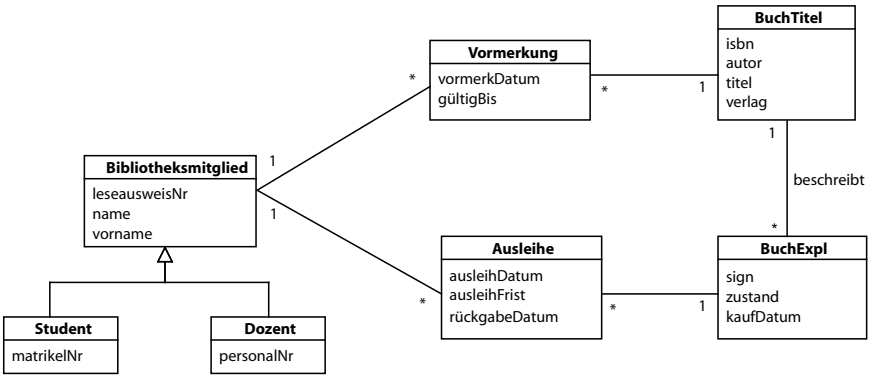
ISBN (isbn) Die ISBN (Internationale Standardbuchnummer) kennzeichnet jedes Buch eindeutig. Wie die ISBN aufgebaut wird und nach welchen Regeln sie vergeben wird, wird auf der Webseite der ISBN Agentur http://www.german-isbn.org erläutert.

Buchexemplar (BuchExpl) Das Buchexemplar ist das gegenständliche Exemplar eines Buchtitels. Der Buchtitel beschreibt den Inhalt des Buchexemplars. Das Buchexemplar hat einen Zustand, ein Kaufdatum usw. In der Bibliothek wird jedes Buchexemplar durch die Signatur identifiziert.

Signatur (sign) Die Signatur eines Buchexemplar setzt sich aus der Kennzeichnung des Standorts und der laufenden Nummer eines Buchexemplars zusammen. Die Signatur wird in der untersuchten Bibliothek nach folgendem Schema aufgebaut...

usw. usf. Auf diese Weise wird ein Glossar aufgebaut und mit den Fachexperten abgesprochen.

Den Zusammenhang dieser Begriffe können wir in einem Klassendiagramm, der Visualisierung des Fachmodells darstellen. An unserem Beispiel haben wir folgendes Klassendiagramm entwickelt (ergänzt um die Darstellung des Sachverhalts, dass in unserer Bibliothek Studenten unbd Dozenten ausleihen können – und dies einen Unterschied machen kann, etwa was die Ausleihfrist angeht):



Als Übung empfohlen: das Beispiel zu einem kleinen Text ausarbeiten, der das Diagramm erläutert und ein vollständiges Begriffsverzeichnis

enthält. Ferner kann man überlegen, worüber das Diagramm nichts aussagt, obwohl es für die Bibliothek von Bedeutung ist.

Vorgehen

Wie geht man vor, um ein Fachmodell und ein Begriffsverzeichnis zu entwickeln?

Informationsquellen

Für die Domänenanalyse verwendet man Informationen aus verschiedenen Quellen, etwa

- Beschreibungen des Systems durch den Kunden (z.B. Angaben über die Eigenschaften und Leistungen des zu konstruierenden Systems).
- Beschreibungen eines bereits vorhandenen Systems und das vorhandene System selbst. Oft werden Softwaresysteme als Weiter- oder Neuentwicklung vorhandener Systeme konstruiert. In diesem Fall ist die Untersuchung des vorhandenen Systems eine wertvolle Informationsquelle.
- In manchen Fällen dient das zu konstruierende System zur Unterstützung von Abläufen, die bisher nicht mit Rechnerunterstützung abgewickelt werden. Man analysiert dann den Ist-Zustand, das aktuelle Vorgehen.
- Fachliche Gegebenheiten sind je nach Fachgebiet oft allgemein zugänglich: Rechtliche Grundlagen, Vorschriften, Beschreibungen des Gebiets aus Wissenschaft und Praxis, Beschreibungen eingesetzter Geräte usw. usf.
- Fachliche relevante Daten findet man oft in Dokumenten und Formularen, die im Fachgebiet eingesetzt werden, z.B. Rechnungen, Kataloge, Produktbeschreibungen...
- Interviews und Befragungen der Auftraggeber, der späteren Anwender des zu konstruierenden Systems und von sonstigen Fachexperten.

Wichtig ist, dass man diese Informationsquellen sachlich auswertet und nicht bereits in eine technische Vision des zu konstruierenden Systems einordnet. Sicherlich ist es nützlich, schon früh eine Idee der Architektur der Lösung zu haben, aber in der Domänenanalyse versuchen wir

das *Problem* zu verstehen – und der Hintergedanke an eine bestimmte Lösung kann manchmal den Blick für die eigentliche Fragestellung verstellen.

Auswertung der Quellen

Wie wertet man diese Quellen aus?

Dies hängt natürlich auch wieder von dem jeweiligen Fachgebiet ab. Geht es etwa um die Steuerung einer Ampelanlage, dann wird man schon verstehen müssen, wie die Ampeln angesteuert werden, wie man erkennt, ob sie einen Fehler haben (Lampe defekt o.ä.) usw. Oft gibt es bei Systemen im technischen Umfeld bereits vorgegebene Beschreibungen und Sprachen, die in diesem Gebiet verwendet werden.

In anderen Fällen, etwa bei unserem Beispiel der Bibliothek, gibt es zwar eine Fachsprache der Bibliothekare, die wir aber nicht einfach so einsetzen können. Wir werden also die Informationen mit der Notation des Klassendiagramms strukturieren.

Welche Vorgehensweisen gibt es?

