

LEIBNIZ UNI HANNOVER
G-Techik und Informatik

MASTER -

ARBEIT

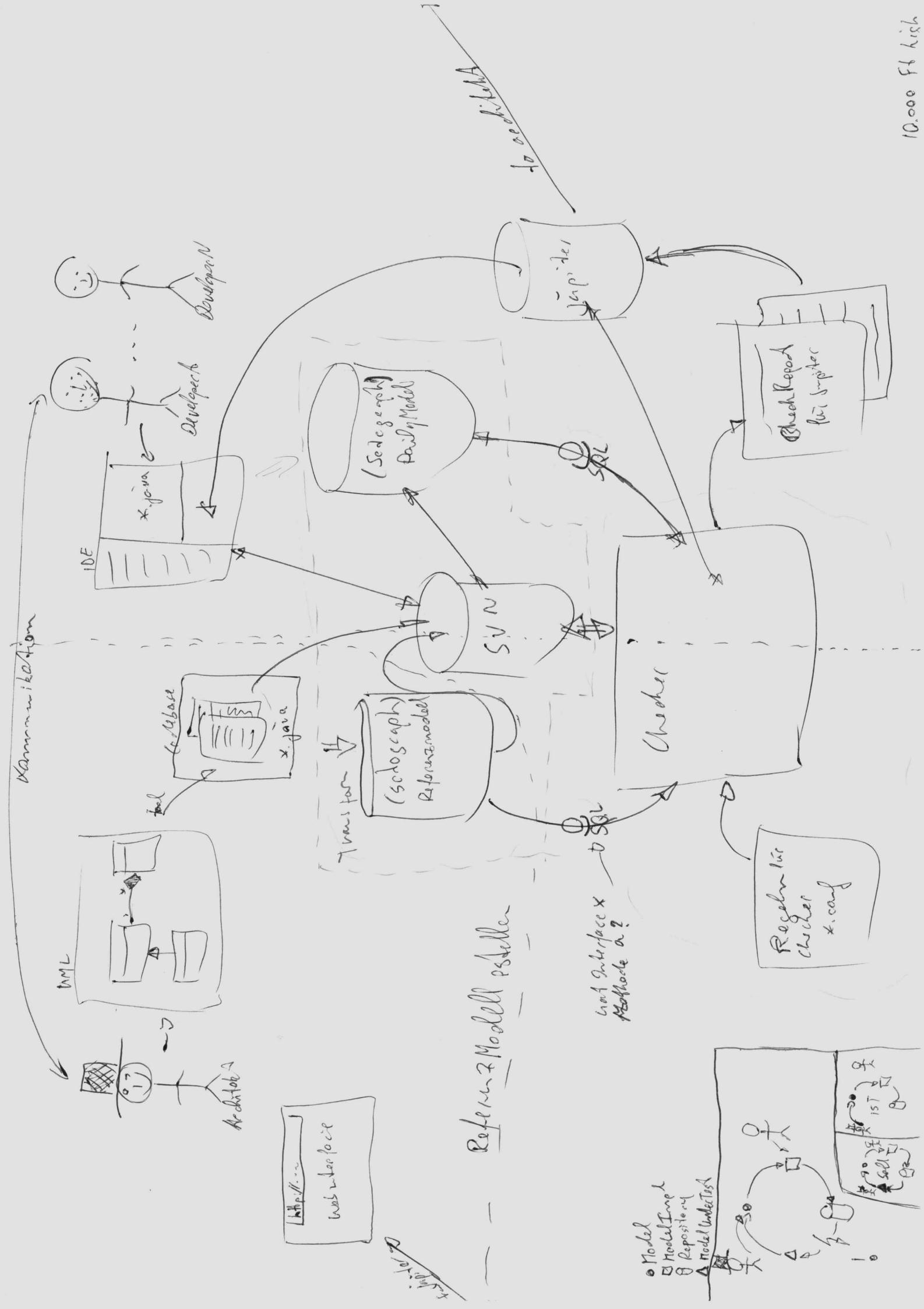
B.Sc. Jan Hinzemann

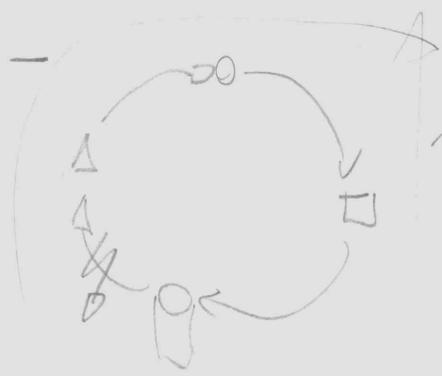
Erläuterung

Ich habe alles selbst
gemacht.

Hannover, den 5. 6. 2007

Jan Hartmann





Ausgangsproblem

- Informationsfluss (Spezies: Flow (Shmidec))
- Theoretische Teil: Codebase, wie, was,
- Fallstudie: DLR, Umfeld, Satolograph ... \rightarrow konkretes Informationsmodell
 - \hookrightarrow Checkins: verletzte Verknüpfung/Modell
 - \hookrightarrow Vorschreibe
 - \hookrightarrow erlaubter Review, Paircheckin ...
 - ...
...
- \rightarrow machine.pdf send auf se-server!
- \rightarrow Code sehr interessant (wie wir für code modellieren)

Master-Arbeit

zu vergeben



SOFTWARE ENGINEERING
UNIVERSITÄT HANNOVER

~~MoDi Satori~~

höchstens
im Absatz

Qualitätssicherung in the Loop von implementierten Architekturenmodellen

DLR

Hintergrund

Im heutigen Vorgehensmodell der objektorientierten Analyse und Design (OOAD) sind im Wesentlichen die drei Rollen des Softwarekundens, des Architekten und des Entwicklers zu identifizieren. Diese Beteiligten kommunizieren mittels UML. Der Architekt führt mit dem Kunden eine objektorientierte Analyse (OOA) durch, bei der er die Anforderungen des Kunden aufnimmt. Im Anschluss entwickelt der Architekt in der objektorientierten Design (OOD)-Phase ein Architekturenmodell, welches die Wünsche des Kunden erfüllt. Dieses sog. *Referenzmodell* wird dann vom Entwickler in das *Implementationsmodell* überführt. Im Laufe eines Softwareprojektes kommt es naturgemäß zu Modelldifferenzen (MoDi) und so muss, nach erneuter Kommunikation zwischen den Verantwortlichen, wenigstens eins der beiden angepasst werden. Damit ein Projekt zum Erfolg führen kann, muss das Implementationsmodell das Referenzmodell abbilden, damit der Kunde zufrieden gestellt wird und es nicht zu Vertragsverletzungen kommt. Dies ist leider selten der Fall.

* In dieser Masterarbeit soll untersucht werden, inwieweit sich Differenzen zwischen Referenz- und Implementationsmodell während der Entwicklung kontinuierlich und automatisiert aufdecken lassen, sodass etwaige Abweichungen frühzeitig erkannt und behoben werden können. Hierzu sollen die Bestandteile von Java-Interfaces untersucht und mögliche Änderungen dieser Bestandteile über die Zeit identifiziert werden.

Aufgabe

- wie wird
Kommunikation
überhaupt
kommen?
1. Da Referenz- und Implementationsmodell zum einen als UML-Modell und zum anderen als Code-Modell vorliegen, ist es nötig, eine Vergleichsbasis der Modelle herzustellen (Diagramm-, XMI-, Code-, Datenbankinhaltsebene).
 2. Anschließend sind Regeln für die Erkennung von Modelldifferenzen zu erstellen und es ist zu prüfen, welche Konsequenzen sich aus Regelverstößen ergeben sollten.
 3. In vielen Fällen muss Kommunikation die Konsequenz von Regelverstößen sein. Hier sind Reports, die die Ergebnisse über die Modelldifferenzen präsentieren, zu generieren und an passende Akteure zu kommunizieren.
 4. Entwurf und Implementierung eines Prototyps für dieses System.
 5. Am Beispiel des DLR eigenen Projektes *SiLEST* soll die Anwendung erprobt werden.

