# UML – Ein Überblick

Erstellt von:
Isabel Drost (if99wP1)

SendMail@isabel-drost.de
http://www.isabel-drost.de

*UML – Ein Überblick* ..... *1*

*1 UML – Unified modelling language* ..... *4*

**1.1 Agenda** ..... **4**
1.1.1 Was ist ein Modell? ..... 4
1.1.2 Warum modelliert man? ..... 4
1.1.3 Modellierung vs. Coding ..... 4
1.1.4 Modelle vs. Diagramme ..... 5

**1.2 Was ist UML?** ..... **5**
1.2.1 Historie ..... 5
1.2.2 Was ist UML nicht? ..... 5

**1.3 UML – konkrete Sprachmittel** ..... **6**
1.3.1 ModellTypen ..... 6
1.3.2 Diagramm Typen ..... 6

**1.4 UML-Architektur** ..... **7**
1.4.1 Wie ist das UML-Metamodell aufgebaut? ..... 7
1.4.2 Extension Mechanisms ..... 7
1.4.3 Relationships ..... 8
1.4.4 Use Case Diagramme ..... 8
1.4.5 Klassendiagramme ..... 9
1.4.6 Statechart Diagram ..... 10
1.4.7 ActivityDiagramme ..... 11
1.4.8 Kollaborationsdiagramme ..... 11
1.4.9 Sequenzdiagramme ..... 12
1.4.10 KomponentenDiagramme ..... 13
1.4.11 Deployment Diagramme ..... 14

**1.5 Profile** ..... **14**

**1.6 FutureTrends** ..... **14**

*2 Abkürzungsverzeichnis:* ..... *16*

# 1 UML – Unified modelling language

## 1.1 Agenda

Modellierung bei der Softwareentwicklung
- Warum modellieren wir?
- Modellierung vs. Coding
- Modelle vs. Diagramme

UML – Was ist das?
- Geschichte
- Was ist UML nicht?
- Was ist damit beschreibbar?
- Spracharchitektur

UML-Grundlagen
- Use Cases
- Klassendiag.
- Verhaltensdiag.
- Implementationsdiag.
- Profile

Zukunftstrends

### 1.1.1 Was ist ein Modell?

Modell=Abstraktion eines Systems oder eines Prozesses, bestimmte Details werden weggelassen, andere hervorgehoben.

- bieten eine spezielle Ansicht eines Systems/ Prozesses
- zeigen unterschiedliche Aspekte einer Sache
- sind einfacher zu verstehen als das Original (denn es fehlen ja einige für uns unwichtige Details)

Es ist möglich, von ein und derselben Sache unterschiedliche Modelle zu erstellen. Das Ganze wird durch ein Set verschiedener Modelle erklärt.

### 1.1.2 Warum modelliert man?

Komplexe Systeme und Prozesse können von einer Einzelperson gar nicht mehr in ihrer Gesamtheit verstanden werden. Man konzentriert sich vielmehr auf bestimmte Modelle.

- Erleichtertes/ besseres Verständnis des Systems (Verstehen von verschiedenen Aspekten (Architektur, Funktion, ...), Verständnis auf verschiedenen Detailebenen)
- Bessere Kommunikation zwischen Managern, Enwicklern und Kunden.

Abbildung 1-1; Kommunikationshilfe UML

### 1.1.3 Modellierung vs. Coding

Modellierung bietet:
- Unterschiedliche Perspektiven und Modelle auf ein System/Prozeß
- ermöglicht es, unterschiedliche Abstraktionsniveaus einzunehmen
- ermöglicht angemessene Notationen für unterschieldiche Perspektiven
- zum Verstehen des Gesamtsystems, muß die Gesamtsicht aus den einzelnen Modellen zusammengebaut werden
- Nachteil ist, dass das Wissen über das Systems sich auf alle das System beschreibenden Modelle verteilt

Codierung:
- bietet nur die Perspektive des Quellcodes, was eine sehr detaillierte Perspektive ist
- Verständnis aus dem Quellcode heraus setzt eine neuerliche Modellbildung voraus

Nachteile:

- Funktion ist in einer Art der Implementierung festgeschrieben (von vielen möglichen)
- Einige Konzepte können nicht adeqat umgesetzt werden.
- Platform abhängige Darstellung

Der Code ist angemessen um eine Implementation zu repräsentieren, unzureichend, um das System in seiner Gesamtheit zu repräsentieren.

### 1.1.4 Modelle vs. Diagramme

- Modelle beinhalten eine Abstraktion eines Systems/ Prozesses
- Diagramme bieten verschiedene Sichten auf die Modelle, das Diagramm entspricht quasi einem Fenster. Verschiedene Fenster können auf die gleichen Inhalte blicken, erst durch alle Diagramme wird das gesamte Modell beschrieben.

## 1.2 Was ist UML?

UML ist eine Sprache, die die Spezifikation von wichtigen Teilen/ Modellen eines Softwaresysteme erlaubt. Es sind mit UML nicht alle Modelle beschreibbar, z.B. ist ein Architekturmodell nicht möglich.