Häufig verwendet wird die *Textanalyse* nach Russel Abbot: Man untersucht die Ausgangstexte in folgenden Schritten

- Substantive markieren: sie sind Kandidaten für Objekte resp. Klassen und Attribute
- Analyse der Liste der gefundenen Begriffe
  - Synonyme finden und einander zuordnen
  - Irrelevante Begriffe streichen
  - Vage Begriffe präzisieren oder streichen
  - Begriffe zu Klassen, Attributen und Beziehungen zuordnen
- Verben markieren: sie sind oft Kandidaten für Beziehungen zwischen Objekten, für Aktionen oder für die Verantwortlichkeit von Objekten
- Analyse der Liste der gefundenen Verben
  - Irrelevante und vage Begriffe präzisieren oder streichen
  - Zuordnen, welche der gefundenen Klassen betroffen sind
  - Assoziationen oder auch Generalisierungen oder Spezialisierungen identifizieren
  - ggfs. den Klassen Methoden oder die Beschreibung ihrer Verantwortlichkeit hinzufügen

Eine andere Vorgehensweise besteht darin, mit einer Liste von Kategorien zu beginnen und die im Fachgebiet gefundenen Begriffe in diese Kategorien einzuteilen. Damit kann man dann das Klassendiagramm konstruieren und strukturieren. Eine solche Kategorienliste könnte z.B. sein:

Kategorie	**Beispiele**
Dinge	Bleistift, Auto, Buch
Beteiligte	Unternehmen, Person, Student
Ort	Stadt, Gebäude, Regal
Rolle	Arzt, Patient, Ausleiher, Leser
Ereignis	Verkauf, Ausleihe, Vormerkung
Vorgang	Leistung, Prozess, Bezahlung
Beschreibung	Typenschild, Katalogeintrag
Beziehung	Heirat, Besitz
Abstrakte Dinge	Zeit, Quantität, Symbol

Auch sinnvoll ist es, ähnliche Probleme zu vergleichen. Es gibt sogenannte Analysemuster, die zeigen, wie man beispielhaft bestimmte Sachverhalte strukturieren kann. Für die eigene Domänenanalyse kann man solche Muster verwenden und auf die eigene Situation übertragen. Analysem uster findet man in dem Buch „Lehrbuch der Objektmodellierung“ von Heide Balzert und in dem Buch „Analysemuster – Wiederverwendbare Objektmodelle“ von Martin Fowler (das jedoch teilweise sehr speziell ist).

Überprüfung

Die Ergebnisse der Domänenanalyse sollten mit den Fachexperten diskutiert und abgestimmt werden. Der Maßstab der Diskussion muss dabei sein: verhält es sich im Fachgebiet wirklich so, wie es im Fachmodell beschreiben ist?

Fragen der Überprüfung sind also

- Treffen beschriebene Sachverhalte zu?
- Sind die wesentlichen Gegebenheiten beschrieben?
- Gibt es unklare Sachverhalte? (Es gibt Fachgebiete mit einem hohem Maß an informellen Sachverhalten – es ist auch wichtig zu wissen, welche dies sind!)
- Sind die gefundenen Begriffe (Klassen und Attribute) klar beschrieben und werden sie von den Softwareentwicklern im fachlich korrekten Sinne verstanden?

Analysemuster

In der Analyse und Beschreibung von Anwendungsgebieten begegnet man immer wieder ähnlichen Konstellationen. Natürlich gibt es große Unterschiede je nach Typus des Anwendungsgebiets: so treten im technischen Umfeld, in dem eingebettete Software entwickelt wird ganz andere Fragestellungen auf als in eher datenzentrierten Anwendungsgebieten.

Solche typischen Fragestellungen, Konstellationen und Lösungskonzepte nennt man „Muster“, speziell hier „Analysemuster“. Wir wollen solche Analysemuster betrachten für datenzentrierte Anwendungsgebiete, wie man sie z.B. bei Geschäftsanwendungen häufig vor sicht hat.

Analysemuster sind im Unterschied zu Entwurfsmusters nicht einheitlich strukturiert: Verschiedene Autoren haben Analysemuster in sehr unterschiedlicher Weise aufgeschrieben; auch der Abstraktionsgrad ist recht unterschiedlich.

Ich habe versucht, die Muster auf einem etwa gleichen Abstraktionslevel zu formulieren, nämlich als allgemeine, einfache Analysemuster, die grundlegende Strukturen von Entitäten darstellen. (Und: möglichst simpel)

Quellen sind:

Für elementare Muster:

- **David C. Hay** Data Model Patterns: Conventions of Thought, Dorset House Publishing 1996
- **Martin Fowler** Analysis Patterns: Reusable Object Models, Addison Wesley 1997
- **Heide Balzert** Lehrbuch der Objektmodellierung, Spektrum Akademischer Verlag 1999

Für komplexe Muster:

- **Peter Coad, Eric Lefebvre, Jeff de Luca** Java Modeling on Color with UML: Enterprise Components and Process, Prentice-Hall 1999

Übersicht über die Analysemuster

- Fundamentale Muster
  - 1 Beschreibung - Exemplar

  - 2 Substitut
- Aggregationsmuster
  - 3 Liste
  - 4 Baugruppe
  - 5 Gruppe
  - 6 Hierarchie
- Beziehungsmuster
  - 7 Rollen
  - 8 Koordinator
  - 9 Wechselnde Rollen
  - 10 Rollenbeziehung
- Historie
  - 11 Historie
  - 12 Gruppenhistorie
- Komplexe Muster
  - Archetypen von Klassen: Vorgang — Beteiligte, Ort, Sache — Rolle — Beschreibung
  - Die allgemeine Muster-Komponente
  - und beispielhafte Anwendungen

Elementare Analysemuster

Muster 1: Beschreibung — Exemplar

Häufig benötigt man Informationen über Typen von Dingen oder anderen Entitäten in einem Anwendungsgebiet. So unterscheidet man in einer Bibliothek etwa ein konkretes Buch von einem Buchinhalt. Das konkrete Buch wird etwa durch seine Signatur beschrieben, während der Inhalt des Buchs durch die ISBN identifiziert wird. Für die Bibliothek ist es wichtig zu wiessen, wer welches konkrete Buch ausgeliehen hat – wer hingegen ein Buch sucht, den interessiert der Inhalt und es ist ihm egal, welches konkrete Exemplar der in der Bibliothek vom gleichen Buch vorhandenen Exemplar er bekommt.