1. Wettbewerb
um MoDi -
Vergleich

2. Veränderungen
in Vorgehens-
modell

Zu der Arbeit ist ein Bericht von etwa 80 Seiten zu erstellen, in dem die Ergebnisse der Untersuchung beschrieben werden und die entwickelte Software beschrieben wird.

Organisatorisches

Betreuer:

intern: Daniel Lübke, 0511 762 19672, daniel.luebke@inf.uni-hannover.de

extern: Axel Berres, 030 67055 242 oder 0531 295 2769, axel.berres@dlr.de

Beginn: ab sofort

Kabinettidee Qualitätssicherung-in the Loop von implementierten Architekturenmodellen

~~Qualitätssicherung-in the Loop von implementierten Architekturenmodellen~~

- wir wollen automatisch Tools untersuchen können

1. Das System muss ~~Architektur~~ in eine interne Repräsentation umwandeln.
- Das System erzeugt die interne Rep.
- Der Architekt wandelt UML in Sourcecode um
- opt: Der Architekt publiziert sein Architekturmodell (es ist Sourcecode des Systems) - benutzt?
- Das System vergleicht IM mit RPT auf Basis von Regeln
- Das System erstellt einen Report über die Vergleich
- <> Rollen> gibt es eigene Regeln für den Vergleich fest
- Das System aktualisiert Sourcecode einen SCM
- Das System nutzt quelloffene Interfaces aus Codebasis
- Das System nutzt quelloffene Interfaces standard (IM->RPT)
- Das System vollständig Anleger, Neues, Gelöscht
- Das System benachrichtigt Rollen mit dem Report

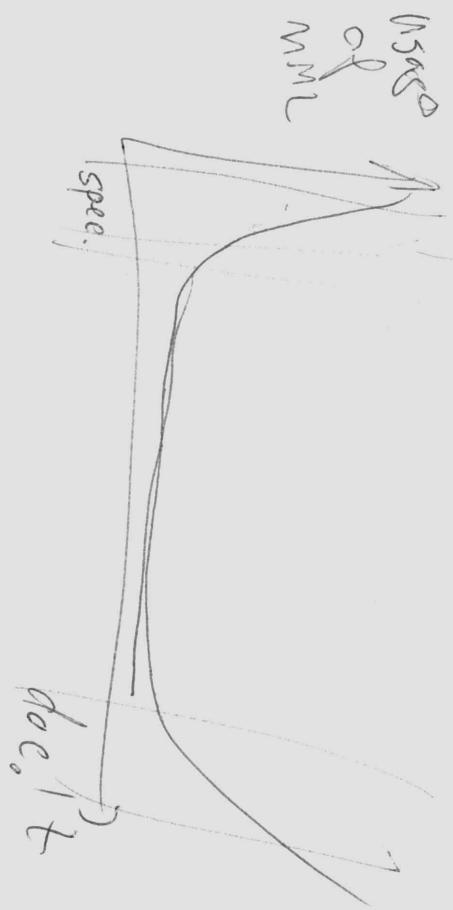
Regeln haben Severity => Check wird obwohl nicht

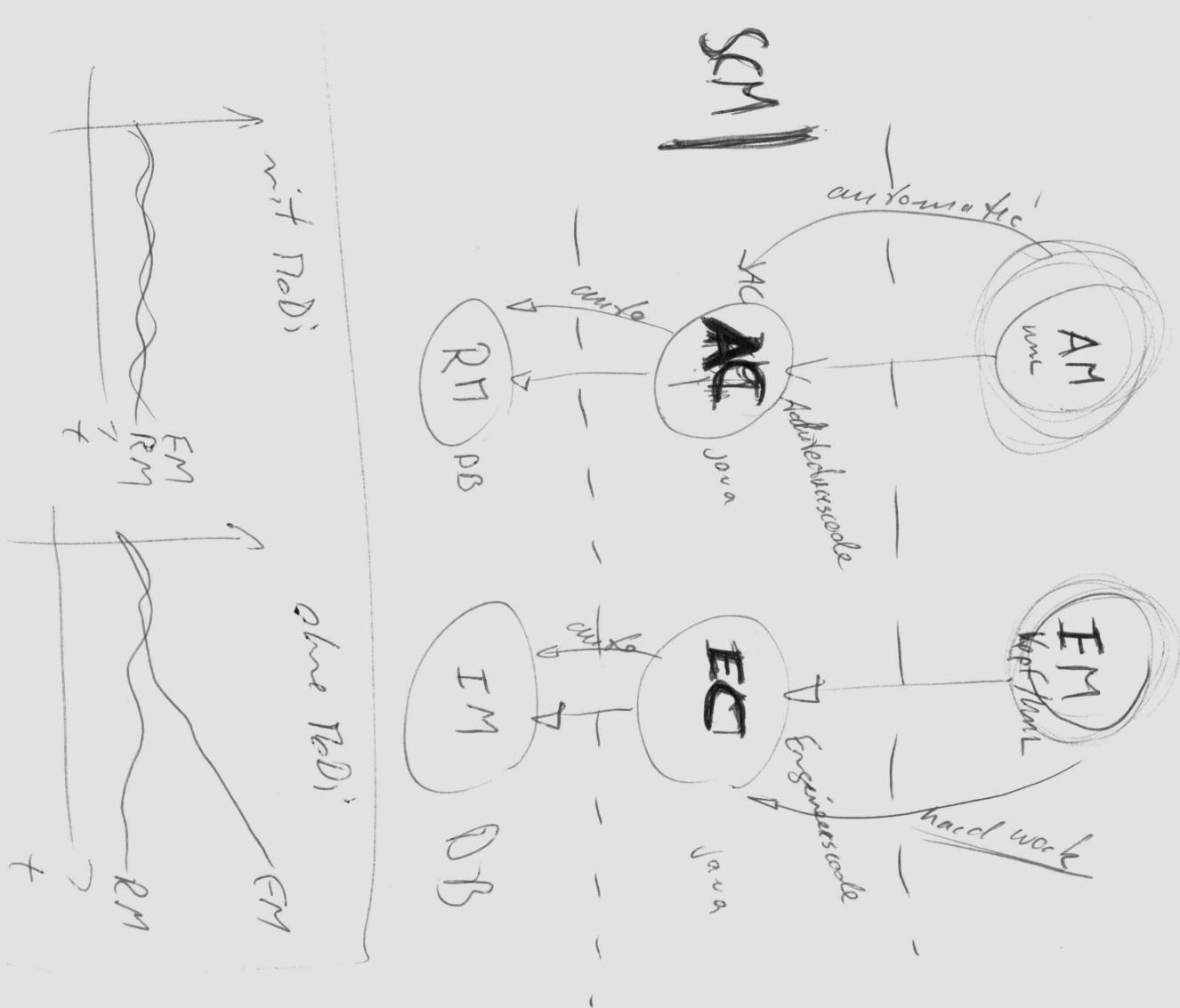
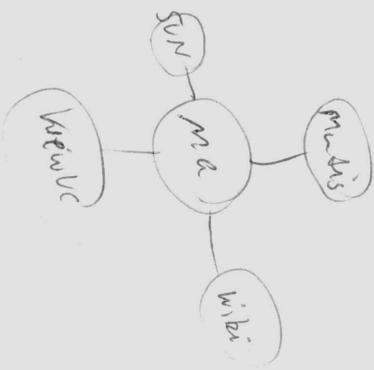
- Recomm:
Das System hält die aktuelle Revision des RMs vor und vergleicht nur die Teile aus dem Bereich des IMs
- Rec Canجب:
Das System vergleicht Revisionen vom RM und IM und führt anschließend den Vergleich durch.