UML definiert verschiedene Elemente zur Beschreibung von Modellen. Diese werden definiert durch folgende Kritierien:

- Syntax – visuell definiert
- Semantik – noch nicht formal definiert (an einigen Stellen können also Mehrdeutigkeiten auftreten)
- Well-Formedness Rules (siehe UML-Spec Read_it\toRead\shine\UML_And_Action_Language_754.pdf)
- Notationen

UML wird standardisiert von der "Object-Management-Group", ist ein weithin akzeptierter oo-Ansatz. Unterstützt typische Modelle und Diagramme, die genutzt werden können für die Anforderungsdefinition, Analyse, Design, Implementierung. Sie ist erweiterbar und untstützt die Dokumentation, Visualisierung und Konzeption von SW-Systemen.

### 1.2.1 Historie

In den 80ern fand die OO den Weg aus den Labors in die industrielle Anwendung. In den frühen 90ern gab es viele ähnliche Ansätze zur Analyse- und zum Designmethoden und –sprachen.

'95 entstand die Unified Methode von Booch und Rumbaugh.
'96 UML von Booch, Rumbaugh und Jacobson ("the three amigos").
'97 Request for Proposal[1] der UML 1.0 bei der OMG method standardization group
'97 UML 1.1 wird Standard
'98 UML 1.2 bringt kleine Änderungen
'99 UML 1.3 bringt signifikante Änderungen und Erweiterungen
'01 UML 1.4 bringt signifikante Erweiterungen
'02 UML 1.4 mit Action Semantik (inoffiziell 1.5)
'03 UML 2.0 signifikante Umstrukturierung, verbesserte Semantik, neue Diagramm Typen

### 1.2.2 Was ist UML nicht?

- ist keine Programmier- sondern eine Modellierungssprache
- hat ein weitaus höheres Abstraktionsniveau
- UML reicht nicht, um detailliertes Verhalten zu spezifizieren (vs. Action Language...)
- Programmiersprachencode kann von UML-Modellen generiert werden (-> model compiler, Shine)

UML ist keine Methode[2]!

- Methoden nutzen UML
- Methoden können mit UML beschrieben werden
- UML ersetzt eine Methode nicht

[1] RFP/ RFC?
[2] wie z.B. Extreme Programming

# 1.3 UML – konkrete Sprachmittel

## 1.3.1 ModellTypen

UML definiert keine vollständigen ModellTypen sondern nur Elemente, die bestimmte Modelle beschreiben können. Ahand dieser Elemente, sind aber verschiedene Modelle denkbar.

Beispiel:

- Use – Case Modell – Requirements capture (wie verwenden potentielle Nutzer das System und was erwarten sie an Funktion)

- Analyse Modell – Beschreibung des Problems mithilfe von Klassen, Packages. Stellt das Verständnis des zu modellierenden Problems sicher.

- Design Modell – Klassen, Packages, Subsysteme, die das Problem lösen

- Implementations Modell – Realisierung der gefundenen Lösung mit den gleichen sprachlichen Mitteln (Klasse, Subsystem) aber alternativ auch mit Implementationssprachenspezifischen Mitteln.

- Deployment Modell – Integration und Installation (Was läuft wo und auf welchem Rechnersystem, im verteilten System, wie sind die Rechner vernetzt?)

- Test Modell – Test

## 1.3.2 Diagramm Typen

- Use Case Diagramm
- Klassendiagramm (Struktur des Problems, der Lösung und Implementierung mit Klassen, Interfaces ...)
- Verhaltensdiagramme
  - Statecharts
  - Activity Diagramme (Erweiterung/ Spezialisierung der Statemachines)
  - Interaktionsdiagramme (Kollaboration, Sequenz)
- Implementationsdiagramme (Komponenten Diagramme (wie ist die Implementation aufgeteilt in Komponenten, die wiederum aus verschiedenen Klassen/ Interfaces bestehen. Die Komponenten sind installierbar auf Rechnerknoten und austauschbar. Die Installation wird dann im Deployment Diagramm erfolgen.)

Denkbar wären auch Objekt-, Package-, Subsystemdiagramme, Modelle, Erweiterungen.

# 1.4 UML-Architektur

- Meta-metamodell – Eine Metamodell Infrastruktur. Definiert eine Sprache für die Beschreibung von Metamodellen (subset von UML, MOF von der OMG)
- Metamodel – Instanz des Meta-metamodells. Definiert die Sprache, zur Beschreibung von Modellen (e.g. beschreibt sie die UML)
- Modell – Instanz des Metamodells. Definiert eine Sprache Informationen zu modellieren.
- User Objects – Instanz des Modells, definiert eine spezifische Umsetzung.

