12. Basic Techniques of SE, Language Families, and Composition of Tools (Structure of M2)

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de
Version 13-0.5, 01.11.13

1) Überblick
2) Datendefinitionssprachen (DDL)
   1) ERD, XSD
3) Datenanfragesprachen (DQL)
   1) Semmle .QL
   2) Xquery
4) Datenkonsistenzsprachen (DCL)
5) Datentransformation (DTL)
   1) Xcerpt see separate chapter
6) Datenmanipulationssprachen (DML) und Verhaltensspezifikationssprachen (BSL)
   1) Datenflussdiagramme
   2) Pseudocode
7) Benutzungshierarchie der Sprachfamilien

Obligatorische Literatur

- http://en.wikipedia.org/wiki/List_of_UML_tools
- http://en.wikipedia.org/wiki/Entity-relationship_model
- http://www.utexas.edu/its/archive/windows/database/datamodeling/index.html
- Sebastian Schaffert, François Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt (2004). In Proc. Extreme Markup Languages.

http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-7.pdf
http://www.rewerse.net/publications/download/REWERSE-RP-2006-069.pdf

Andere Literatur

- Informatik Forum http://www.infforum.de/
- De Marco, T.: Structured Analysis and System Specification; Yourdon Inc. 1978/1979. Siehe auch Vorlesung ST-2
- McMenamin, S., Palmer, J.: Strukturierte Systemanalyse; Hanser Verlag 1988

- ARIS tool (IDS Scheer, now Software AG)
  - http://en.wikipedia.org/wiki/Architecture_of_Integrated_Information_Systems
- MID Innovator (insbesondere für Informationssysteme)
  - http://www.modellerfolg.de/

Ziel

- Lerne die verschiedenen Sprachfamilien kennen, und damit die Struktur von M2 der Metahierarchie
- .. und wie sie zur Beschreibung von Basistechniken in Werkzeugen und Werkzeugaktivitäten eingesetzt werden können
- .. und wie sie zur Komposition von Werkzeugen eingesetzt werden können

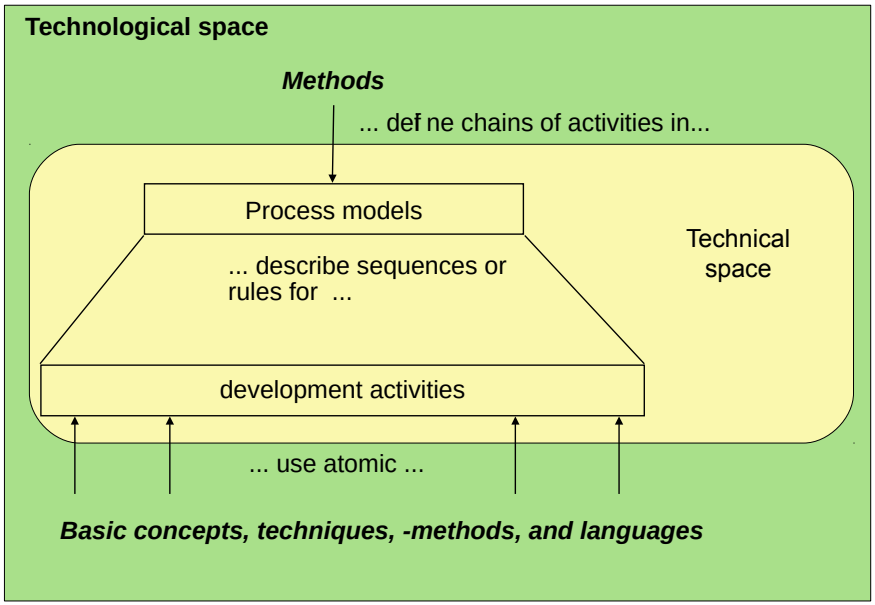
12.1 Basic Techniques of Software Engineering, Language Families, and Tool Composition

Begriffserläuterung

- **Vorgehensweise (Vorgehensmodell)**: Vorgehensweisen enthalten den Weg zu etwas hin, d.h. sie machen Softwareentwicklungsmethoden anwendbar.
- **Prozess:** Eine automatisiert ausführbare, geführte Vorgehensweise
- **Aktivitäten**: Eine Aktivität ist die konkrete Durchführung von definierten Aktionen innerhalb eines Software-Entwicklungsprozesses.
- **Basistechniken**: unterstützen Aktivitäten im Entwicklungsprozess, die gekapselt in unterschiedlichen Methoden angewandt werden.
- Basistechniken besitzen eine **(Basis-)Sprache** mit Notation (Syntax) und Semantik
- Basissprachen bilden konkrete Formen von **Basiskonzepten**, d.h. abstrakten Sprachen

Quellen: [3, S. 36], [31, S. 81], [24, S. 41], Arbeitskreis GI-Fachgruppe 5.11 „Begriffe und Konzepte der Vorgehensmodellierung“; http://www.tfh-berlin.de/~giak/arbeitskreise/softwaretechnik/themenbereiche/grundbgr.html

Basic Techniques of SE, Process Models, and Development Activities

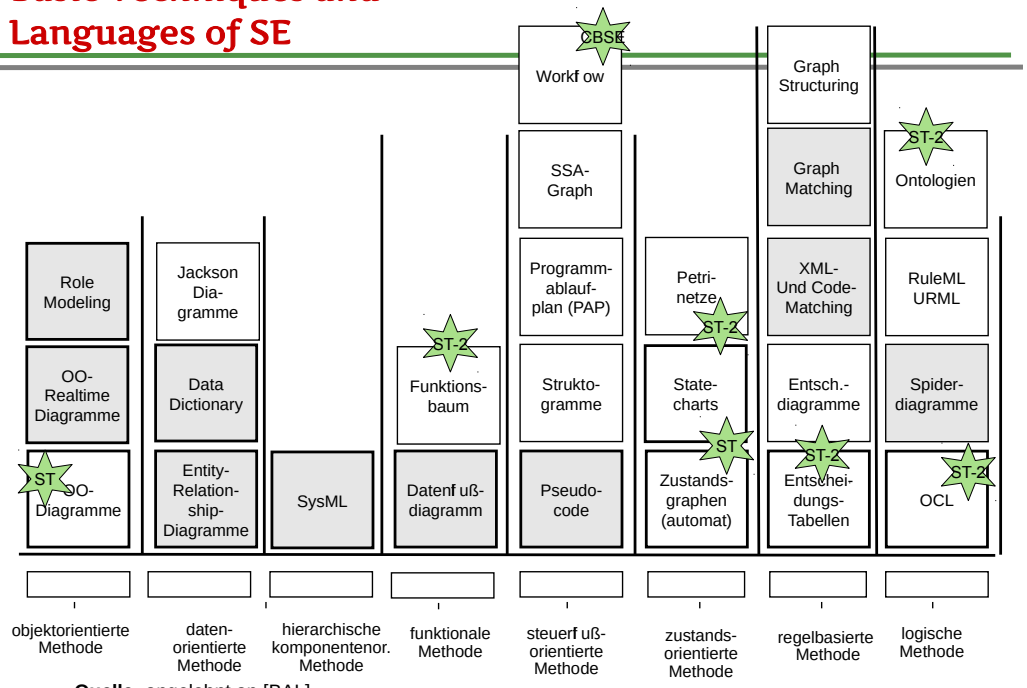


[Raasch]

Method Engineering (Process Engineering)

Process Engineering (Method Engineering) is the discipline of constructing and running processes for a team of people to conduct a project.
Software Process Engineering (Software Method Engineering) focuses on software development processes.