TODO

- Th {
1. Untersuchung von Interfaces: was gibt es? (Method, Parameter, Rückgabewerte)
 1. 1. was darf ~~verändert~~ verändert werden nicht?
 1. 2. das kann Regel erfüllen.
 1. 3. wie wird mit Regeln umgegangen (implenfert, überprüft)
- (2. Vergleichsbasis herstellen (diag-one - diagonal)
 + Coall - coall
 3. XML - > XML)
2. 1. Evaluation { 1., 2., 3. } -> 2. 2
 2. 2. Coalescense Socialisierung kleine C file, RDBMS, XMLDB
 2. 3. Reportgenerierung / Präsentation des Ergebnisses (Jupiter/Metaplace)
- Prx {
3. 1. Anschaffung
 3. 2. ...
- Bsp {
4. Anwendung
 4. 1. auf Silo A
5. Fazit

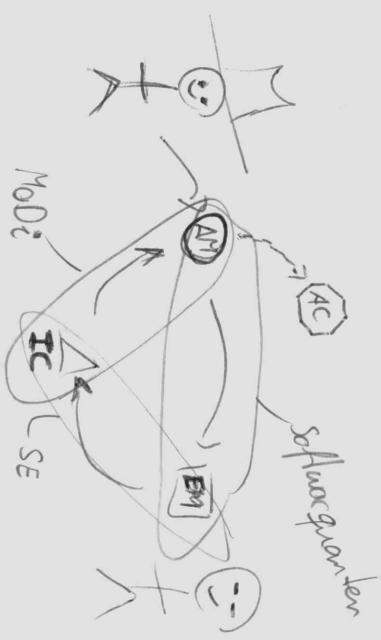
14/45



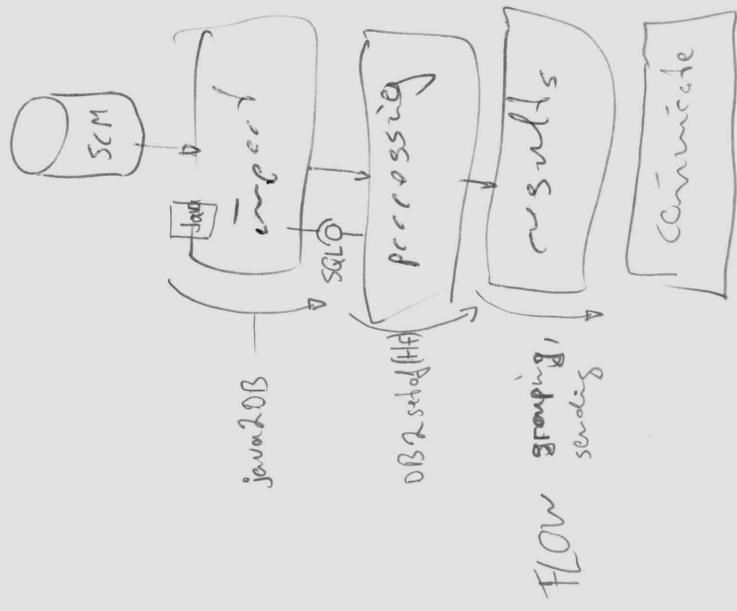


$$\begin{aligned}
 \text{AC} &\equiv \text{AC} ? \text{ or } \text{report} \\
 \text{AC} &= \text{UML} \quad \text{AC} \neq \text{Java} \\
 \text{AM} &\rightarrow \text{UML} \\
 \text{AC} \rightarrow \text{Java} & \\
 \text{EC} \rightarrow \text{Java} & \\
 \text{AC} + \text{EC} &= \text{IC}
 \end{aligned}$$

$\text{IR}(\text{AC}) = \text{PM}$



AM: Automatisches Modell
EM: Einzelhandel Modell
AC: Architektonische Code
EC: Einrichtungs Code
RM: Referenz Modell
IM: Implementations Modell
IR: Interne Repräsentation



- 5. Ein Nutzer erstellt eine Regel
- 6. Ein Nutzer ändert eine Regel
- 7. Ein Nutzer löscht eine Regel

AM
IM
↗ Regel :: (ist == soll)? ✓ : add Report Entry

Es scheint sich also eine drei-schichten Architektur zu ergeben:

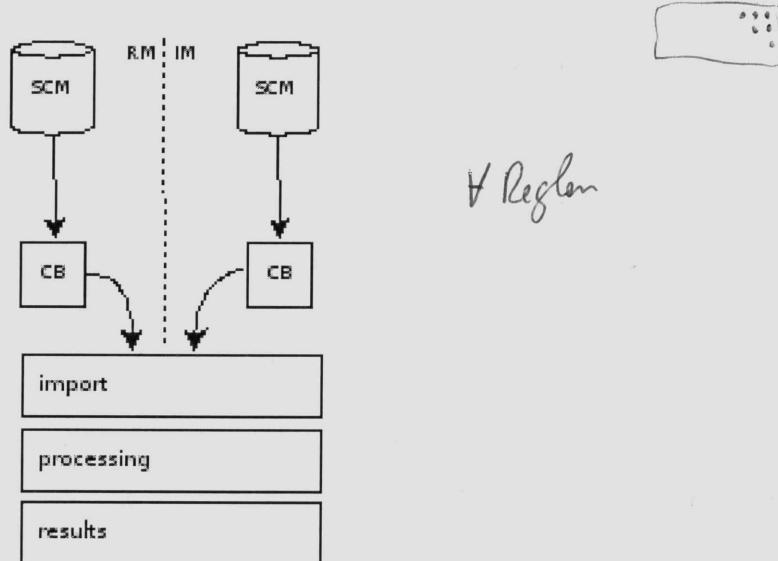
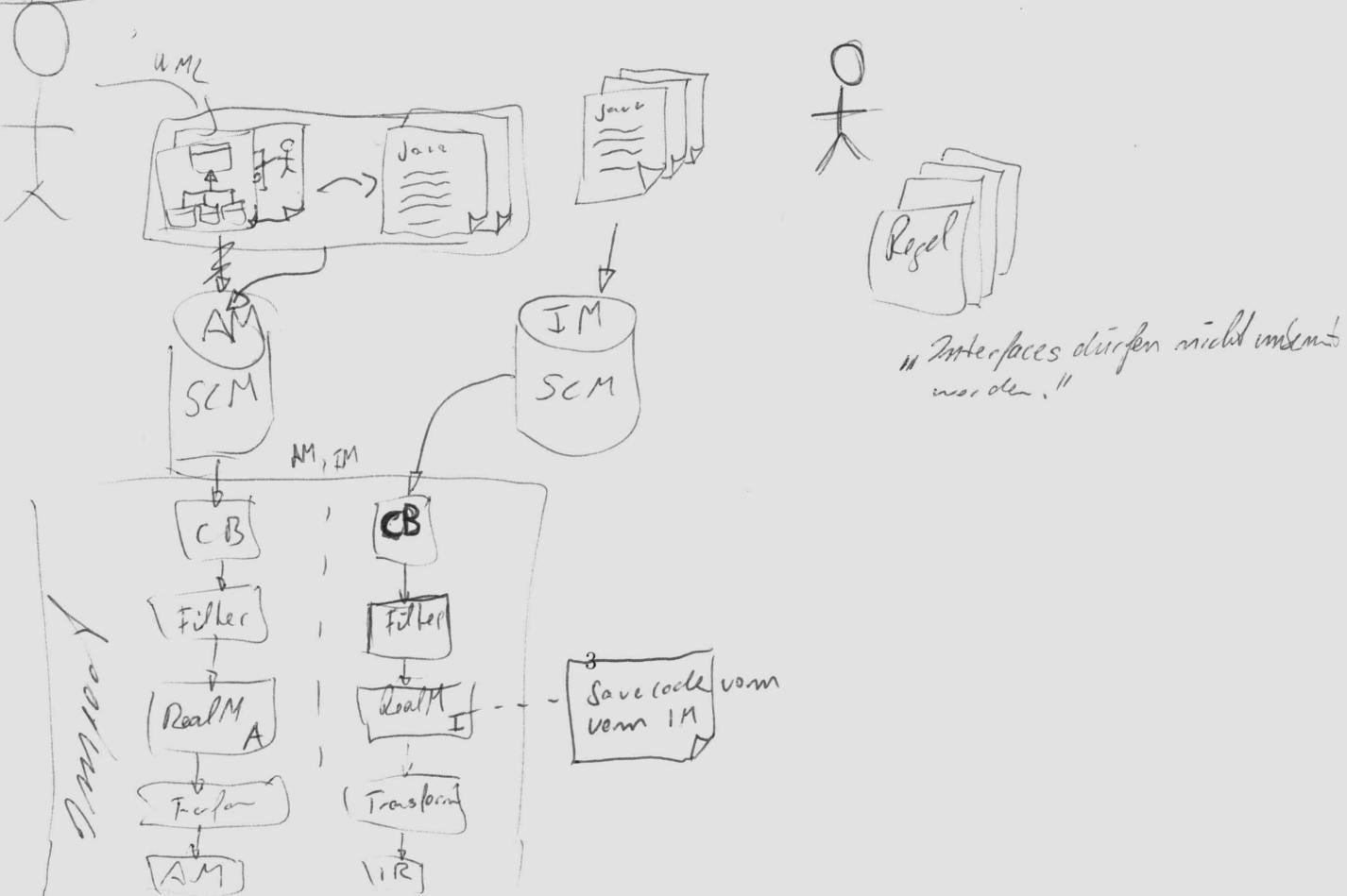
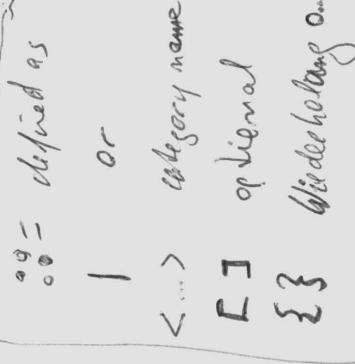