## 1.4.1 Wie ist das UML-Metamodell aufgebaut?

Definiert die UML wie folgt[3]:

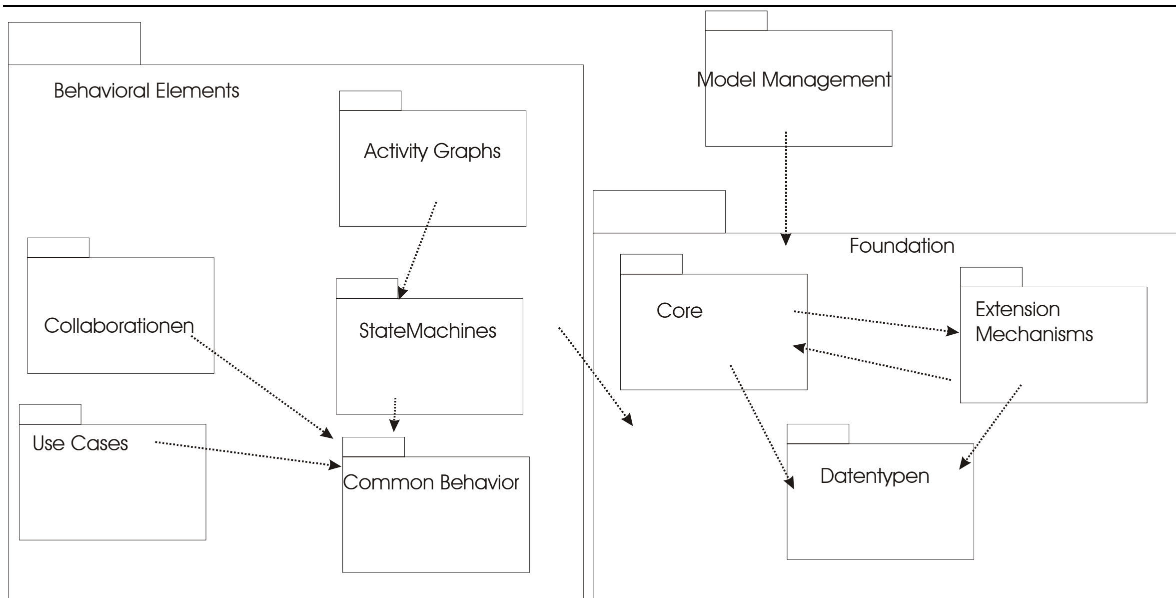



Abbildung 1-2; Das UML- Metamodell

Abbildung 1-3; Von UML definierte Elemente - Überblick

## 1.4.2 Extension Mechanisms

UML kann transparent erweitert werden durch Stereotypen, Tagged Values, Constraints. `<<Stereotypen>>` sind dabei: "Sie bieten einen Weg, der zusätzlichen Klassifizierung von Modellelementen außerhalb von Vererbungshierarchien, ohne das UML-Metamodell um neue Elemente zu erweitern. Erweiterungen wirken wie neue Metamodellklasen." Einige sind vordefiniert. Andere durch den Entwickler mit viel Vorsicht und Umsicht hinzuzufügen.

[3] "The Rational Unified Process" 3 Amigos, Addison Wesley und "Object Oriented Software Development" Anton Eliens

Tagged Values sind explizite Name-Wert (`tag=value`) Paare, die jedem verwendeten Modellelement zugeordnet werden können. Können zusätzliche Informationen beinhalten, die über Vererbungsinformationen hinausgehen. Einige sind vordefiniert.
`{Constraints}` sind zusätzliche Beschränkungen, die den verwendeten Modellelementen aufgelegt werden können. Einige sind vordefiniert. Die zu ihrer Definition verwendete Sprache ist die OCL (Object Constraint Language), andere Sprachen sind aber ebenfalls möglich. OCL wurde genutzt, um die UML well-formedness Rules zu deklarieren.

### 1.4.3 Relationships

Dependency/Abhängigkeit – Beziehung zwischen beliebigene Modellelementen, eine Änderung des einen Modellelements zieht sehr wahrscheinlich eine zwangsläufige Änderung im Dependencyclient – einem anderen Element – nach sich.

Abstraktion – der Dependenyclient stellt das gleiche dar, wie der DependencySupplier, nur aus einem anderen Blickwinkel

Realisierung – ein Modellelement realisiert Konzepte, die ein anderes Modellelement (der Realizationsupplier) vorgibt (z.B. besteht diese Beziehung zwischen Klasse und Interface)

Generalisierung – Beziehung zwischen einem allgemeineren und einem spezifischeren Element. Das spezifische Element ist voll kompatibel mit dem generelleren Element und beinhaltet zusätzliche Informationen. Ein spezifischeres Element kann überall da genutzt werden, wo das generelle auch genutzt wird.

Assoziation – Es besteht eine Kommunikationsbez. zwischen den Instanzen von Classifieren und Klassen im speziellen. Z.B. ein Objekt ruft die Methoden des anderen auf, auf Objektebene ist dies ein Link in UML, auf Classifierebene handelt es sich um eine Assoziation.

Aggregation – die Objekte einer Klasse enthalten Objekte einer anderen Klasse (Auto – Rad Beziehung). Die Einzelteile sind aber auch lebensfähig ohne den Container.