Basic Techniques and Languages of SE



Quelle: angelehnt an [BAL]

Building Software Tools for Basic Techniques is Expensive

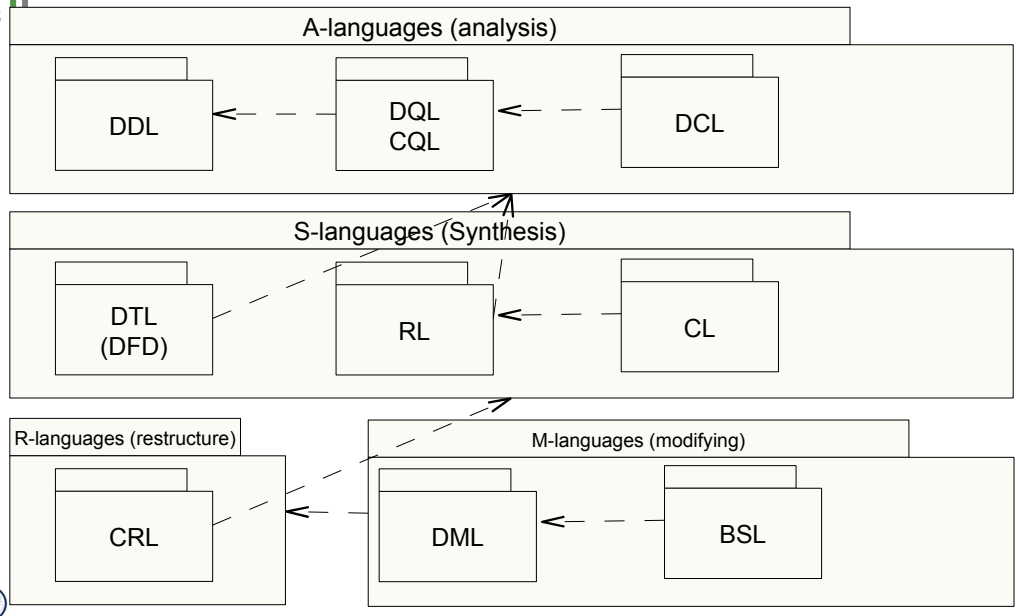
Werkzeug	Person years	Cost in kEuro
Compiler	1-2	100
Optimizer	1-3	150
Back-End	0.5-1	100
Compiler component framework	20	1000
UML-Werkzeug	5	250
Java-Refactorer	2-4	200
Energy Unit Test-Framework	1	50
Werkzeug zum Anforderungsmanagement	2-4	200
Mobile Phone Test-Framework	2	100

Questions

How can I reuse tools for more complex tools, to support several basic techniques? (tool composition)
How can I compose tools in an IDE?

By composing languages systematically

Basic Language Families (Structure of M2)



Basic Language Families (Structure of M2)

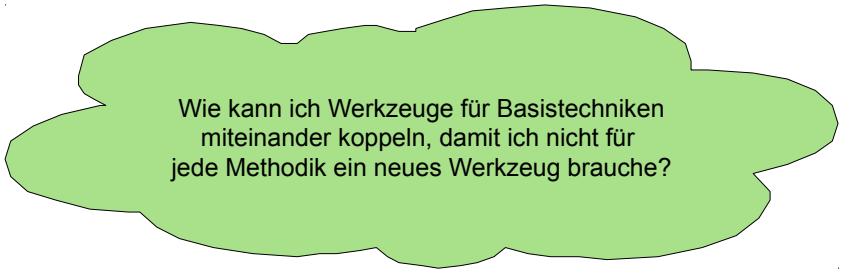
- Datenmodellierung mit **Datendefinitionssprachen** (data definition languages, DDL)
  - Werden zur Definition von Daten (Repositories, Strömen, Dateien) genutzt
  - DDL bilden die Basispakete von M2, die von allen anderen Pakete importiert werden (MOF → UML-CD → UML-Statecharts)
  - EBNF-Grammatiken, Relationales Modell (RM), Entity-Relationship-Modell (ER), UML-Klassendiagramme, SysML-Komponentendiagramme
- **Analyse-Sprachen** (A-Sprachen):
  - Daten-Abfrage mit **Abfragesprachen** (data query languages, DQL)
    - Code-Abfragen mit Code-Abfragesprachen (code query languages, CQL)
  - Sprachen zur **Daten-Konsistenzprüfung** (data constraint languages, DCL) und der Wohlgeformtheit der Daten
  - Logiksprachen
- **Daten-Synthesesprachen** (E-Sprachen)
  - **Datentransformationssprachen** (z.B. data flow diagrams, DFD)
    - Term- und Graph-Ersetzungssysteme
    - XML-Transformationssprachen

Basic Language Families (Structure of M2) (ctd.)

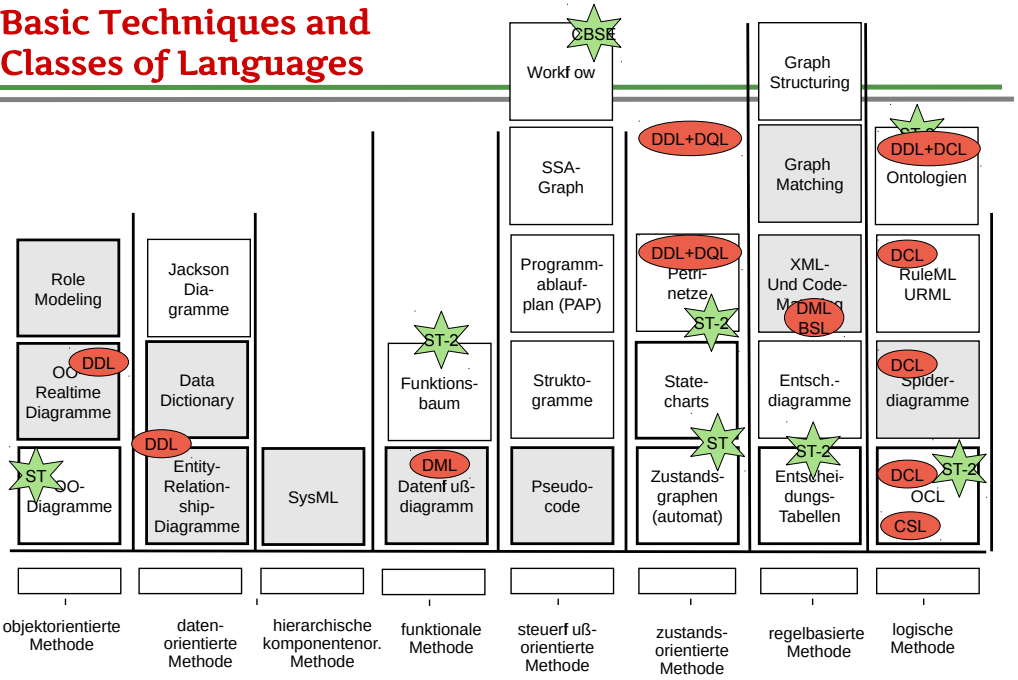
- **Wiederverwendungssprachen (reuse languages, RL)**
  - Vertragssprachen (contract specification languages, CSL)
  - Composition languages (CL), Architectural description languages (ADL)
  - Template-Sprachen (template languages, TL)
  - ==> course CBSE
- Daten-**Restrukturierungssprachen** (R-Sprachen, data restructuring languages, DRL)
  - **Datenaustauschsprachen** (data exchange languages)
  - **Data representation languages** (for representation change)
- Daten-**Manipulationssprachen** und **Verhaltensspezifikationssprachen** (M-Sprachen, data manipulation and transformation languages, DML)
  - Sprachen zur **Verhaltensspezifikation** (declarative behavior specification language, BSL) mit einer **formalen Semantik**
    - Aktionsbasiert, mit Zustandssystemen (Endliche Automaten und Transduktoren)
    - Datenflusssprachen, Workflowsprachen
    - Condition-Action-Sprachen (z.B. Entscheidungstabellen), Event-Condition-Action-Sprachen (ECA)
  - Imperative languages
  - Siehe auch Vorlesung ST-2, hier stehen daten-orientierte Sprachen im Vordergrund

Software Engineering vs Programming

- Eine Softwareentwicklungsmethode benutzt immer mehrere Basistechniken, d.h. nehrere Sprachen.
  - DDL, DQL, DCL, DRL, DML, TL, RL, CSL, DML, BSL
- Homogene Software-Konstruktion gibt es nicht!