Abbildung 1: Grobarchitektur

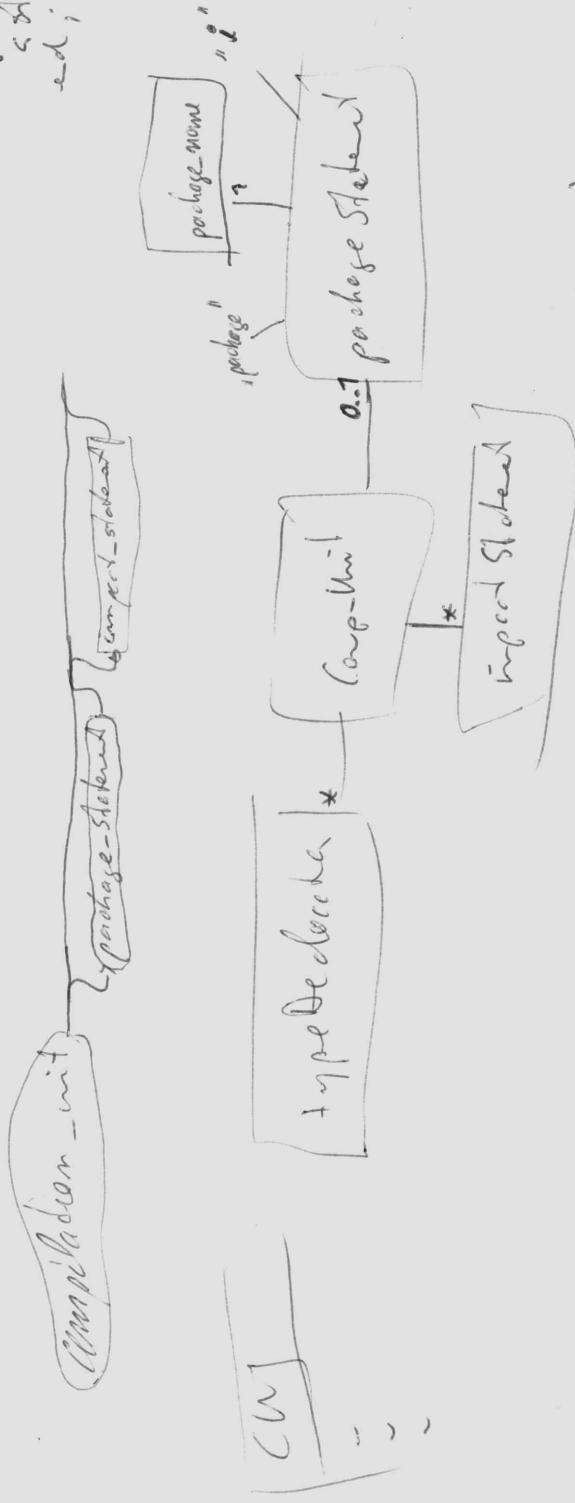


[BNF]



Program ::= Program
 < declarations>
 begin
 < statements>
 end;