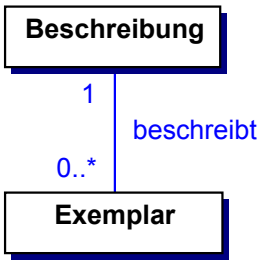
Diese Situation führt zum Analysemuster Beschreibung — Exemplar:

Namen
Beschreibung — Exemplar, auch: Exemplartyp (Balzert);

Zweck

- Unterscheide zwischen einem konkreten Exemplar und seiner Beschreibung.
- Verschiedene Exemplare haben eine gemeinsame Beschreibung.

Struktur



Eigenschaften

- Die Beschreibung enthält Typinformationen über Exemplare
- Exemplare enthalten die Information, die sie voneinander unterscheidet
- Eine Beschreibung kann manchmal ohne Exemplare existieren
- Ein Exemplar hat stets genau eine Beschreibung.

Beispiele

- Buchtitel — Buchexemplar
- Autotyp, -serie — Auto mit Fahrgestellnummer
- Position auf einer Rechnung — Beschreibung des Artikels
- GOZ Gebührenordnung für Ärzte — konkrete Leistung des Arztes
- tritt oft auf bei 2. Normalform im Datenbankdesign

Muster 2: Substitut

Oft hat man die Situation, dass eine bestimmte Klasse von Objekten an die Stelle einer anderen treten kann – oder dass bestimmte Klassen gleichartig verwendet werden können.

Ein typisches Beispiel ist das einer natürlichen Person oder einer juristischen Person: in vielen Situationen, z.B. bei Bestellungen, Rechnungen u.ä. kann sowohl eine Privatperson (natürliche Person) oder eine Firma (juristische Person) beteiligt sein.

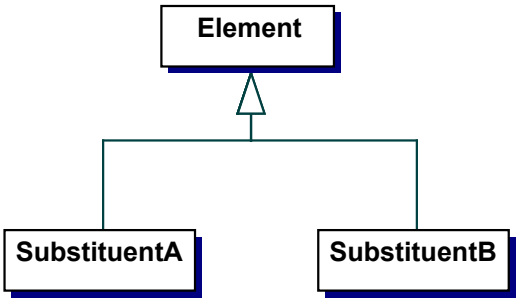
Im angelsächsichen Sprachraum spricht man oft von „Party“, wenn Beteiligte gemeint sind, die von unterschiedlicher Art sein können, die jedoch im Kontext gegeneinander substituierbar sind.

Namen
Substitut, als wichtiges Beispiel Party (Fowler)

Zweck

- Eine Oberklasse erlaubt die Verwendung von Substituenten, wo immer ein Element erwartet wird.
- Dadurch wird die gleichartige Behandlung substituierbarer Objekte ermöglicht.

Struktur



Eigenschaften

- Die Oberklasse, das Element enthält gemeinsame Eigenschaften. (Sie kann auch abstrakt sein).
- Besonderheiten der jeweiligen Substituenten werden in den Subklassen modelliert.
- Regelmäßig gibt es die gleichartige Verwendung der Substituenten.

Beispiele

- Beteiligte (Party) — Person und Institution
- Konto — Privatkonto und Geschäftskonto
- Rolle — verschiedene Rollen
- Fall — Normalfall und Spezialfall (Fowler)
- Restriktion: Person — Mann, Frau
- Erweiterung: Person — Angestellter, Kunde

Verwendung

- unterschiedliche Struktur – gleiche Verwendung (z.B. Party)
- unterschiedliches Verhalten – analoge Verwendung (z.B. Geometrische Figuren)

Muster 3: Liste

Geradezu allgegenwärtig ist das Analysemuster der Liste: gleichartige Positionen werden in tabellarischer Form aufgelistet.

Informationen in Rechnungen, Bestellung, Lieferscheinen z.B. sind in aller Regel als Liste organisiert. Eine Rechnung etwa besteht aus einem Rechnungskopf und den einzelnen Rechnungspositionen.

Oft besteht zwischen der Liste und ihren Elementen die Beziehung der Delegation von Aufgaben: so wird etwa die Gesamtsumme einer Rechnung als Summe der Einzelpositionen berechnet; jedes Listenelement seinerseits berechnet die Teilsumme der Position als Multiplikation der Anzahl mit dem Einzelpreis.

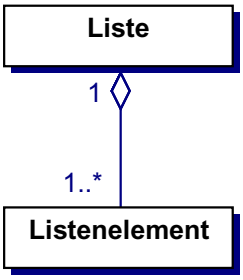
Namen
Liste (Balzert), Item - LineItem, Collection - Worker (Coad)

Zweck

- Strukturierung eines Objekt, das aus gleichartigen Teilen zusammengesetzt ist als Liste.
- Gemeinsame Information für die Gesamtliste, gleichartige Informationen für die Listenpositionen

- Listenpositionen sind nur im Kontext der Liste sinnvoll
- Delegation von Aufgaben des Gesamtobjekts als Teilaufgaben an die Listenelemente.

Struktur



Eigenschaften

- „Besteht aus“-Beziehung (Aggregation, auch Komposition)
- Die Teile sind gleichartig, sie benötigen den Kontext der Liste – oft „schwache Entitätstypen“
- Eigenschaften des Aggregatobjekts, der Liste treffen auf alle Elemente zu.

Beispiele

- Rechnungen, Bestellungen, Angebote, Lieferscheine; jeweils Kopfinformationen – Positionen
- Lager — Lagerplatz
- Transaktion — Transaktionsposition
- Fahrplan einer Buslinie – Haltestellen mit Zeiten
- ...

Muster 4: Baugruppe

Von einer Baugruppe spricht man, wenn es sich um Dinge handelt, die fix aus anderen Dingen, ihren „Bauteilen“ zusammengesetzt sind.

Um mal ein eher technisches Beispiel zu nennen: Ein Dialog einer graphischen Benutzeroberfläche besteht oft aus einer fix definierten Menge

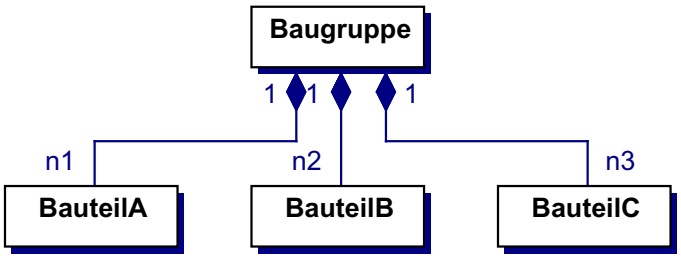
von verschiedenen Elementen wie Textfelder, Buttons usw. Diese Elemente sind in dieser Konstellation fest mit dem Dialog verbunden und werden mit dem Dialog erzeugt und auch mit ihm wieder gelöscht. Es handelt sich um eine Baugruppe.