Komposition – spezielle Form der Aggregation, die aussagt, dass jedes Objekt a, das Objekte der Klasse b enthält, dafür verantwortlich ist, diese Objekte anzulegen und wieder freizugeben.

Abbildung 1-4; Beziehungen zwischen Objekten

### 1.4.4 Use Case Diagramme

Visualisiert Relationships zwischen Use-Cases und Aktoren in einem System. Nutzer sitzen außerhalb des Systems. Use Cases definieren die Grenzen des angestreben Systems. Sie visualisieren die Funktion des Systems, wird häufig eingesetzt, um die Anforderungen an das System zu beschreiben.

Use Case= Spezifikation einer Sequenz von Aktionen, die durch einen Aktor ausgelöst wird. Use Cases sind Classifier, d.h. sie können andere UC erben, erweitern und in anderen UC enthalten sein. Sie werden durch Klassen und Subsysteme realisiert, die nicht Bestandteil von UC sind.

Der zweite Bestandteil von UC sind Aktoren (kohärente Menge von Rollen, die die Nutzer von UC einnehmen können, wenn sie mit den UC interagieren). Ein Aktor hat in einem UC genau eine Rolle. Auch Aktoren sind Classifier. Sie repräsentieren Menschen, Machinen, Organisationen ...

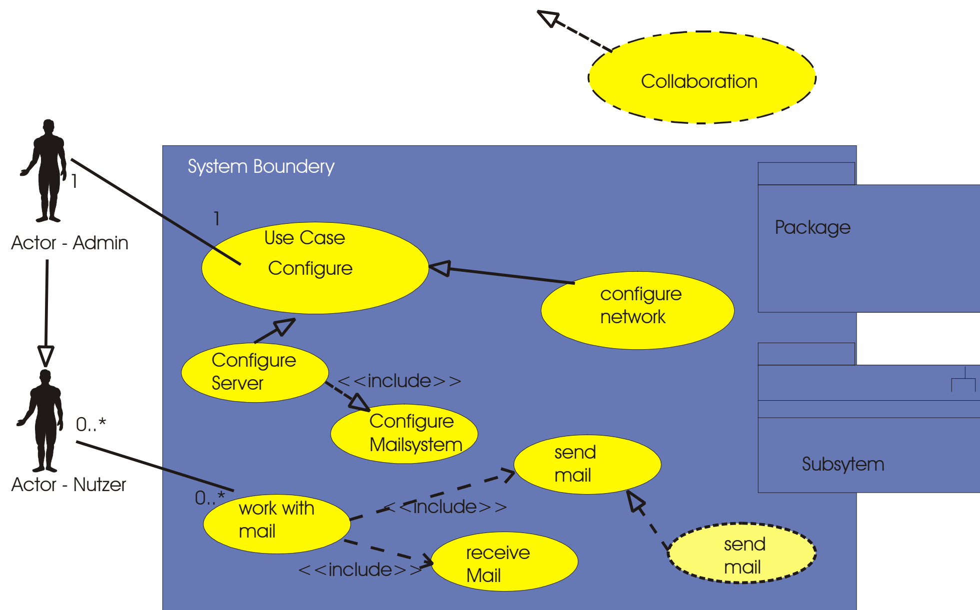



Abbildung 1-5; Use Cases - Aufbau

Relationships in UC:

- Actor – UC … Actor erwartet einen Service, der vom UC zur Verfügung gestellt wird.
- Actor – Klasse/ Subsystem ... Actor sendet eine Nachricht an die Klasse, die dann eine durch einen Use Case spezifierte Funktion ausführt.

## 1.4.5 Klassendiagramme

Zeigen Klassen , Interfaces, Pakete, Modelle, Subsysteme, Objekte und ihre Bezihungen. Zeigen die Struktur von UML-Modellen.

Abbildung 1-6; Elemente im Klassendiagramm

UML-Architektur

Klasse: Eine Beschreibung eines sets von Objekten, die die gleichen Attribute, oberationen, Mothoden, Ralationships und Semantik teilen. Eine klasse kann ein Set von Interfaces spzifizieren, Collections von Operationen stellen ihre Umgebung dar.
Können andere Klassen spezialisieren oder genalisieren, können Interfaces realisieren.

Interfaces ist ein benanntes Set von Operationen, die das Verhalten eines Elementes charakterisieren.

Ein Subsystem ist eine Gruppierung von Elementen, die eine in sich geschlossenes Verhalten repräsentiert. Es bietet Schnittstellen, die es implementiert. Die konkrete Implementierung erfolgt durch die enthaltenen Klassen.

Packages dienen der Gruppierung von Modellelementen.

Ein Objekt ist ein Entity mit einer definierten Grenze und Identität (z.B. Adresse des Objekts im Speicher). Es kapselt einen Status und Verhalten. Ein Objekt ist eine Instanz einer Klasse.