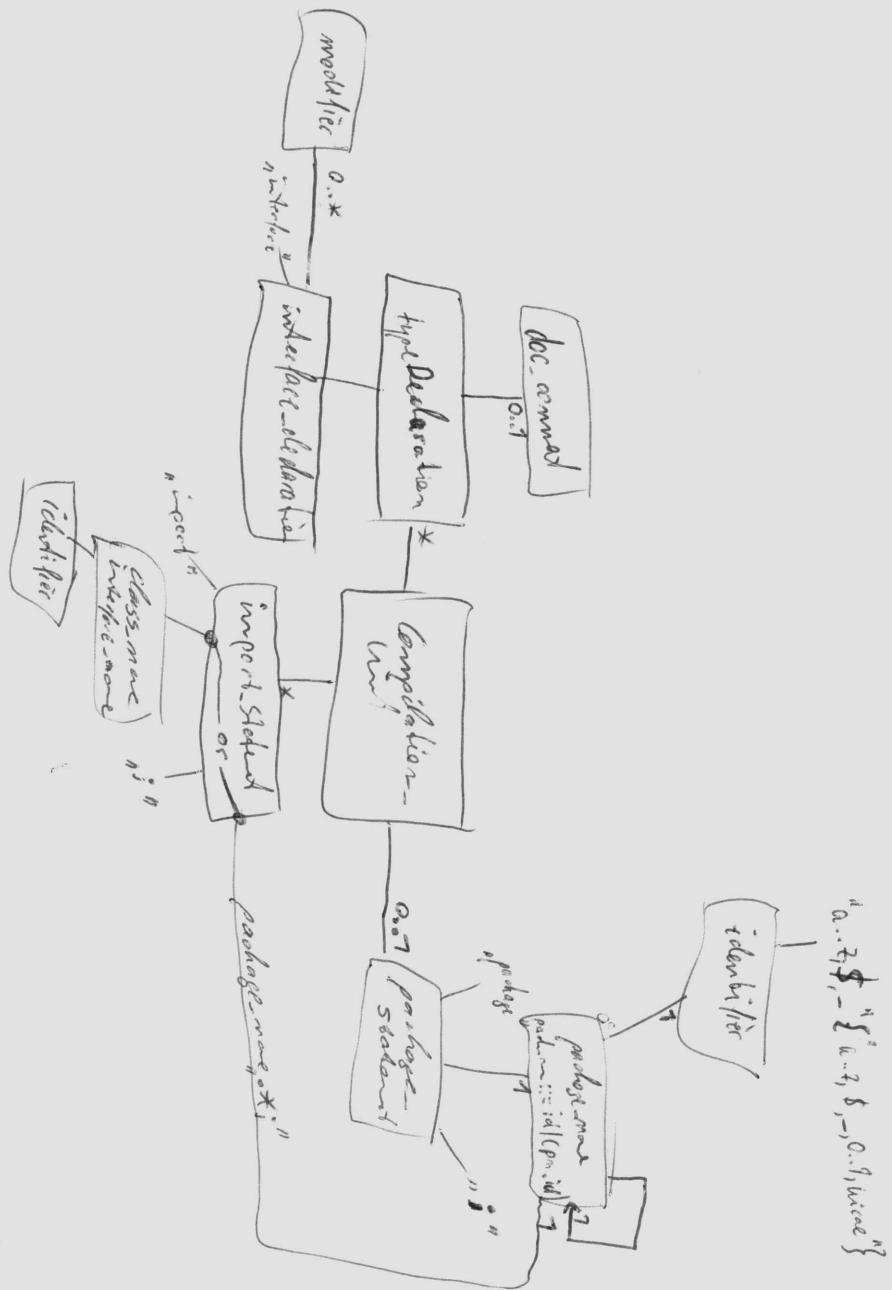
19.5

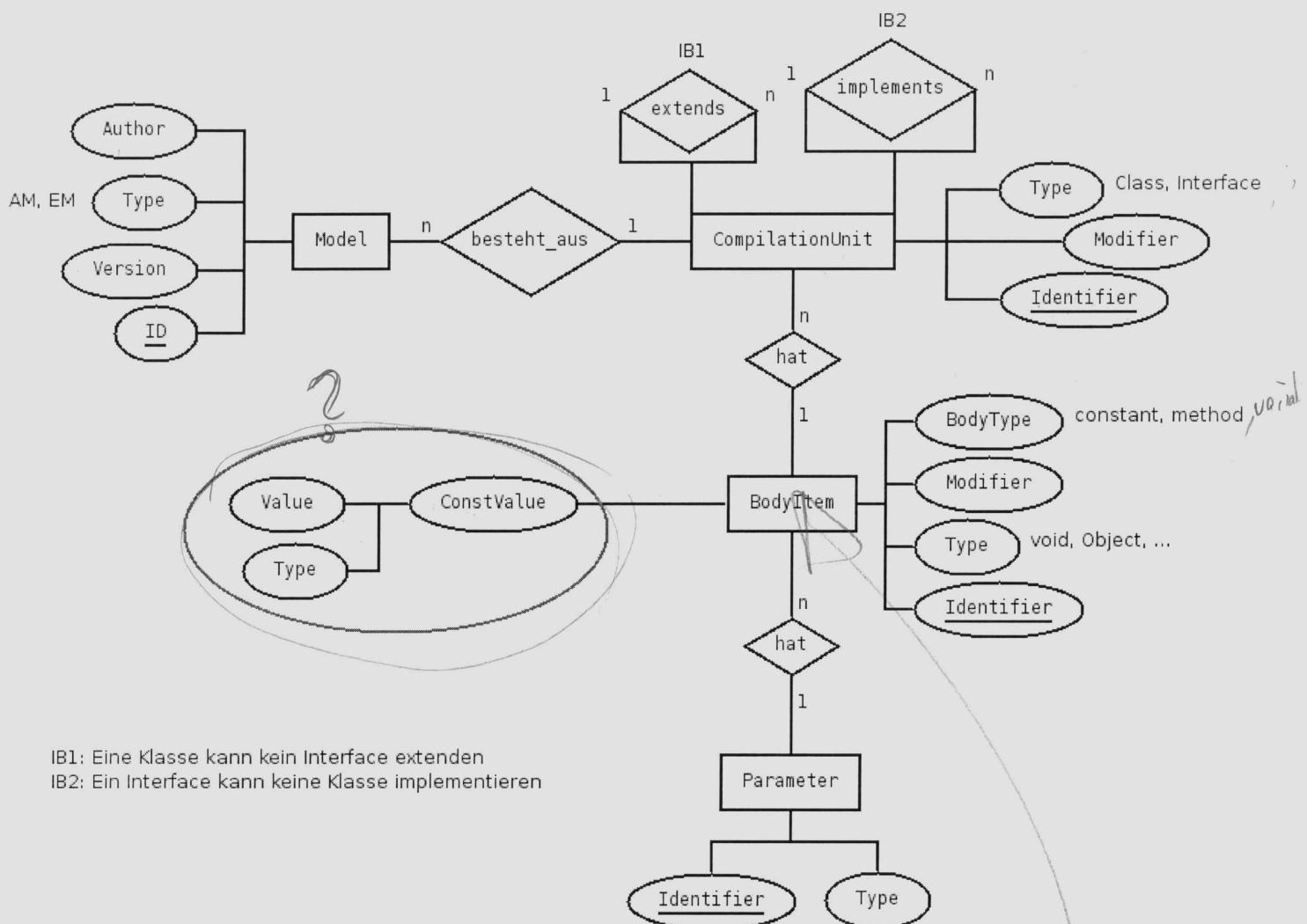


in top-level ::= identifier | package-name | "import"

class-name ::= identifier | package-name | "import"

jump main�main /loc /java BNF/..