Basic Techniques and Classes of Languages



Quelle: angelehnt an [BAL]

12.2 Data Definition Languages (DDL)

The basic layer of M2

Datenkataloge als Grundlage für Informationssysteme und Softwarewerkzeuge

- Ein **Datenkatalog (data dictionary)** enthält alle Modelle und Typen von Daten, die in einem System benutzt werden
  - Der Datenkatalog *typisiert* die Datenablage oder den Datenstrom
  - Datenkataloge können lokal zu einer Anwendung, zu mehreren oder zum ganzen Unternehmen und der Zuliefererkette bezogen sein
- Ein **homogener Datenkatalog** wird in *einer* DDL, ein **heterogener Datenkatalog** in mehreren DDL spezifiziert
  - EBNF definiert Stringsprachen, d.h. Mengen von Strings oder Typen
  - Relationales Model (RM) definiert Relationen und Tabellen
  - XML Schema (XSD) definiert Baumsprachen, d.h. Mengen von Baum-Typen
  - ERD oder UML-Klassendiagramm definieren Graph-Modelle
- Ein **Informationssystem** ist ein Softwaresystem, das Datenanalysen über einer **Datenablage** (einem **Repositorium**) durchführt.
  - Informationssysteme werden in den Datenbank-Vorlesungen gesondert betrachtet
  - Data warehouses, business intelligence, data analytics
- Ein **strombasiertes Informationssystem** ist ein Softwaresystem, das Datenanalysen über einem Datenstrom durchführt

Artefact Information Systems

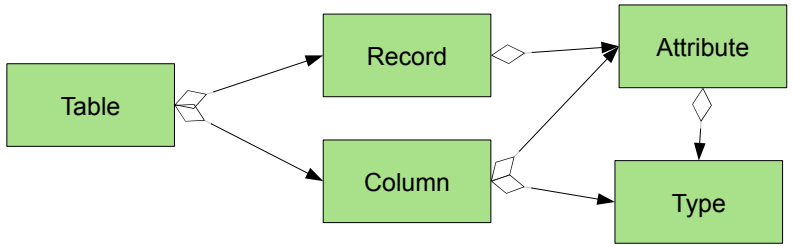
- Every software tool, every IDE relies on an artefact information system
  - maintaining artefacts (programs, models, documents)
  - giving information about them
  - typed by the types in a data dictionary
- The data dictionary is described in a data definition language
  - Metasprachen sind A-Sprachen (Analysesprachen); sie bestehen aus DDL und DCL
  - Selbstbeschreibende Metasprachen bestehen aus einem gelifteten Metamodell

Textual Data Dictionary in TS Grammarware: Syntax with Grammars in metalanguage EBNF

Symbol	Bedeutung	Beispiel
name	Bezeichner (Entitytyp, Bez.typ,Attr.)	A = B + C
"text"	prim. Wert (nicht mehr zerlegbar)	B = "W1" + R
=, ::=	besteht aus	X = X1 + X2 + X3
+	Sequenz, auch einfach Juxtaposition	X = X1 X2 X3
@	Schlüsselkennzeichen	P = @Pnr + N + Adr
[...|...]	Selektion (entweder ... oder)	P = [P1 | P2]
n { ... } m	Iteration von n bis m	B = 1 { C } 10
(...)	Option (kann vorhanden sein)	A = B + (C)
A // “,”	Liste von A mit innenliegendem ','	C = D // “,”
* ... *	Kommentar	X = B + C *Kommentar*
< a > b	Modif er (Kommentar)	< alt > A < neu > A
SYN	Synonym für Name	K SYN P

Technikraum Relationale Algebra mit Metasprache Relationales Schema

- Die Relationale Algebra (Codd) wird hier als bekannt vorausgesetzt
  - Ihr Schema bilden Tabellen mit Tupeln aus Attributen
  - Siehe Datenbank-Vorlesungen



Relational Schema

Key	FirstName	Surname	Street	Town
@1	Uwe	Aßmann	Bakerstreet 5	New York
@2	Frank	Miller	Northstreet 9	Pittsburgh
@3	Mary	Baker	Magdalenstreet	Oxford

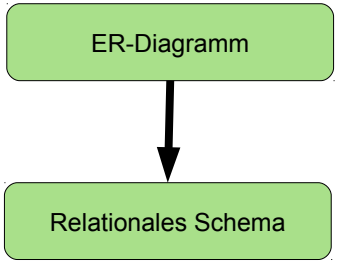
12.2.1 Technikraum Graphware mit Beispiel-DDL Entity-Relationship-Diagramme (ERD)

Eine einfache DDL mit Abbildung auf die Relationale Algebra

Relationen + Entitäten (ohne Vererbung)

Vorteile der Entity-Relationship-Modellierung

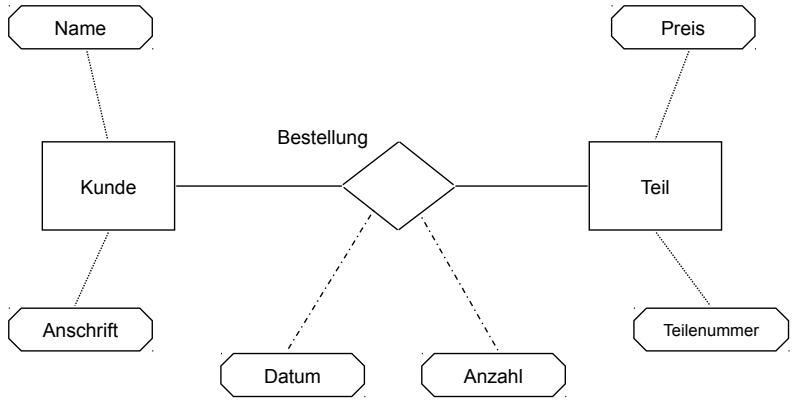
- Vorteil: Sehr leicht abbildbar auf Relationale Algebra (mit 1:n-Abbildung, ER-R-Mapping)
  - Entitäten bilden spezielle Relationen mit “Identifikator” (Schlüssel, surrogate)
  - ER-Diagramme sind daher sehr einfach in Datenbanken ablegbar