Namen
Baugruppe (Balzert)

Zweck

- Ein komplexes Ganzes besteht aus Teilen
- Die Teile bilden eine fixe Gruppe und werden immer als Ganzes verwendet

Struktur



Eigenschaften

- Das Ganze besteht aus Teilen verschiedener Art.
- Die Komposition ist statisch, die beteiligten Objekte treten fix in dieser Zusammensetzung auf und sind starr verbunden.
- Die Teile sind allein nicht sehr sinnvoll, sondern werden durch die Zusammenstellung als Baugruppe im Ensemble nützlich.

Beispiele

- Bestandteile graphischer Benutzeroberflächen
- Technische Geräte, eine Kamera z.B. besteht aus Gehäuse, Objektiv, Blende, Verschluss usw.

Muster 5: Gruppe

Wenn eigenständige Objekte zeitweise oder zu einem bestimmten Zweck zusammengefasst werden, spricht man von einer Gruppe.

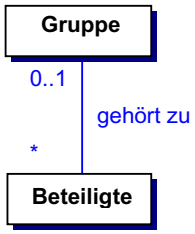
So können etwa die Angestellten einer bestimmten Firma eine Gruppe bilden, die die Aufgabe hat ein bestimmtes Projekt durchzuführen. Im Unterschied zu den Positionen einer Liste sind die Mitglieder der Gruppe für sich „First-Class-Citizens“, d.h. sie können auch außerhalb und unabhängig von der Gruppe existieren.

Namen
Gruppe (Balzert)

Zweck

- Selbständige Einzelobjekte gehören (zu einem bestimmten Zeitpunkt) zu einer Gruppe.

Struktur



Eigenschaften

- Eine Gruppe hat zu einem Zeitpunkt in der Regel mehrere zugeordnete Mitglieder.
- Ein Mitglied ist nicht *nur* als solches definiert, sondern hat eine eigenständige Existenz auch auáerhalb der Gruppe.
- Die Zugehörigkeit kann sich aber ändern, d.h. Objektbeziehungen können auf- und abgebaut werden, während die beteiligten Objekte bestehen bleiben.

Beispiele

- Verein — Mitglied
- Verband — Person (etwa ein Berufsverband)
- Verband — Person/Institution

Muster 6: Hierarchie

Baumartige Strukturen sind bekanntlich bei Informatikern besonders beliebt. Dies mag daran liegen, dass solche Strukturen in vielen Anwendungsgebieten häufig auftreten, man nennt das zugehörige Analysemuster „Hierarchie“.

Üblicherweise sind Organisationen hierarchisch organisiert: Z.B. Konzern — Firma — Abteilung — Gruppe. Wenn man für jede dieser Teile eine eigene Klasse verwendet, dann bekommt man umständliche Probleme, weil es in der Regel gleichartige Aktionen und Strukturen mit den Teilen der Hierarchie gibt.

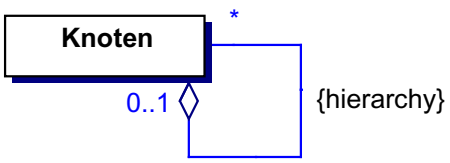
Besser ist es also, wenn man die einzelnen Objekte der Stufen aus derselben Klasse schnitzt und sie dann geschickt zu einem Baum zusammensetzt.

Namen
Hierarchie, Organisationshierarchie (Fowler), Kompositum (GoF), Stückliste (Balzert)

Zweck

- Zerlegung von Objekten in hierarchische Aggregationen.
- Objekte der Hierarchie sollen gleichartig beschrieben und gehandhabt werden.
- Die Hierarchie ist leicht erweiterbar.

Struktur



Eigenschaften

- Die Hierarchie ist als Baumstruktur gleichartiger Elemente aufgebaut.
- Jede Stufe der Hierarchie kann gleich behandelt werden.
- Die Tiefe der Hierarchie kann wechseln.
- Aufgaben können delegiert werden, ohne dass unterschieden werden muss: welches konkrete Element der Hierarchie oder welche Ebene

Beispiele

- Organisationen sind in der Regel hierarchisch aufgebaut
- Politische Gliederung der BRD
- Lager: Lagerstätten, Gebäude, Räume, Regale, Boxen
- Dokumentstrukturen: Kapitel, Abschnitt, Absatz
- Dateisysteme
- Menüstrukturen in graphischen Benutzeroberflächen

Muster 7: Rollen

Beteiligte, Dinge oder auch Orte spielen in einem Anwendungsgebiet oft eine bestimmte Rolle. Die Rolle kann man in einem UML-Klassendiagramm am Ende einer Assoziation vermerken. Es kann aber auch sein, dass mit der Rolle eigene Attribute und Methoden verbunden sind – in diesem Fall wird die Rolle eine eigenständige Klasse.

Beispiel: An einem Seminar an einer Hochschule nehmen Zuhörer und Vortragende teil. Ein Student kann in einem Seminar mehrere, verschiedene Rollen spielen: an einem Termin hat er den "Hut"des Vortragenden auf, an einem anderen Termin beteiligt er sich in der Rolle des Zuhörers und Mitdiskutanten.

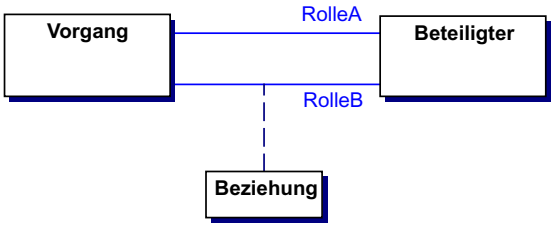
Namen
Rollen (Balzert, Coad)

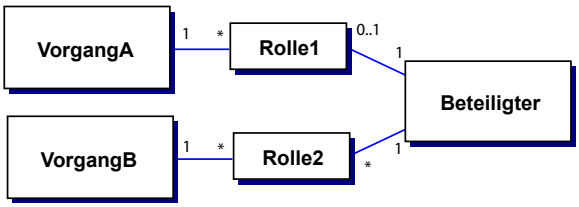
Zweck

- Darstellung der Art der Beteilung von Personen (allgemeiner: Party, Place, Thing) an einem Ereignis, einer Aktivität.