## 1.4.6 Statechart Diagram

... ist ein Diagramm, das eine Statemachine repräsentiert. StateMachine ist ein Diagramm, das die Sequenzen von Stati spezifiziert, die ein Objekt oder eine Interaktion während seines Lebens durchlaufen werden kann, indem auf best. Ereignisse reagiert wird. Beschreibt konzeptuell, auf Spezifikations- oder Implementationsebene, welche Zustände die beschriebenen Objekte einnehmen können.
Sie werden genutzt in Classifiern/ Klassen, um das Verhalten ihrer Instanzen zu spezifizieren. Die StateMachine besteht aus Zuständen, Pseudo-Zuständen und Transitionen. UML-Statemachines sind eine Verallgemeinerung der Harel Statecharts (Hierarchy (Zustände werden hierarchisch geschachtelt)+Orthogonality (mehree Zustände können parallel eingenommen werden)), Verallgemeinerung der Finite Statemachines (Read_it\toRead\shine_DONE_1997_essential_issus_in_codesign_46.pdf).

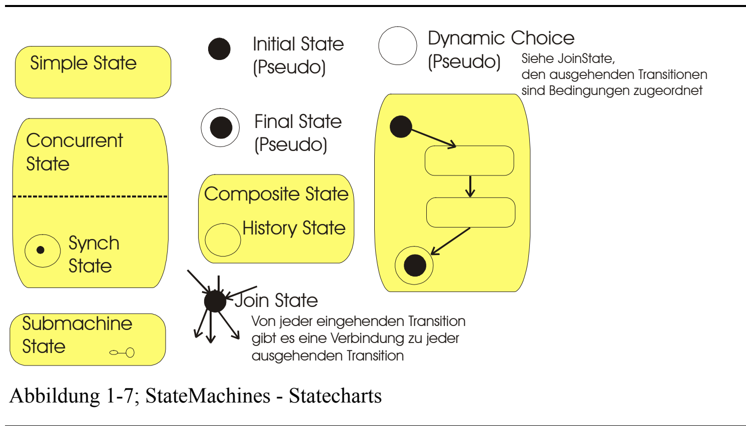
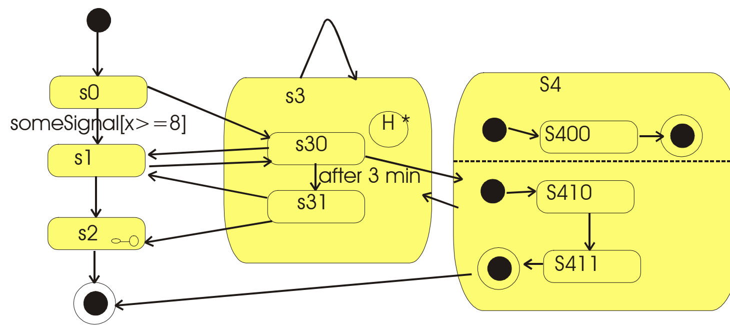


Abbildung 1-7; StateMachines - Statecharts

State: Bedingung oder Situation, die während der Lebenszeit eines Objektes auftreten kann. Während dieser Status besteht, kann das Objekt Aktionen ausführen oder auf Events warten. Auch beim Übergang in einen anderen Zustand kann das Objekt best. Aktionen ausführen.

Initialzustände, History States

Eine Transition ist eine Bez. zwischen (Pseudo-)zuständen, die besagt, dass das Objekt von einem in einen anderen Zustand übergeht. Sie wird unter Erfüllung best. Bedingungen oder bei best. Ereignissen durchgeführt.
Aktivitäten werden ausgeführt beim Verlassen eines Zustandes, beim Zustandübergang und beim Einnehmen eines neuen Zustands ausgeführt (Actions = Abstraktionen von Berechnungsprozeduren). Sie rufen meist eine Änderung des Zustandes des Objektes, in dessen Zustands sie ausgeführt werden, hervor.




S4 ist ein Orthogonaler State, bei dem S400 und S410/411 gleichzeitig ausgeführt werden können. Bei Eintritt werden beide InitialStates aktiv, bei Austritt beide Chats inaktiv.

Abbildung 1-8; StateChart Beispiel

### 1.4.7 ActivityDiagramme

Diagramm zeigt einen Aktivitätsgraphen. Es ist ein spezieller StateGraph, der genutzt werden kann, um Prozesse zu modellieren, die mehrere Klassen beinhalten.

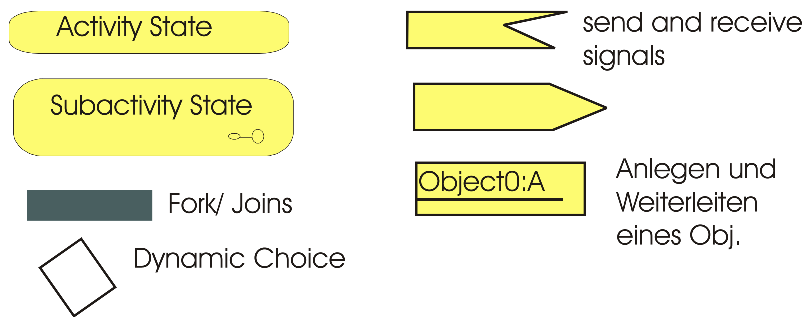



Abbildung 1-9; Activity Diag.