ERD-Modellnotation nach CHEN

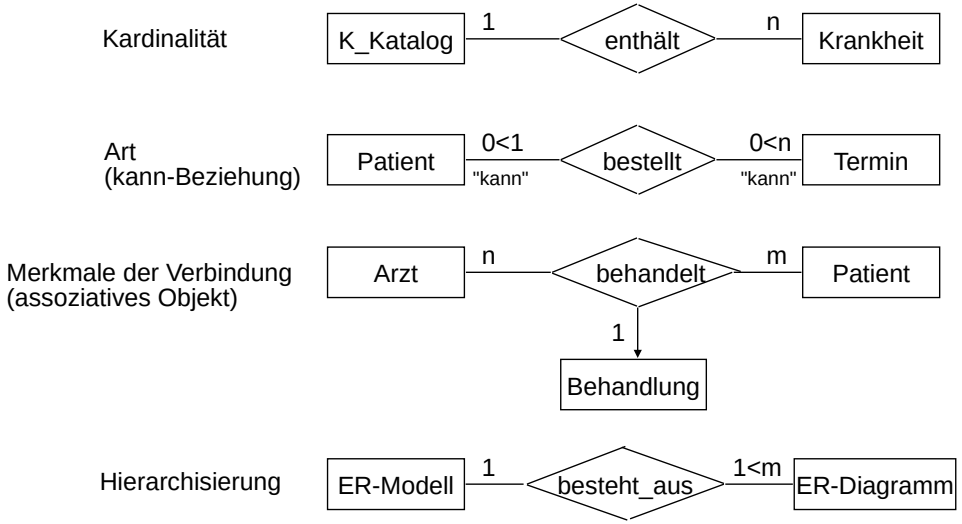
graph. Notation	Bedeutung
name	**Entitytyp:** Abstraktion einer Menge gleichartiger Datenobjekte beschrieben durch (mehrere) Attribute. Jedem Datenobjekt sind eindeutig Attributwerte zugeordnet.
name	**Beziehungstyp:** Menge von Beziehungen zwischen Entitytypen, beschrieben durch verknüpfte Aufzählung identif zierender Schlüssel der Entitytypen.
Name (selten dargestellt)	**Attribut:** Beschreibende Eigenschaften von Entitytypen.Def niert durch Menge zulässiger Attributwerte.
1, n 0 < n	**Kardinalität:** Ganze Zahlen an den Verbindungs-linien, die angeben, wieviele Instanzen des anderen Entitytyps mit einer Instanz dieses Entitytyps in Verbindung stehen.

Ein einfaches ER-Modell

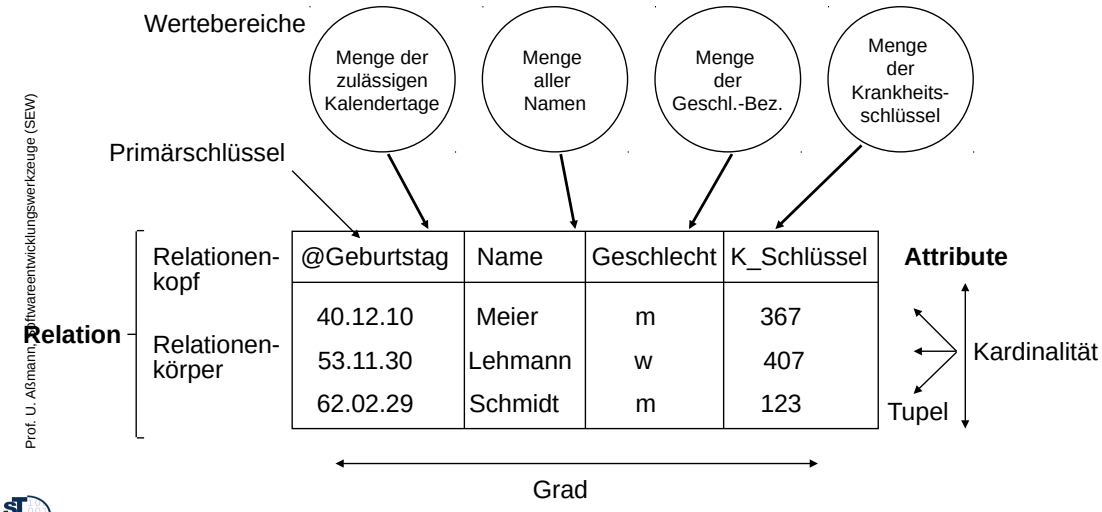


ERD-Beispiele in CHEN-Notation

Eigenschaften der Beziehungstypen:

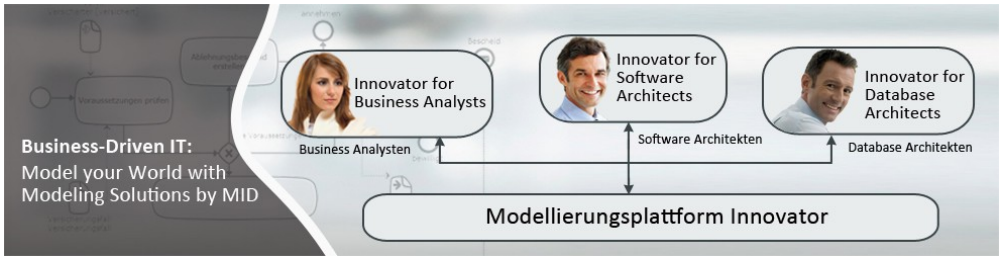


Beispiel des Entitytyps "Patient" und seine Abbildung auf das Relationenmodell



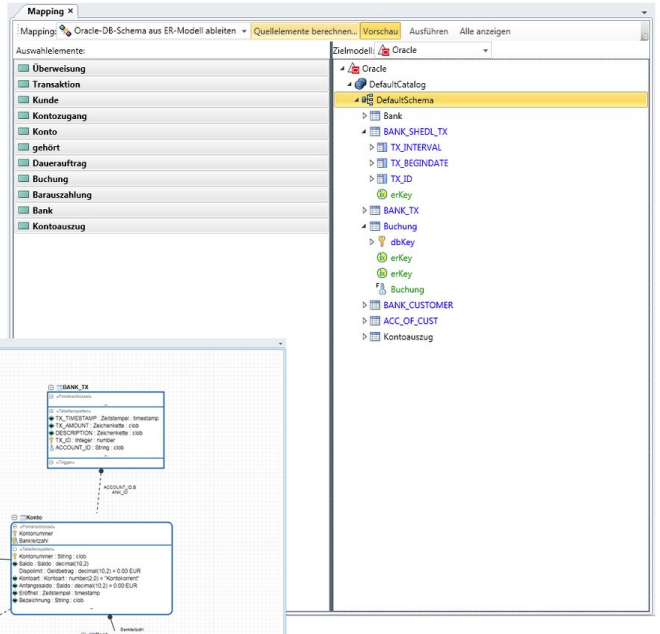
Wichtigkeit von ERD

- ERD ist sehr einfach (1:1) auf das Relationenmodell abbildbar
  - Eigentlich das “bessere” Relationenmodell.
  - ERD-Anwendungen sind einfach mit Persistenz auszustatten
- ERD besitzt keine Vererbung bzw. Polymorphie
  - Einfacher
  - Leichter verifizierbar, z.B. beim Einsatz für sicherheitskritische Systeme
- Typisches Werkzeug: MID Innovator für Datenbankarchitekten:



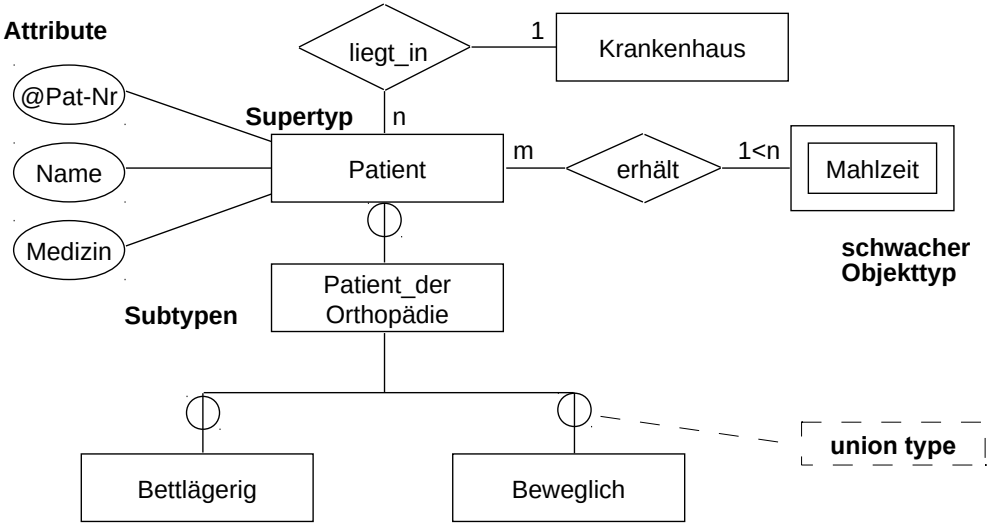
http://www.mid.de/index.php?id=541
http://www.mid.de/uploads/pics/Banner_Modellierungsplattform_03.jpg

Mapping ERD to RS in MID

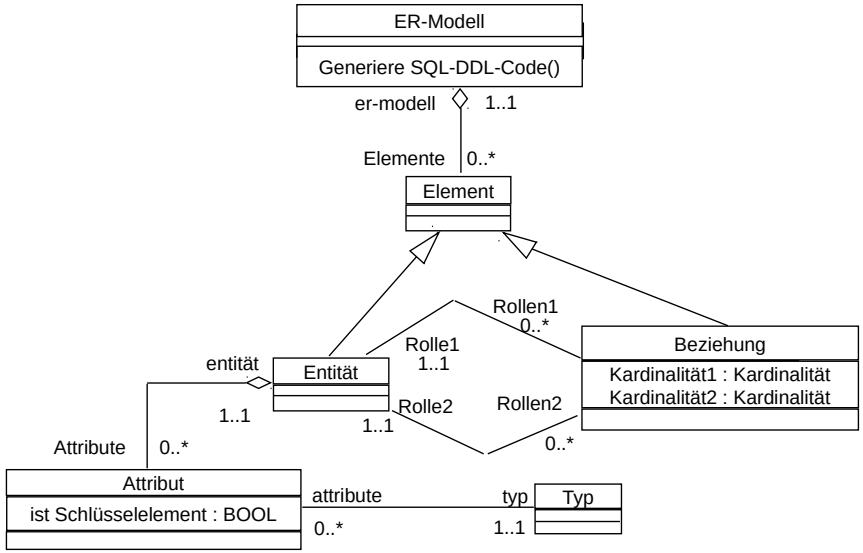


http://www.mid.de/typo3temp/pics/f0df65b8a2.jpg

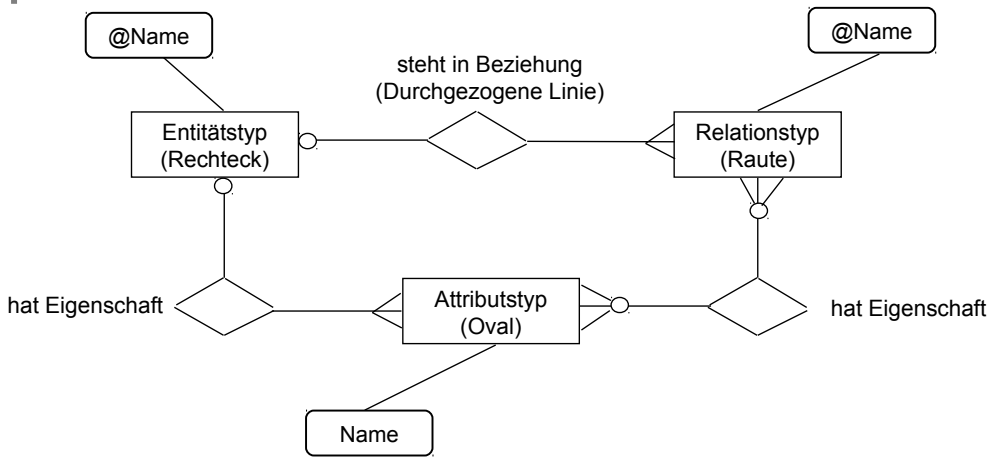
Beispiel für erweiterte ERD: Patientenakte



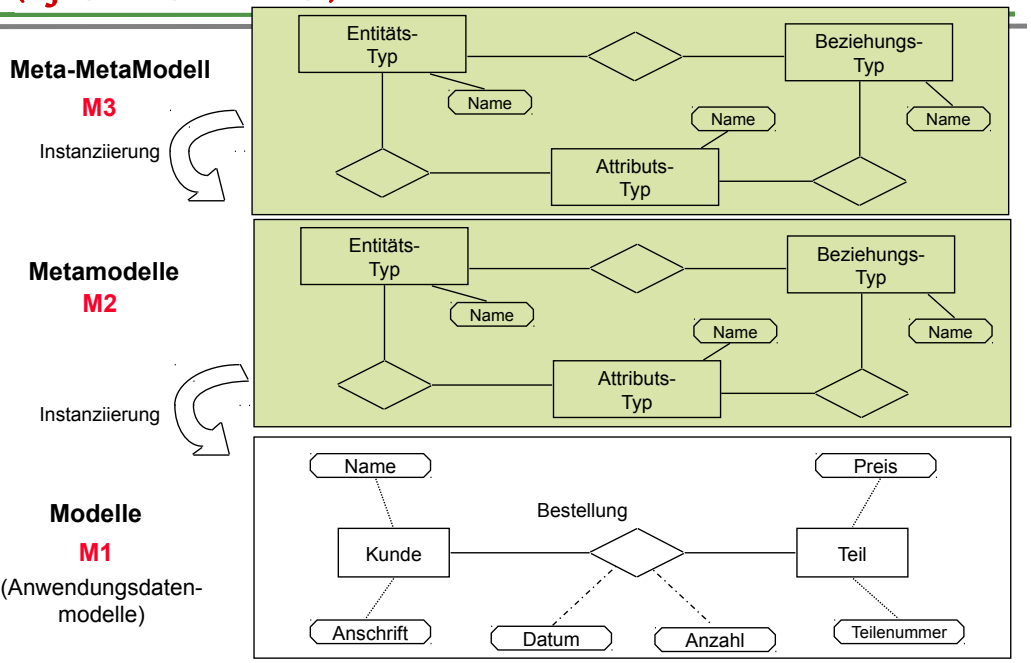
Meta-Modell von EntityRelationship-Diagrammen (in MOF)