- Rollen spezifizieren die Assoziation zweier Klassen genauer.
- Dasselbe Objekt kann in unterschiedlichen Rollen auftreten (gleichzeitig – oder auch zeitlich hintereinander)

Struktur





Eigenschaften

- Ein Objekt kann in Bezug auf die Objekte der anderen Klasse eine oder mehrere Rollen spielen.
- Die Objekte haben die gleichen Eigenschaften, der Unterschied der Rollen kommt nur durch die Beziehung auf die Objekte der anderen Klasse zustande.
- Eigenschaften der Beziehung in einer Assoziationklasse
- Variante: spezielle Objekte für die Rollen

Beispiele

- Seminar — Student
- Produktionsprozess — Produkt: Rolle als Halbfertigprodukt, Werkstoff u.ä.
- Aufgabe — Angestellter: Rolle als Entwickler, Dokumentator, Auftraggeber

Verwendung

- Da Objekte zur Laufzeit in ihrer Struktur nicht verändert werden können, lagert man die Rolle aus
- Objekte können dann zur Laufzeit Rollen annehmen oder ablegen

Muster 8: Koordinator

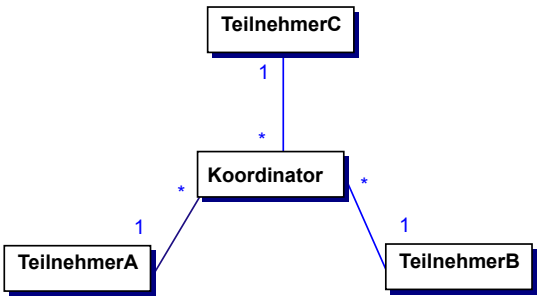
Häufig findet man in einem Anwendungsgebiet die Situation, dass verschiedene Klassen in einem Zusammenspiel auftreten. Nun könnte man versucht sein, sie paarweise durch eine Assoziation zu verbinden – oft ist es jedoch besser stattdessen einen *Koordinator* einzuführen, der für das Zusammenspiel verantwortlich ist.

Namen
Koordinator (Balzert), Mediator (GoF)

Zweck

- Eine Klasse verbindet andere Klassen, sie sorgt für ihr Zusammenspiel
- Die Beteiligten kommunizieren nicht direkt, sondern über den Koordinator
- löst mehrstellige Assoziationen auf

Struktur



Eigenschaften

- Die Koordinatorklasse ersetzt mehrstellige Assoziationen oder eine binäre Assoziation

- Oft besitzt der Koordinator selbst nicht viele Eigenschaften oder Operationen, sein Hauptzweck ist seinem Namen entsprechend.
- Der Koordinator vermittelt die Teilnehmer, ohne dass jeder Teilnehmer jeden anderen kenn muss.

Beispiele

- Mehrstellige Assoziationen sind leicht zu finden:
  - Flug — Passagier – Sitzplatz
  - Projekt — Entwickler — Programmiersprache
  - Student – Professor – Veranstaltung
- Koordination im engeren Sinn
  - Steuerung von Logik in einem Dialog
  - Webserver, der Anfragen entgegennimmt und an verantwortliche Komponenten weiterleitet

Muster 9: Wechselnde Rollen

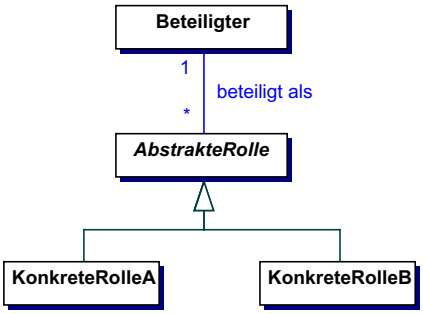
Das Muster *Wechselnde Rollen* erweitert das Muster *Rollen*, in dem eine Dynamik ins Spiel kommt: Beteiligte haben Rollen, die auch wechseln können.

Namen
Wechselnde Rollen (Balzert), Role Object (Fowler)

Zweck

- Eine Person (besser Party), ein Ding oder ein Ort spielt eine Rolle, die wechseln kann.
- Die Rollen unterscheiden sich in ihren Eigenschaften nennenswert, sind aber als Rolle substituierbar.

Struktur



Eigenschaften

- Ein Objekt kann verschiedene Rollen spielen. In jeder Rolle sind unterschiedliche Eigenschaften wichtig.
- Die unterschiedlichen Rollen sind andererseits substituierbar.

Beispiele

- Arzt – Status
- Bankkunde kann in verschiedenen Rollen auftreten – Kontoinhaber, Schuldner, Depotbesitzer...
- Adaption einer gegebenen Klasse an einen bestimmten Kontext: Person – Elternvertreter in einer Schule, Lehrer an der Schule

Muster 10: Rollenbeziehung

Ein Beispiel: In einer Organisation gibt es verschiedene Gruppen und ein Angestellter der Organisation kann mehrere Rollen in verschiedenen Gruppen spielen: Ein Softwareentwickler ist vielleicht in einem Projekt als Entwickler eingesetzt, in einem anderen als Qualitätssicherer usw.

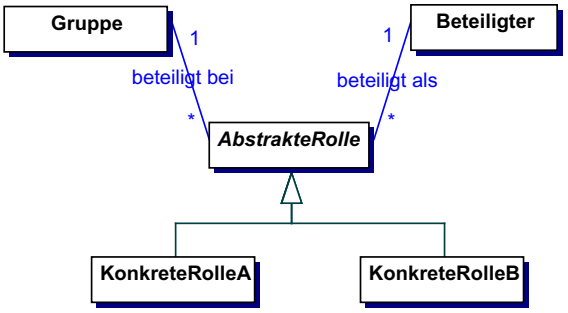
In diesem Fall haben wir nicht nur eine wechselnde Rolle, sondern jeweils einen Bezug auf eine Gruppe, in der die jeweilige Rolle gespielt wird. Für solche Situationen eignet sich das Muster *Rollenbeziehung*.