### 1.4.8 Kollaborationsdiagramme

... zeigen den Nachrichtenaustausch zwischen Objekten. Im Sinne der UML handelt es sich um zusammenarbeitende Objekte. Innerhalb einer Kollaboration spielen die Objekte bestimmte Rollen.
Sie werden während der Analyse und im Design verwendet, z.B. um die Requirements der UC besser zu analysieren und zu verstehen. Sie sind Classifier und unterliegen so den normalen Vererbungsmechanismen. Sie beinhalten Interaktionen zwischen Objekten. Eine Interaktion beschreibt, wie Objekt1 Nachrichten an Objekt 2 schickt. Hier spricht man allerdings nicht von Nachrichten, sondern von Stimuli.
Wie die Stimuli zu behandeln sind, ist mithilfe von StateCharts oder Methoden zu beschreiben.

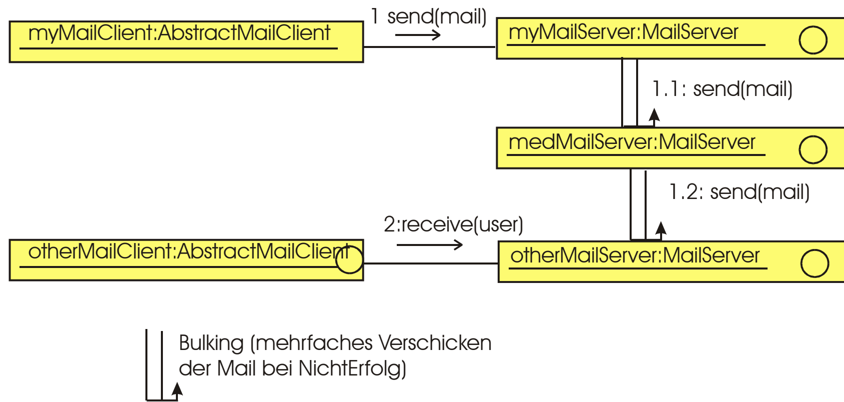



Abbildung 1-10; Beispiel eines Sequenzdiagrammes

## 1.4.9 Sequenzdiagramme

- drücken das gleiche aus, wie Kollaborationsdiagramme, allerdings auf der Ebene von Classifiern
- werden genutzt, um Szenarios aufzustellen

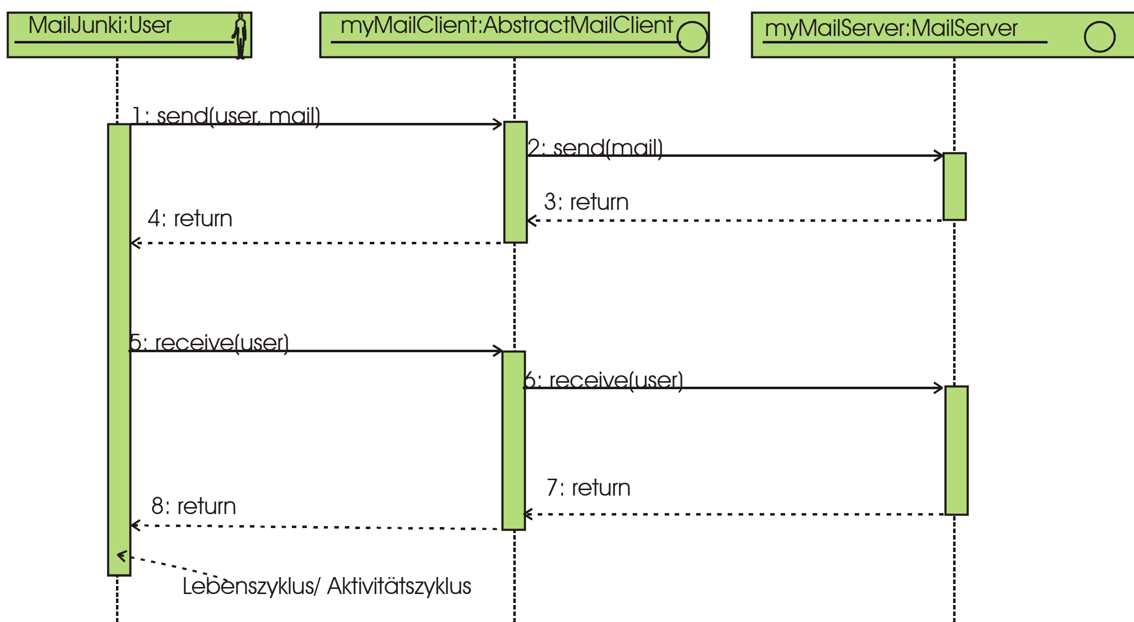



Abbildung 1-11; Sequenzdiagramm- Beispiel

## 1.4.10 KomponentenDiagramme

Eine Komponente ist eine modulare, installierbare und austauschbare Einheit unseres Systems. Sie implementiert ein eigenes Interface. Sie ist dann austauschbar, wenn die andere Komponente das gleiche Interface implementiert. Sie besteht aus Klassen und Subsystemen, die einen Systemteil implementieren. Ihre Funktion entspricht der Funktion der Klassen, die sie enthält.
Quellcodefiles und Executables, die Funktion für die Komponente zur Verfügung stellen, werden mithilfe von Artefakten dargestellt. Ein Artefakt ist also ein phys. Stück Information, das während eines [SW]Entwicklungsprozesses entwickelt und/ oder genutzt wird.