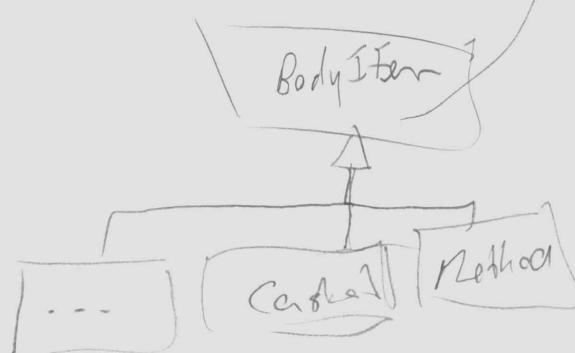


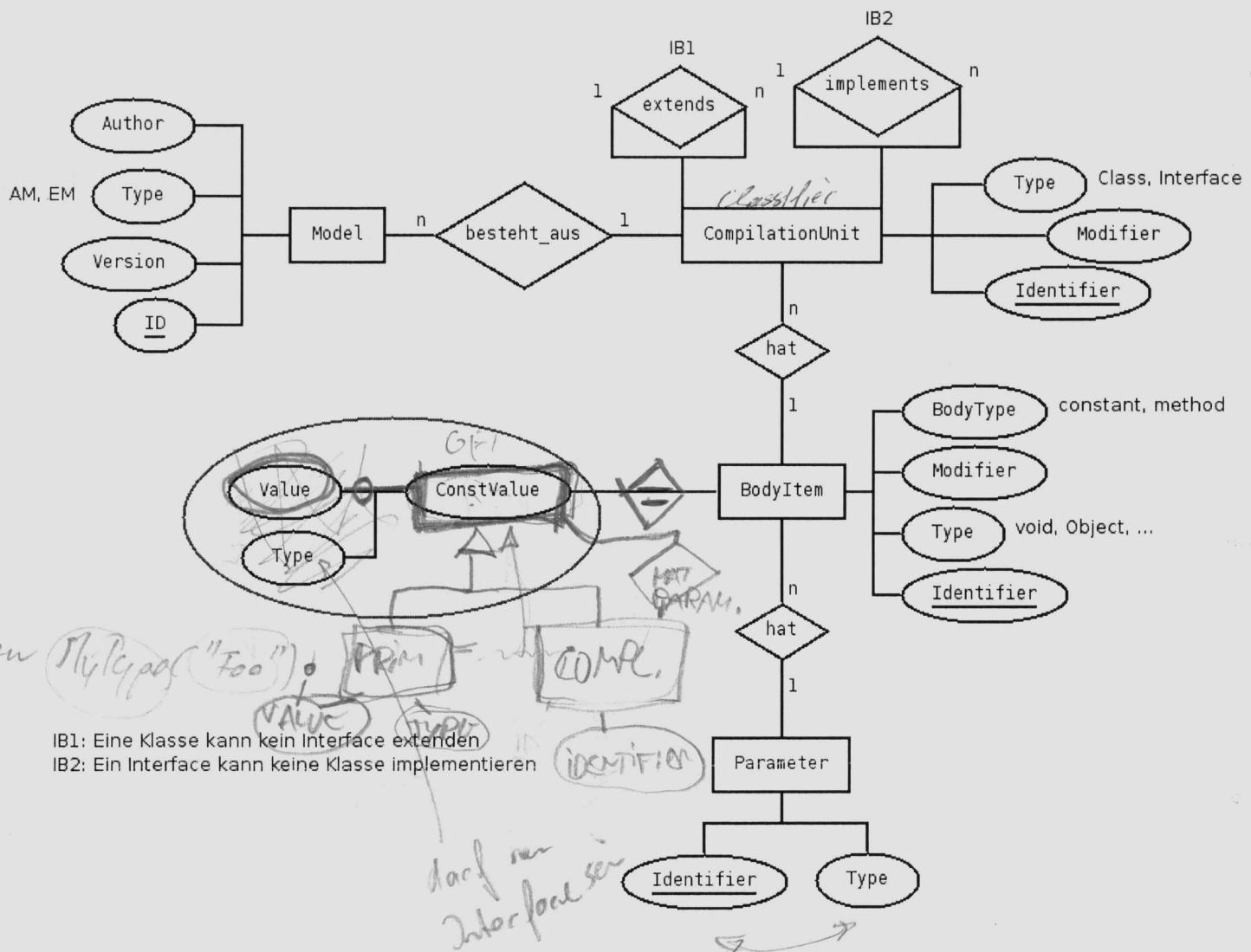
package names? \Rightarrow Full qualified class names

inner classes?
anonymous

... grandparent

de.dllc.MyClass
de.dllc.MyOtherClass\$



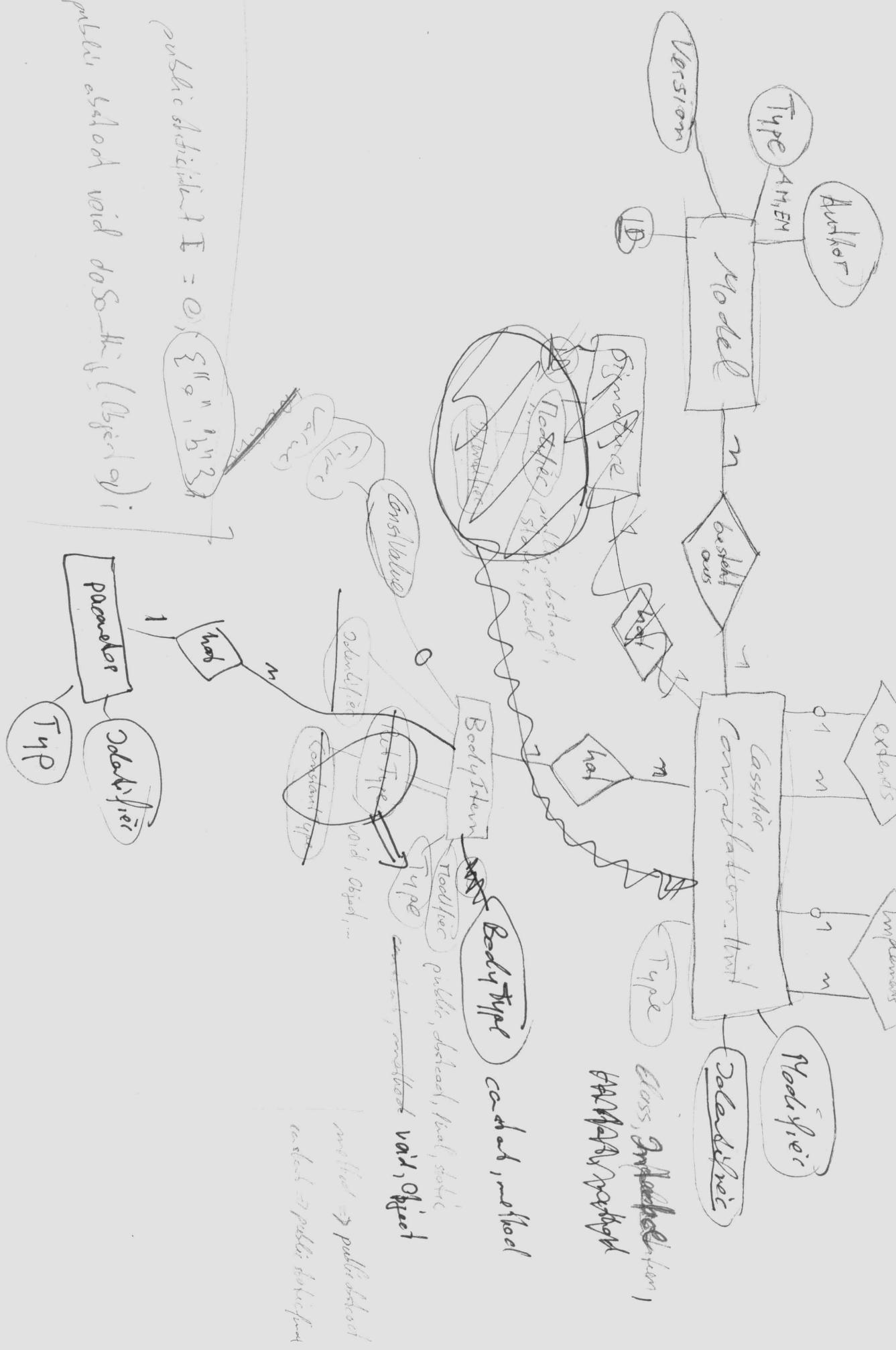


int INT = 0;
public static final Object O = new MyType(0, "foo", MyObject);

IC. 1.07 v2

ER2 =

ER1 =



1. erTabell

public class

variables
constants
methods

functions
methods

[obj... opt.-] E

public constants
variables

public methods

value

public
method

Object!

Parameter

3

2. make Tabelle

variable

visibility

name

type [Text. int.]