Namen
Rollenbeziehungen, Role Relationship (Fowler, Coad)

Zweck

- Eine Assoziation tritt in verschiedenen Ausprägungen auf.
- Die Rolle der Beziehung ist nicht immer vom gleichen Typ, sondern kann unterschiedliche Eigenschaften haben.

Struktur



Eigenschaften

- Eine Beziehung tritt in unterschiedlicher Gestalt auf.
- Die Beziehung selbst wird durch eine abstrakte Rolle dargestellt, die konkrete Ausprägungen hat.
- Eine Variante wäre eine Assoziationsklasse mit abgeleiteten, spezielleren Unterklassen.

Beispiele

- Verschiedene Aufgaben in einer Beziehung: z.B. Rollen eines Entwicklers in einem Projekt.
- Verschiedene Rollen einer Sache in einer Aktivität: z.B. Rollen als Maschine, Spielzeug, Werkzeug...

Muster 11: Historie

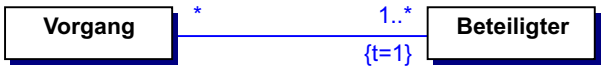
Bisher haben wir nur Muster betrachtet, die in der Regel eine Situation zu einem bestimmten Zeitpunkt beschreiben, einen *Schnappschuss*. Oft braucht man jedoch auch Informationen samt den Veränderungen im Laufe der Zeit.

Namen
Historie (Balzert), Historic Mapping (Fowler)

Zweck

- Veränderung einer Assoziation über die Zeit.
- Mehrere Objektbeziehungen treten hintereinander, aber nicht gleichzeitig auf.

Struktur



Eigenschaften

- Eine Assoziation erhält eine höhere Multiplizität, weil die Historie mitgeführt wird.
- {t=k} bedeutet, dass zu einem Zeitpunkt nur k Objektbeziehungen bestehen können.
- Oft hat die zeitlich bestimmte Klasse ein Attribut Zeitraum.

Beispiel

- Bücher, die ausgeliehen werden: Zu einem Zeitpunkt kann ein bestimmtes Exemplar nur bei höchstens einem Leser sein
- Tätigkeiten eines Angestellten
- Aufenthaltsorte einer Sache...

Muster 12: Gruppenhistorie

Das letzte der elementaren Analysemuster, das wir betrachten, bringt auch eine zeitliche Dimension ins Spiel.

Soll die Zugehörigkeit zu einer Gruppe (siehe Muster *Gruppe*) nicht nur zu einem Zeitpunkt festgehalten werden, sondern gibt es hier eine Historie, dann müssen wir die Möglichkeit haben, den Zeitraum der Gültigkeit der Zugehörigkeit zu verzeichnen: *Gruppenhistorie*.

Namen
Gruppenhistorie (Balzert), Historic Mapping (Fowler)

Struktur

- Eine Assoziation, die Zugehörigkeit bekommt eine Historie: nicht nur die aktuelle Zugehörigkeit interessiert, sondern auch die Entwicklung über die Zeit.

Struktur

Eigenschaften

- Ein Objekt gehört im Laufe der Zeit zu verschiedenen Gruppen und die historische Entwicklung wird festgehalten: dazu dient eine Zwischenklasse oder eine Assoziationsklasse
- Wie bei der Historie gibt wieder eine Beschränkung {t=k} an, was zu einem Zeitpunkt gilt.

Beispiele

- Jede Gruppe kann als Gruppenhistorie auftreten, wenn die Veränderung über die Zeit beobachtet wird.

Komplexe Analysemuster

In diesem Abschnitt werden komplexe Muster studiert: nicht nur einzelne Ausschnitte aus einem Modell, sondern das Zusammenspiel verschiedener Klassen.

Man stellt fest: Es treten immer wieder dieselben Kategorien von Klassen auf: sogenannte *Archetypen*.

Und man stellt fest: Diese Archetypen sind immer wieder in derselben Konstellation, sie bilden eine *Musterkomponente*.

Peter Coad hat sich systematisch mit diesen Musterkomponenten befasst und dabei zusammengestellt, welche typischen Konstellationen man immer wieder in Geschäftsanwendungen findet.

Dies gesehen schlägt er vor, bei der Analyse mal umgekehrt vorzugehen: Man startet mit einem Prototypen der typischen Konstellation und passt diese auf das konkrete Anwendunsggebiet an. Dies wollen wir nun studieren:

Die Archetypen:

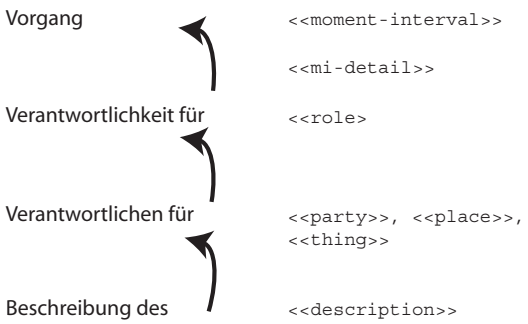
Moment-Interval - Vorgang Vorgang, Ereignis, Aktivität. In der Regel als Liste. Im Zentrum des Modells. Coad markiert diesen Archetyp durch die Farbe rot.

Role - Rolle Art, in der Beteiligte, Orte, Sachen an einem Vorgang beteiligt sind. Farbe: gelb.