Abbildung 1-12; Komponenten Diagramm

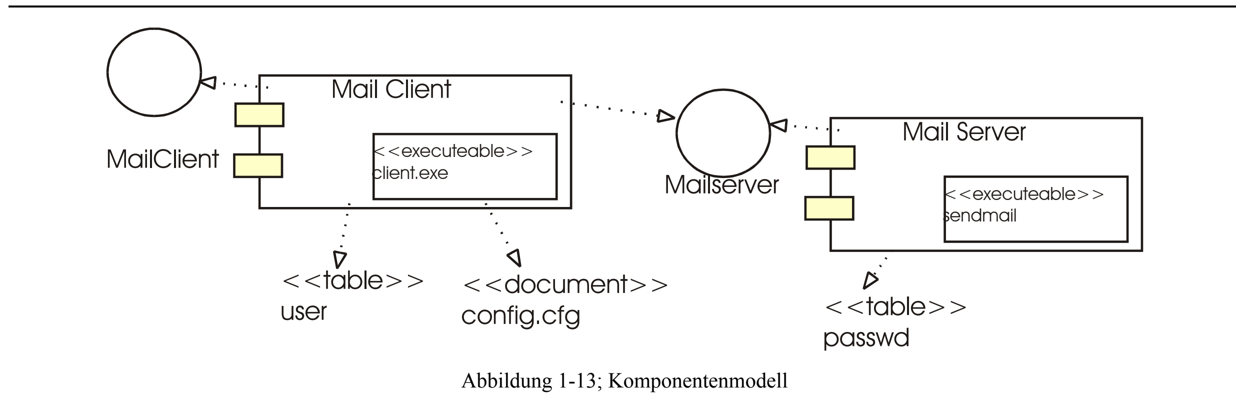



Abbildung 1-13; Komponentenmodell

### 1.4.11 Deployment Diagramme

... zeigen die Verteilung der SW-Einheiten auf verschiedene Ressourcen. Bestehen aus Knoten (sind Classifier, die erben, spez., verallg. können), die Speicher oder CPUs oder ganze Rechner repräsentieren.

Beziehungen:
BKnoten zu Knoten (Kommunikationskanäle, Busse, serielle Verbindungen...)
Knoten zu Komponenten (Komponente wird auf diesem Knoten ausgeführt)

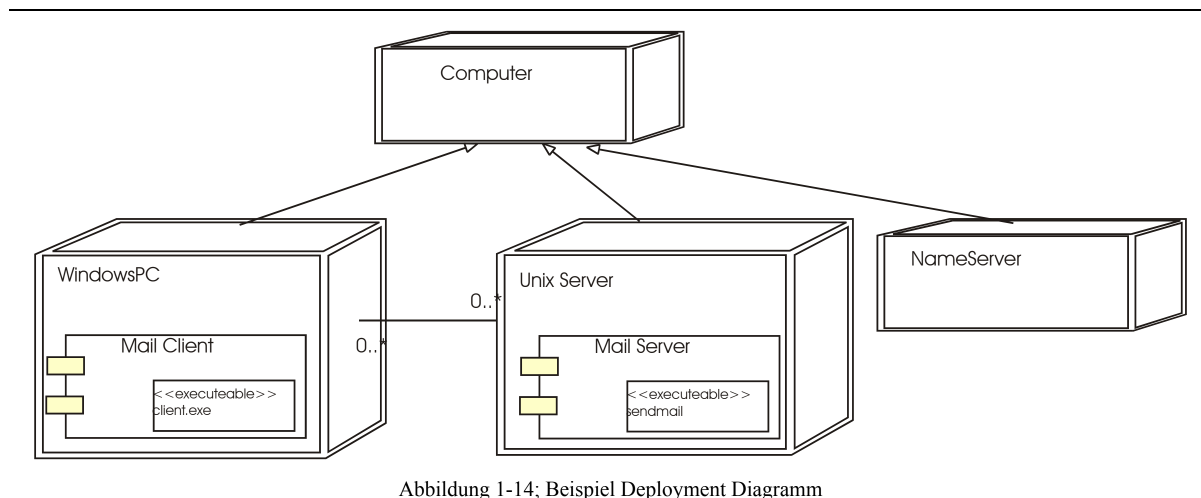



Abbildung 1-14; Beispiel Deployment Diagramm

Nicht für die detaillierte Verteilung zu nutzen!

## 1.5 Profile

UML ist eine sehr mächtige Sprache, die mehr Möglichkeiten bietet als eine spezielle Programmierumgebung bietet. Um die Nutzbarkeit von UML darauf anzupassen, kann UML-Subsets/ Profile erstellt werden. Diese werden mittels der Extensions erstellt werden und ohne Widersprüche zum UML-Metamodell beinhalten dürfen.