visibility

type

wave

number

counts

wave

last

type

testType

visibility

name

type

value

function

method

type

value

function

method

type

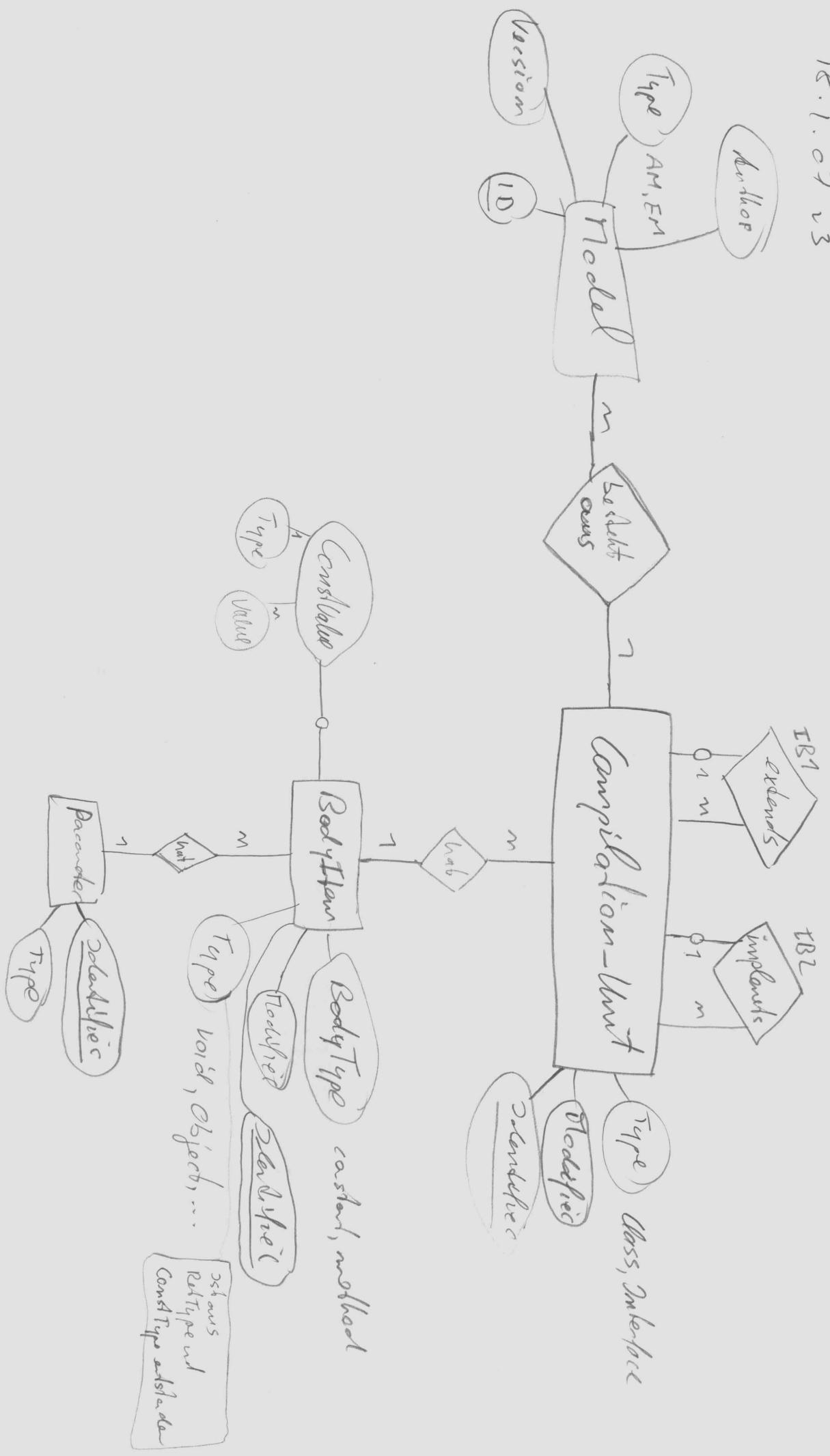
value

Spec.
LKW

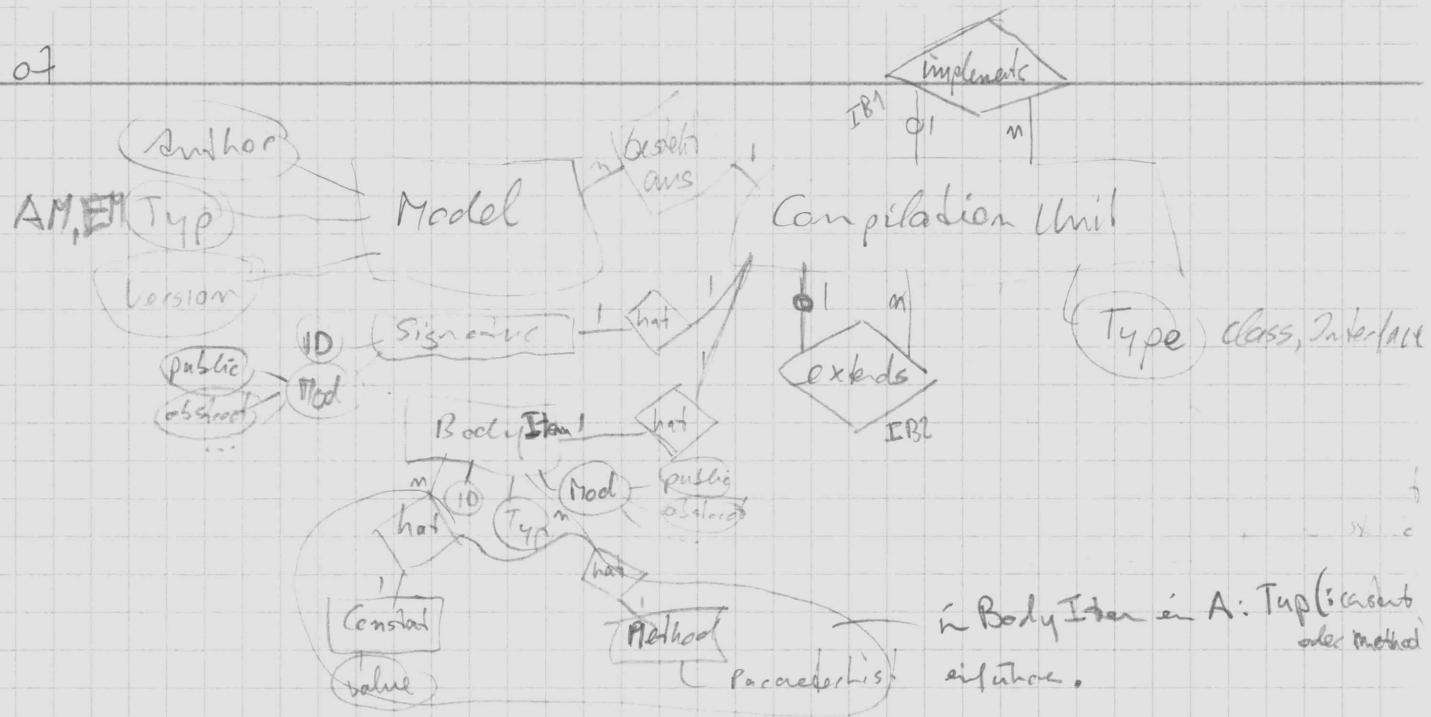
height
position

1
1
1

18.1.07 23