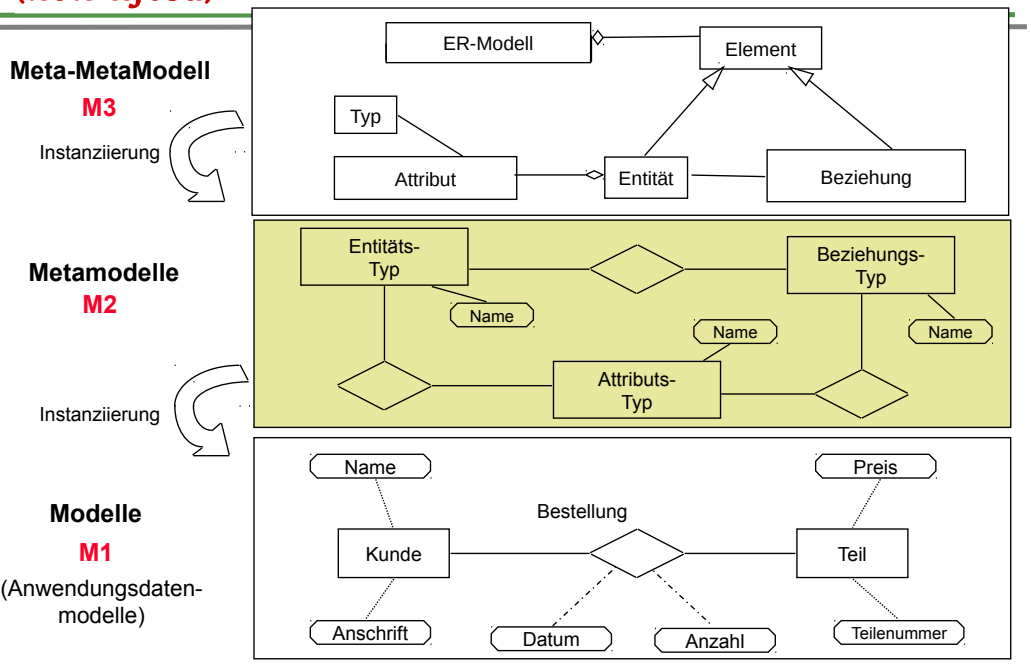
Das Metamodel von ER in ER



Metahierarchie mit ER als Metasprache (lifted metamodel)



Metahierarchie mit MOF als Metasprache (non-lifted)



Well-formedness of Models (Wohlgeformtheit von Modellen)

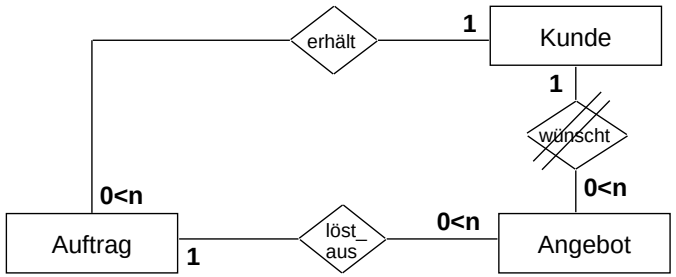
A model is **well-formed (consistent)**, if it fulfils the context-sensitive constraints (integrity rules, consistency rules) of its metamodel.

Die Überprüfung kann durch **semantische Analyse (Kontextanalyse)** erfolgen:

- Namensanalyse ermittelt die Bedeutung eines Namens
- Typanalyse ermittelt die Typen
- Typcheck prüft die Verwendung von Typen gegen ihre Definitionen
- Bereichsprüfungen (range checks) prüfen auf Gültigkeit von Wertebereichen
- Strukturierung von Datenstrukturen (Vorl. ST-II)
  - Azyklizität, Schichtbarkeit (layering), Zusammenhangskomponenten
- Verbotene Kombinationen von Daten

Bsp.: Analyse auf strukturierte, hierarchische, zyklenfreie Darstellung

- Integritätsbedingungung (Konsistenzbedingung): Zyklenfreiheit
  - Check: Auffinden von Zyklen (graphentheor. Problem)
  - Korrektur: Auftrennen von Zyklen an der "am wenigsten relevanten Stelle":



Quelle: [Raasch]

Konsistenzprüfung von ER-Modellen durch Werkzeuge

Datenmodelle in ER, MOF oder UML-CD, können auf folgende Integritätsregeln geprüft werden:

- **Bereichsprüfungen** für Wertebereich von Attributen (Typ, Range)
- **Ermittlung von Schlüsseln:**
  - **Eindeutigkeit** von Attributen: Ein (ggf. zusammengesetztes) Attribut K einer Relation R heißt **Schlüsselkandidat**, wenn zu keinem Zeitpunkt verschiedene Tupel von R denselben Wert für K haben
  - **Minimalität** eines Schlüssels: Ist Attribut K zusammengesetzt, kann keine Komponente von K entfernt werden, ohne die Eindeutigkeitsbedingung zu stören. Jedes Tupel einer Relation muß über einen **Primärschlüssel** eindeutig identif zierbar sein
    - Falls es weitere Schlüsselkandidaten gibt, werden sie als **Alternativschlüssel** bezeichnet.
- **Fremdschlüssel-Verbindung** (“indirekter Primärschlüssel”)
  - Ein **Fremdschlüssel** ist ein Attribut einer Relation R2, dessen Werte benutzt werden, um eine Verbindung zu einer Relation R1 über deren Primärschlüssel aufzubauen.
- **Referentielle Integrität**
  - Das Datenmodell darf keine ungebundenen Fremdschlüsselwerte enthalten

Praktische Vorgehensweise bei der Erstellung eines ERD

- Ähnlich wie strukturgetriebene Vorgehensweise in der ST-1-Vorlesung
- 1) Festlegen der Entitytypen
- 2) Ableitung der Beziehungstypen
- 3) Zuordnung der Attribute
  - zu den Entitytypen unter dem Gesichtspunkt der natürlichsten Zugehörigkeit, d. h. sie sind "angeborene" Eigenschaften unabhängig von ihrer Nutzung.
  - Kardinalitäten festlegen
- 4) Konsistenzprüfung
  - 5a) Fremdschlüssel definieren für die Herstellung notwendiger Verbindungen zwischen Entitytypen und Eintrag ins DD
  - 5b) Fremdschlüssel-Regeln spezifizieren, nach Rücksprache mit dem Anwender
- 5) Eintrag ins DD

Beispiel “Arztpraxis”

Aufgabenstellung:

“Es sind in einer **Arztpraxis** die organisatorischen Abläufe für das Bestellwesen der **Patienten**, den Aufruf aus dem Wartezimmer, die **Arztbehandlung** und die Abrechnung unter Einsatz von PCs weitgehend zu rationalisieren. Spätere Erweiterungen sollen leicht möglich sein.”

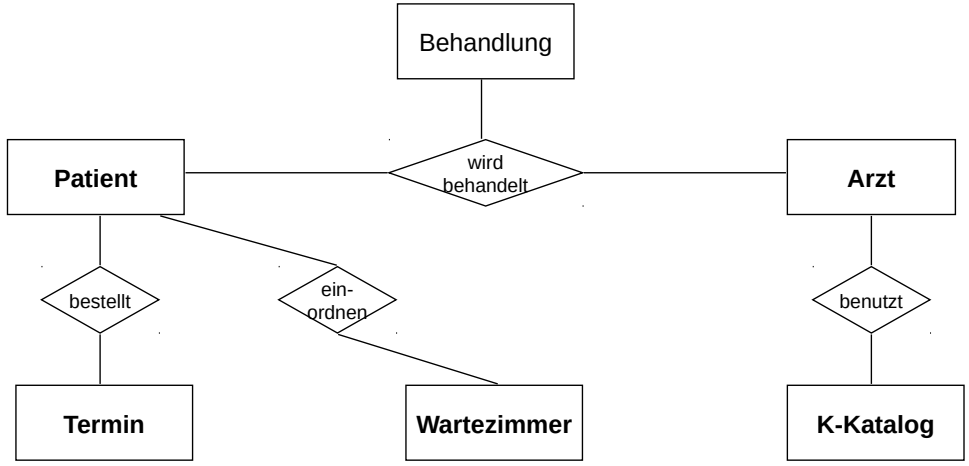
Analyse mit Verb-Substantiv-Analyse

ERD “Arztpraxis” (1)

Schritt (1)