## 1.6 FutureTrends

- executable UML
  - komplette, präzise Systemspezifikation
  - schon auf der Ebene der UML-Modelle soll eine Verifikation und Simulation möglich sein
  - Applikationsspezifische Modellcompiler[4]
- MDA – Model Driven Architecture
  - Unterstützt den Austausch von HW- und SW-Plattformen
  - PIM → PSM → PDA
  - ermöglicht ausführbare UML

[4] Shlaer-Meller – Bridgepoint; "Executeable UML – A Foundation for MD" Meller, Balzer (im LEC da)

UML – Unified modelling language

- Profile
  - Applikationspezif. Subsets von UML
  - e.g. OMG RT-Profile für Schedulablitiy
- HW/SW-Codesign
  - Basiert auf MDA und Executable UML
  - e.g.: Shine Model Compiler for RTR Architectures
    (Read_it\toRead\shine_DONE_2002_Fröhlich_codesign.pdf und
    Read_it\toRead\shine_DONE_2003_Fröhlich_uml_compiler.pdf und
    Read_it\toRead\shine_DONE_2003_Fröhlich_umlcodesign_12.pdf )
    Übernimmt Systemverifikation, automatisches Platform Mapping von HW+SW, automatische und vollständige Synthese von HW + SW. Compiler liest PIM ein und spuckt ein PDA aus.

*Abbildung 1-1; Von Neumann Prinzip* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-2; Hochparallele Hardware - Beweis* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-3; Aufbau eines FPGA nach Xilinx* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-4; Realisierung eines CLB* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-5; lose FPGA Kopplung* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-6; Runtime Reconfigurable System - Aufbau für's Diplom* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-7; Prinzip der sicheren parallelen Entwicklung von HW und SW* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-8; Prinzip der klassischen parallelen HW-SW Entwicklung* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-9; OO Berechnungsmodell* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-10; Diamondstruktur* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-11; Co-Design von Specification zur Implementation* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-12; Plattformmapping* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-13; Möglichkeiten der HW/SW-Partitionierung* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 1-14; Synthese* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 2-1; Was ist ein Compiler?* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 2-2; Compileraufbau aus Front-/Backend und Engine in UML* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 2-3; CDFG einer If-Anweisung* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 2-4; Transformierung eines Zwischenprogramms in einen CDFG* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 3-1; Klassendiagramm der Parser von Shine* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 3-2; Gegebene Hierarchie eines Softwaremodells* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 3-3; Darstellung von Basic Blocks als Knotenhierarchie* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 3-4; Beispiel für einen unpaired Knoten: Branch Knoten Syntax im CDFG* **Fehler! Textmarke nicht definiert.**
*Abbildung 3-5; Beispiel: CDFG für den paired Knoten if/endif* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 3-6; Beispiel CDFG für eine if-Anweisung* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 3-7, while/end-while* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 3-8; allgemeine Darstellung eines switch im CDFG* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 3-9; begin/end* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 3-10; Compiler Architektur* ........ **Fehler! Textmarke nicht definiert.**
*Abbildung 4-1; Kommunikationshilfe UML* ........ 4
*Abbildung 4-2; Das UML- Metamodell* ........ 7
*Abbildung 4-3; Von UML definierte Elemente - Überblick* ........ 7
*Abbildung 4-4; Beziehungen zwischen Objekten* ........ 8
*Abbildung 4-5; Use Cases - Aufbau* ........ 9
*Abbildung 4-6; Elemente im Klassendiagramm* ........ 9
*Abbildung 4-7; StateMachines - Statecharts* ........ 10
*Abbildung 4-8; StateChart Beispiel* ........ 11
*Abbildung 4-9; Activity Diag.* ........ 11
*Abbildung 4-10; Beispiel eines Sequenzdiagrammes* ........ 12
*Abbildung 4-11; Sequenzdiagramm- Beispiel* ........ 13
*Abbildung 4-12; Komponenten Diagramm* ........ 13
*Abbildung 4-13; Komponentenmodell* ........ 14
*Abbildung 4-14; Beispiel Deployment Diagramm* ........ 14

*Gleichung 1; Beispielrechnung* ........ **Fehler! Textmarke nicht definiert.**

# 2 Abkürzungsverzeichnis:

| Abk. | Abkürzung |
|---|---|
| FPGA | Field Programmable Gate Arrays |
| PIM | Platform Independent Model |
| PSA | Platform Specific Architecture |
| PDA | Platform Dependent Application |
| UML | Unified Markup Language |
| HTML | Hypertext Markup Language |

Abkürzungsverzeichnis:

| | |
|---|---|
| XML | X-Markup Language |
| CDFG | Control Data Flow Graph |
| JVM | Java Virtual Machine |
| MDA | Model Driven Architecture |
| SMG | |
| XMI | eXtensible Model Language |
| VHDL | Very Highspeed Integrated Circuit Description Language |