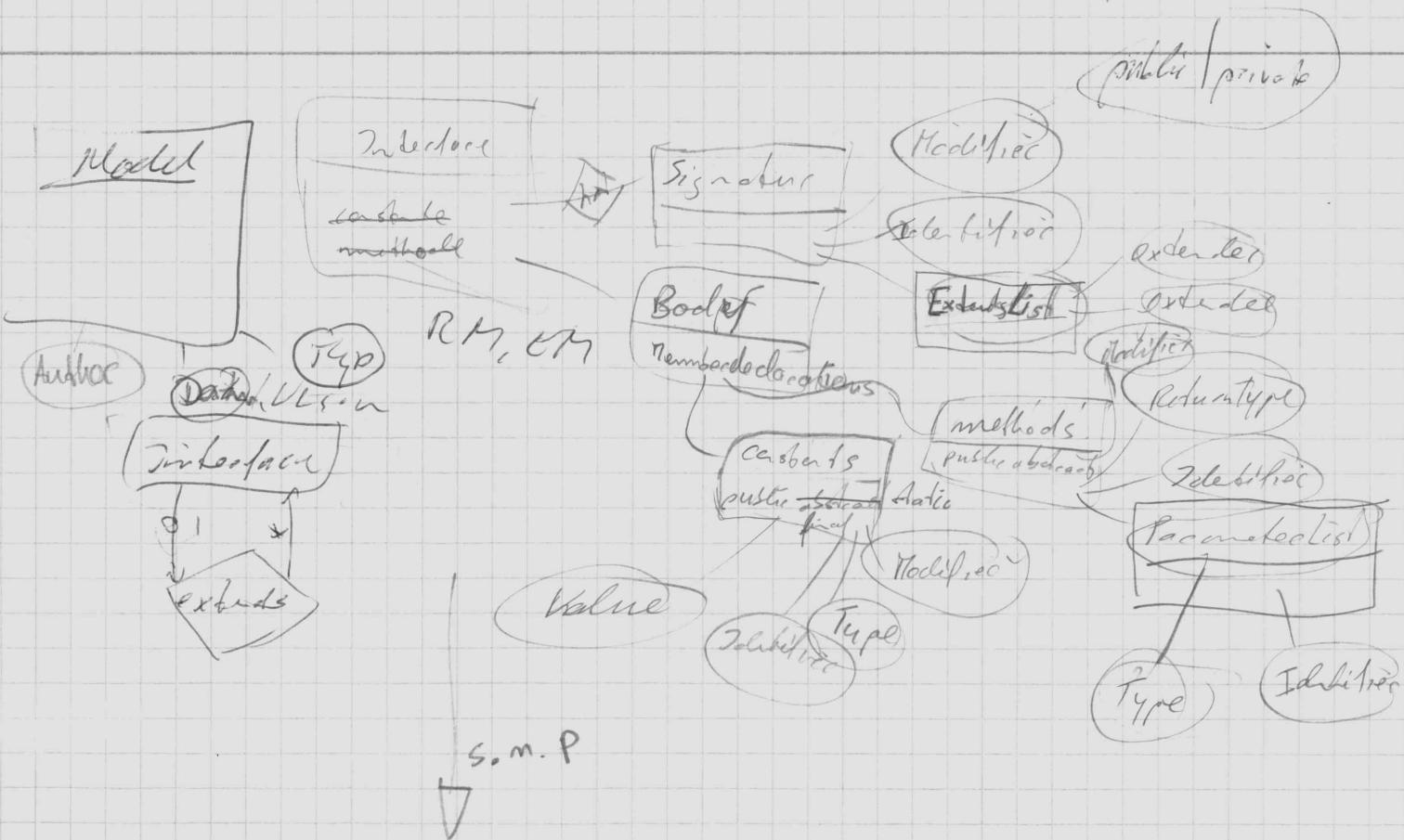


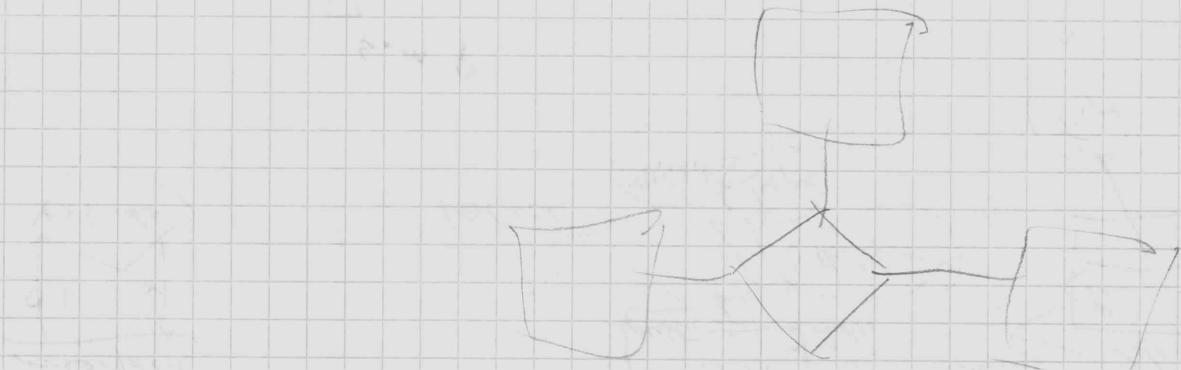
17.1.07

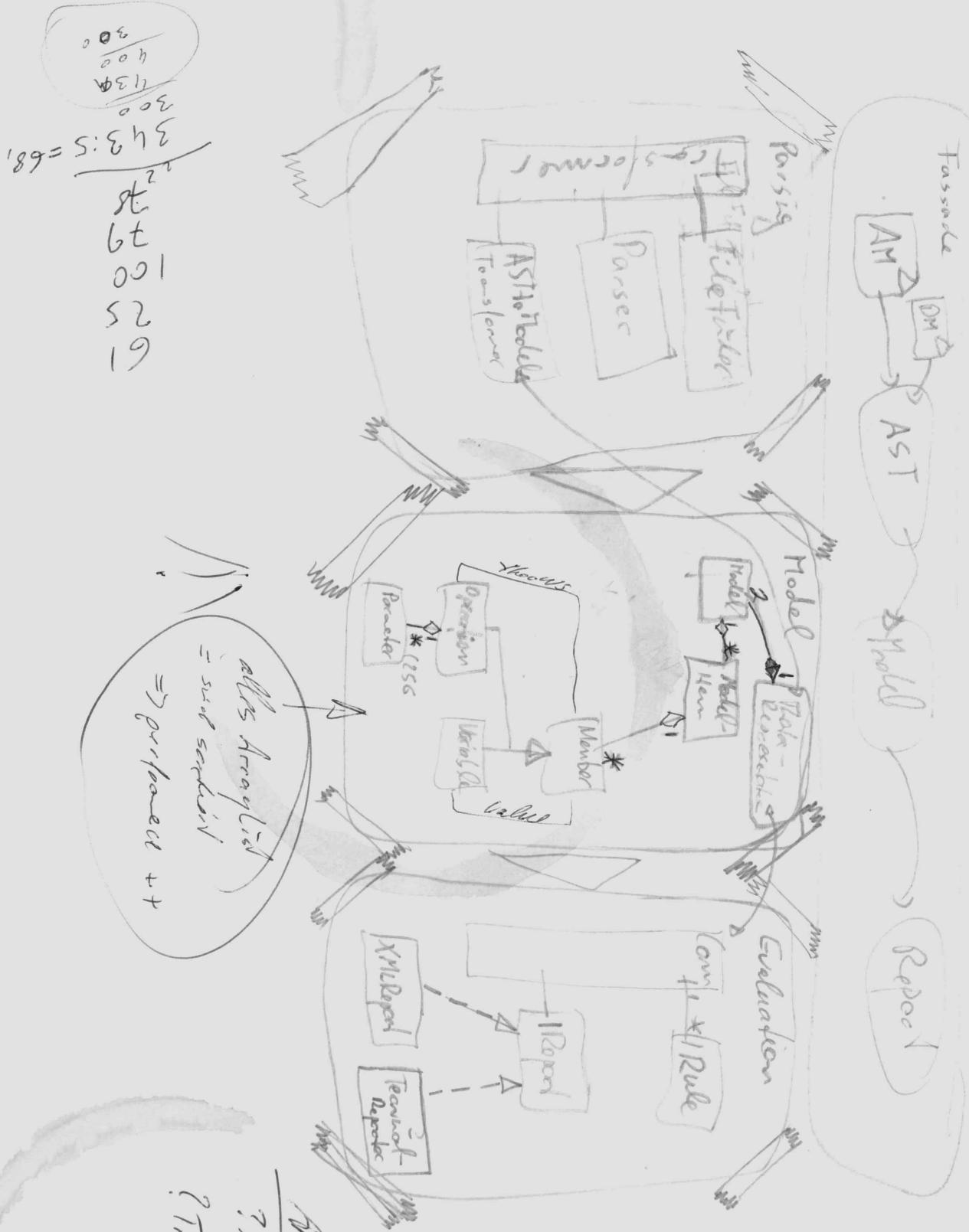


IB1: CU-Typ = class, wenn implements \Rightarrow Es in Interface bei Klasse implementiert

IB2: Eine Klasse kann kein Interface erweitern







Stk.