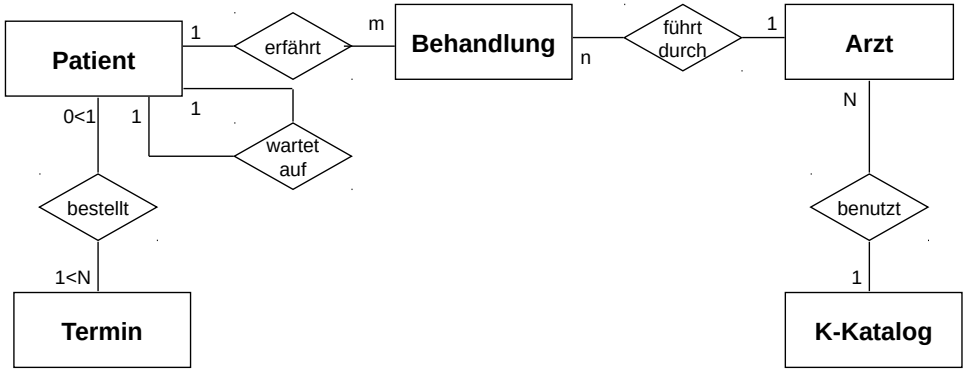
ERD “Arztpraxis” (2)

Schritt (2)



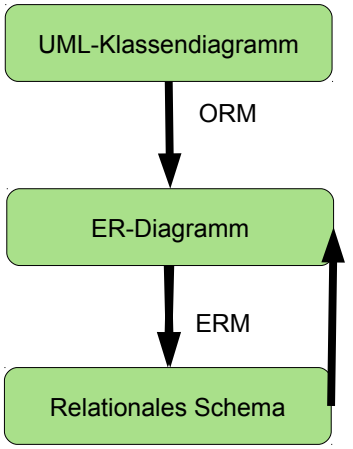
ERD “Arztpraxis” (3)

Schritt (4,5)



Datenmodellierung für Informationssysteme mit UML-CD, ERD und RS

- Objektrelationale Abbildung (OR-Mapping)
  - Auflösung der Vererbung durch Kopien der Oberklassenattribute oder durch Delegation
  - Ermittlung von Schlüsseln (Primär, Fremd)
  - Auflösung von Mehrfachvererbung durch Auffalten (Kopieren)
- Zwischen ERD und RS kann synchronisiert werden (ER-Mapping, Rückverwandlung ohne Informationsverlust)



MOF und EMOF

- MOF erweitert ERD um Mehrfachvererbung und Methodensignaturen
- MOF muss auf Java abgeflacht werden:
  - Mehrfachvererbung
  - ungerichtete Assoziationen
- EMOF lässt nur zu
  - einfache Vererbung
  - gerichtete Assoziationen
- EMOF kann direkt auf Java oder C# abgebildet werden

12.2.3 Technikraum Treeware und Metasprachen für XML

Daten im Baumformat, mit Überlagerungsgraphen

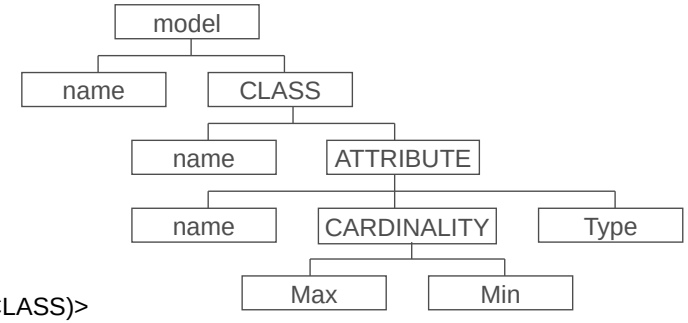
(Technikraum Treeware)

XML

- XML bezeichnet eine Familie von Baumsprachen, hauptsächlich zur Repräsentation von Dokumenten (Daten).
  - Dem Baum überlagert kann ein *Überlagerungsgraph (overlay graph)* sein (Hyperlinks)
- http://www.w3.org/XML
- XML kann zur Spezifikation von Datenkatalogen (data dictionaries) eingesetzt werden, z.B. bei Content Management Systems (CMS)
- XML wird oft als Austauschformat benutzt
- XML besitzt mehrere Metasprachen:
  - Document Type Definitions (DTD)
  - XML Schema Definition (XSD)
  - RelaxNG

12.2.3.1 Document Type Definition (DTD) für XML

Eine **DTD** ist eine einfache Metasprache




```
<! ELEMENT model (name, CLASS)>
<! ELEMENT CLASS  (name, ATTRIBUTE*)>
<! ELEMENT name #PCDATA REQUIRED>
<! ELEMENT ATTRIBUTE  (name, CARDINALITY?, Type?)>
<! ELEMENT CARDINALITY  (Min, Max)>
<! ELEMENT Min  (#PCDATA) REQUIRED>
<! ELEMENT Max  (#PCDATA)>
<! ELEMENT Type  (#PCDATA)>
```

Quelle: Tolksdorf, R.: XML und darauf basierende Standards: Die neuen Aufzeichnungssprachen des Web; Informatik-Spektrum 22(1999) H. 6, S. 407 - 421

Property Lists in XML


```
<!ENTITY % plistObject "(array | data | date | dict | real | integer | string | true
false )" >
<!ELEMENT plist %plistObject;>
<!ATTLIST plist version CDATA "1.0" >
<!-- Collections -->
<!ELEMENT array (%plistObject;)*>
<!ELEMENT dict (key, %plistObject;)*>
<!ELEMENT key (#PCDATA)>
<!--- Primitive types -->
<!ELEMENT string (#PCDATA)>
<!ELEMENT data (#PCDATA)> <!-- Contents interpreted as Base-64 encoded -->
<!ELEMENT date (#PCDATA)> <!-- Contents should conform to a subset of ISO 8601
    (in particular, YYYY '-' MM '-' DD 'T' HH ':' MM ':' SS 'Z'.  Smaller units
     may be omitted with a loss of precision) -->
<!-- Numerical primitives -->
<!ELEMENT true EMPTY>  <!-- Boolean constant true -->
<!ELEMENT false EMPTY> <!-- Boolean constant false -->
<!ELEMENT real (#PCDATA)> <!-- Contents should represent a floating point
    number matching ("+" | "-")? d+ ("."d*)? ("E" ("+" | "-") d+)?
    where d is a digit 0-9.  -->
<!ELEMENT integer (#PCDATA)> <!-- Contents should represent a (possibly
    signed) integer number in base 10 -->
```

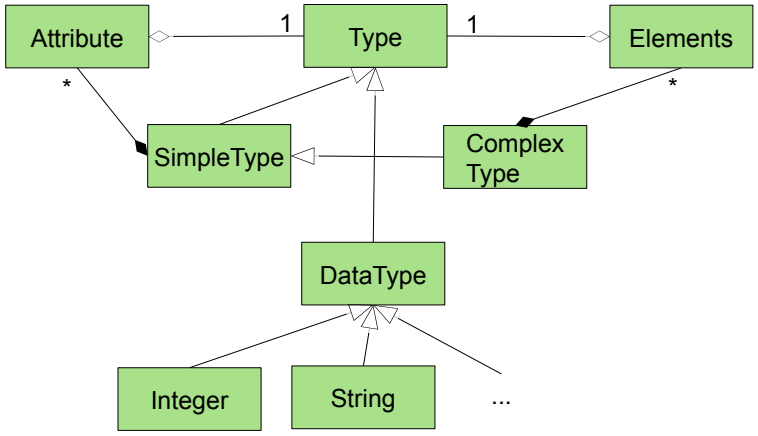

Beispiel für eine Dokumenteninstanz

- Verwendet alle ELEMENT als “tags”

```
<? xml version = „1.0“?>
<! DOCTYPE oomodel SYSTEM „oom.dtd“>
<model>
   <name> „Model 1“ </name>
   <CLASS>
     <name> „Class 1“ </name>
     <ATTRIBUTE>
          <name> „attribute 1“ </name>
          <CARDINALITY>
                <Min> „1“ </Min>
                <Max> „1“ </Max>
          </CARDINALITY>
          <Type> „Class 1“ </Type>
     </ATTRIBUTE>
   </CLASS>
</model>
```