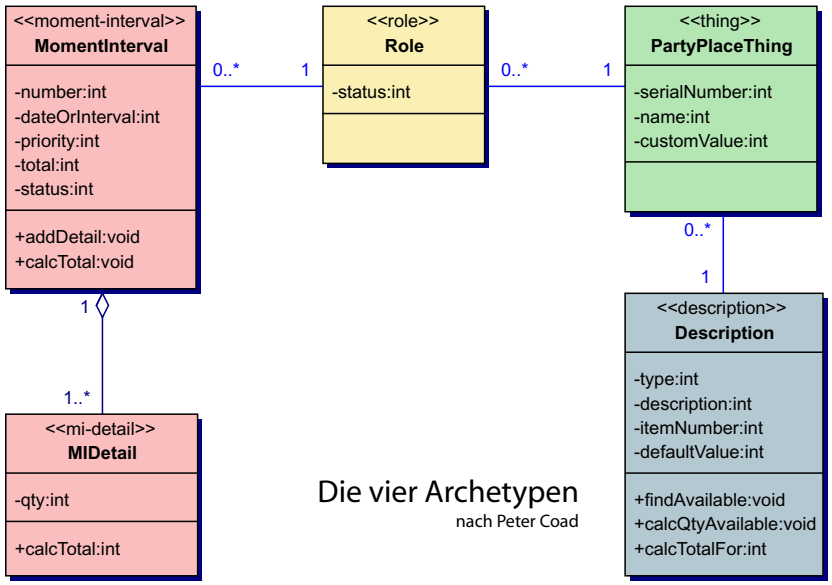
Party, Place, Thing - Beteiligte, Ort, Sache Das, was an dem Vorgang beteiligt ist. Farbe: grün.

Description - Beschreibung Beschreibung der Beteiligten, Orte und Sachen. Farbe: blau:

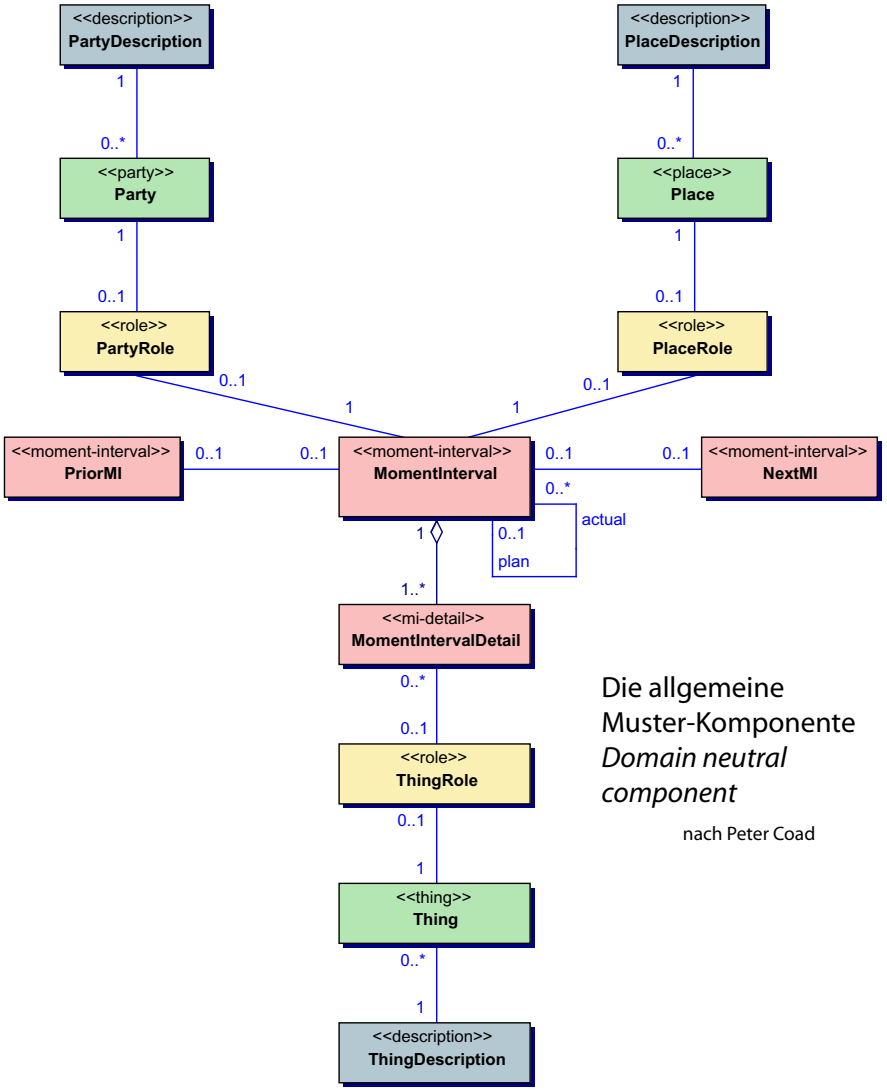
Zusammenhang der Archetypen



Die 4 Archetypen



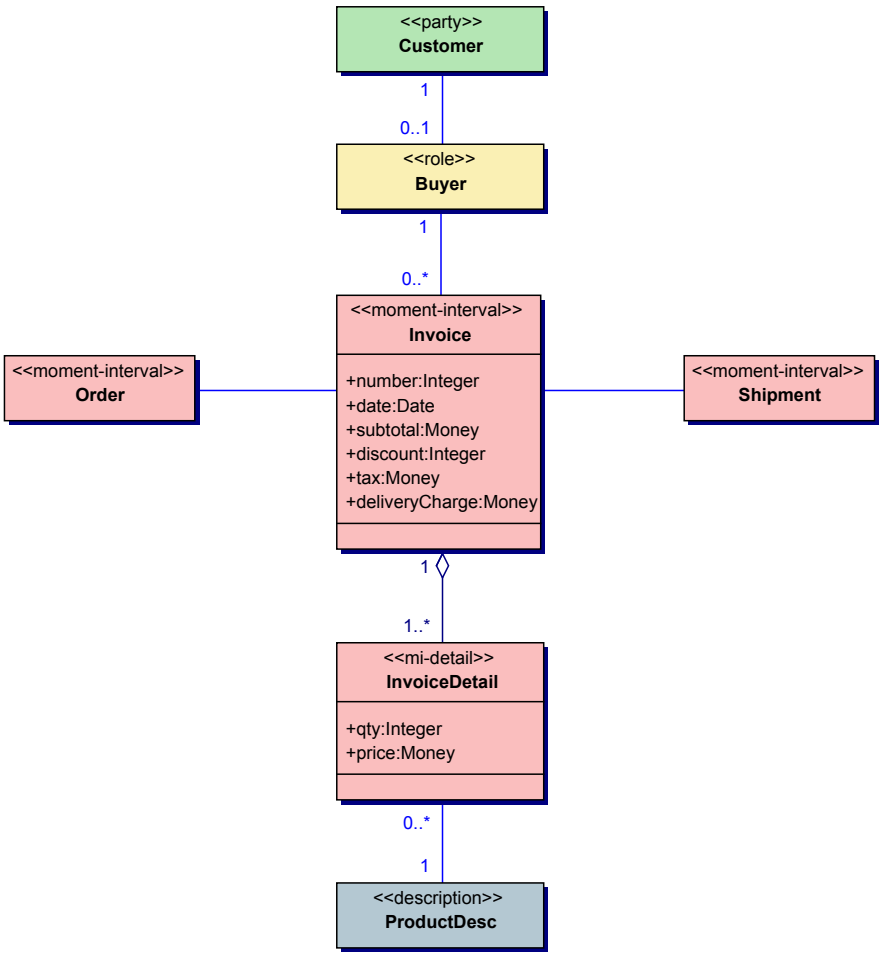
Die allgemeine Muster-Komponente



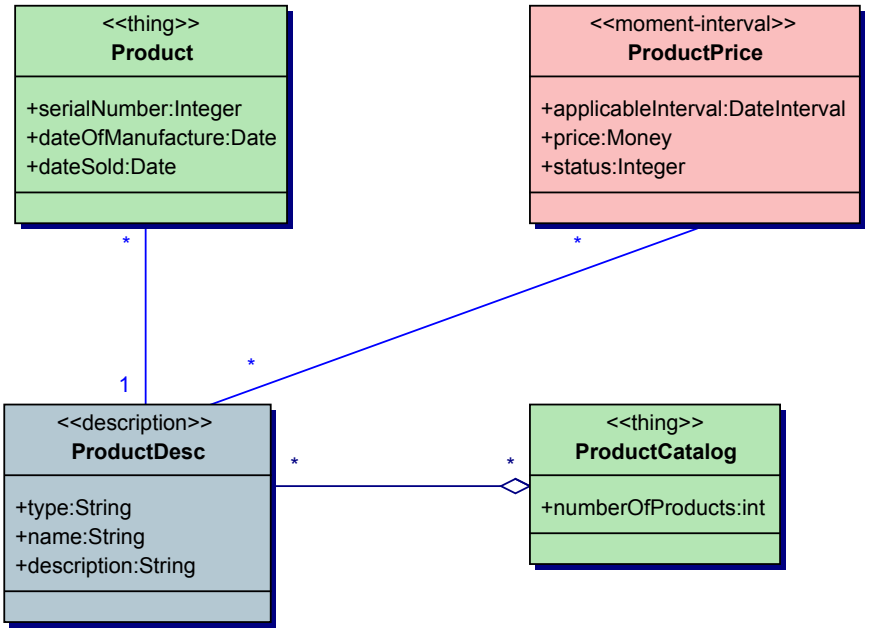
Wie gesagt: Die Idee besteht nun darin, von der allgemeine Muster-Komponenten zu starten und dann diejenigen Klassen und Assoziationen zu verwenden, die man für die konkrete zu analysierende Situation benötigt.

Auf den folgenden Seiten kommen Beispiele, die aus dem Buch von Peter Coad stammen und die ich für die Vorlesung etwas vereinfacht habe – damit das Prinzip deutlich wird.

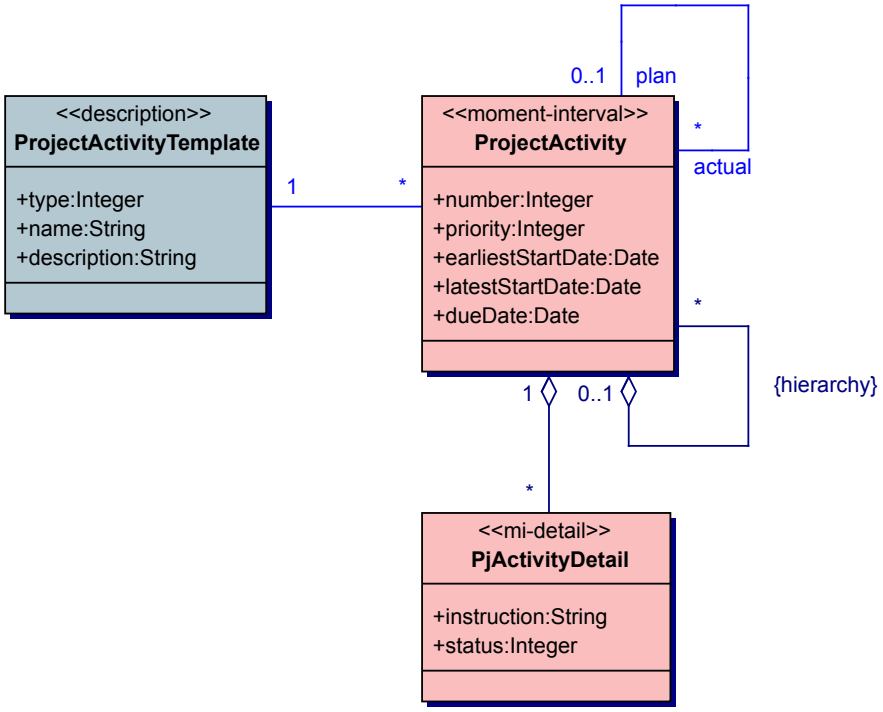
Beispiel Rechnung



Beispiel Produkt



Beispiel Projekt



Fazit

Die Domänenanalyse beschreibt die für die Konstruktion der Software wesentlichen Gegebenheiten im Fachgebiet. Sie stellt das gemeinsame Vokabular für die Fachexperten und die Softwareentwickler bereit.

In vielen Fällen ändern sich die Fachgebiete viel langsamer als die Software, d.h. das Fachmodell als Ergebnis der Domänenanalyse ist oft sehr stabil und überlebt in seiner Grundstruktur auch schnelle technische Entwicklungen in der eingesetzten Software.

Burkhardt Renz
Technische Hochschule Mittelhessen
Fachbereich MNI
Wiesenstr. 14
D-35390 Gießen

Rev 3.0 – 16. April 2012