12.2.3.2 XML Schema Definition (XSD)

- XSD ist die Meta-Sprache zur Definition von XML-Sprachen, d.h. Zur Festlegung der validen “tags” eines XML-Dokuments
  - Wiederum in XML-Syntax
- MOF-Metamodell von XSD:



XML Example


```
<treatment>
  <patient insurer="1577500" nr='0503760072' />
  <doctor  city   ="HD"     nr='4321' />
```

simple

```
  <service>
    <mkey>1234-A</mkey>
    <date>2001-01-30</date>
    <diagnosis>No complications.
    </diagnosis>
  </service>
```

complex

```
</treatment>
```

[W. Löwe, Växjö Universitet]

Example: Definition of Simple and Complex Tag Types with XML Schema (XSD)


```
<simpletype name='mkey' base='string'>
  <pattern value='[0-9]+(-[A-Z]+)?'/>
</simpletype>

<simpletype name='insurer' base='integer'>
  <precision value='7'/>
</simpletype>

<simpletype name='myDate' base='date'>
  <minInclusive value='2001-01-01'/>
  <maxExclusive value='2001-04-01'/>
</simpletype>

<complextype name='treatment'>
  <element name='patient' type='patient'/>
  <choice>
    <element ref='doctor'/>
    <element ref='hospital'/>
  </choice>
  <element ref='service' maxOccurs='unbounded'/>
</complextype>
```


Example: XML Schema Attributes


```
<complextype name='patient' content='empty'>
  <attribute ref ='insurer' use='required'/>

  <attribute name='nr' use='required'>
    <simpletype base='integer'>
      <precision value='10'/>
    </simpletype>
  </attribute>

  <attribute name='since' type='myDate'/>
</complextype>
```


12.3 Anfragesprachen (Query Languages, QL)

DQL – Data Query Languages

CQL – Code Query Languages

DQL und CQL in Werkzeugen

- Im Allgemeinen leisten DQL:
  - Beantwortung von Fragen über die Daten eines Repositorium oder eines Stroms (Kanal)
  - Datenanalysen wie Metriken (“Business Intelligence”)
- In Softwarewerkzeugen leisten CQL
  - Beantwortung von Fragen über die Artefakte eines Repositoriums
    - Programmanalysen
    - Metriken (Zählen von Softwareeinheiten)
  - Filtern von Artefakten, die über einen Strom eingehen
    - Mustersuche in Code
- Wir sind insbesondere an strombasierten CQL-Werkzeugen interessiert

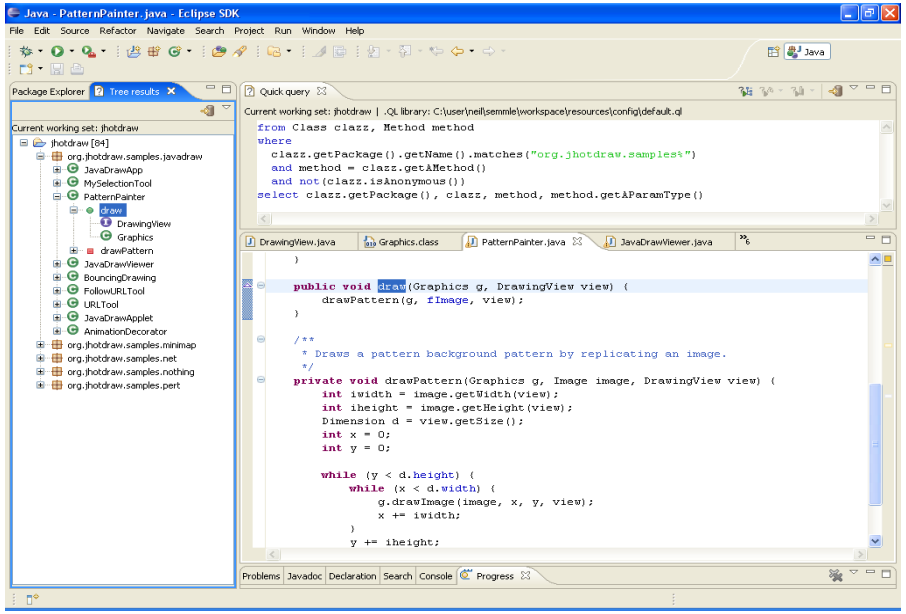
12.3.1 .QL – relational Queries on Source Code in Technical Space Java

Courtesy to Florian Heidenreich
http://semmle.com

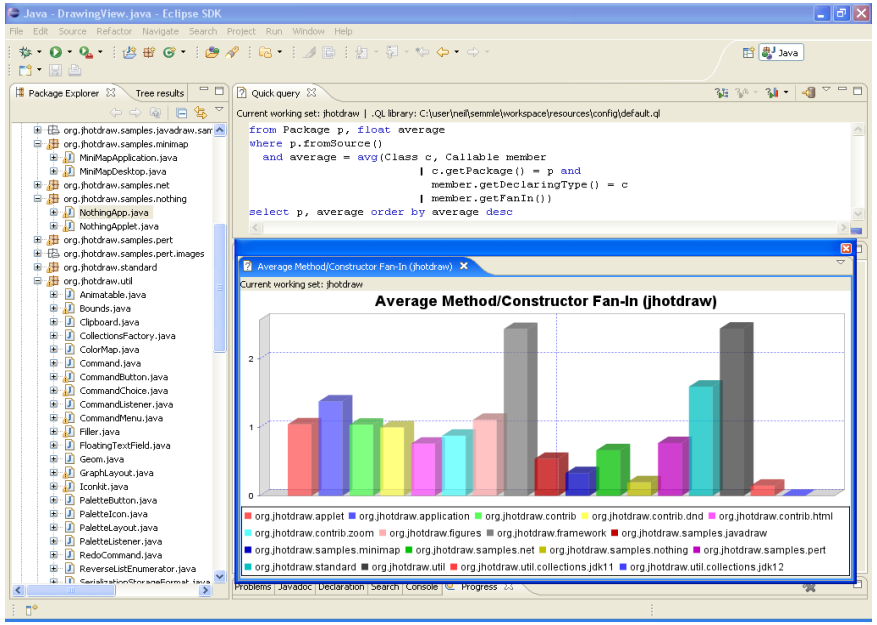
Die repository-zentrierte CQL .QL

- .QL ist eine objektorientierte Query-Sprache, entwickelt in der Gruppe von Oege de Moor (Oxford)
  - Semmle ist die Ausgründung, die Produkte auf der Basis von .QL anbietet
  - SemmleCode unterstützt Anfragen auf Repositorien mit Java Quellcode
- Mit Semmle kann man also Code abfragen, so wie man mit SQL oder relationale Daten oder mit Xcerpt XML abfragen kann
  - .QL eignet sich also für Prozesse, die Code-Ein- und Ausgabeströme besitzen (Code-Transformatoren)
- Klassen werden als Mengen aufgefasst

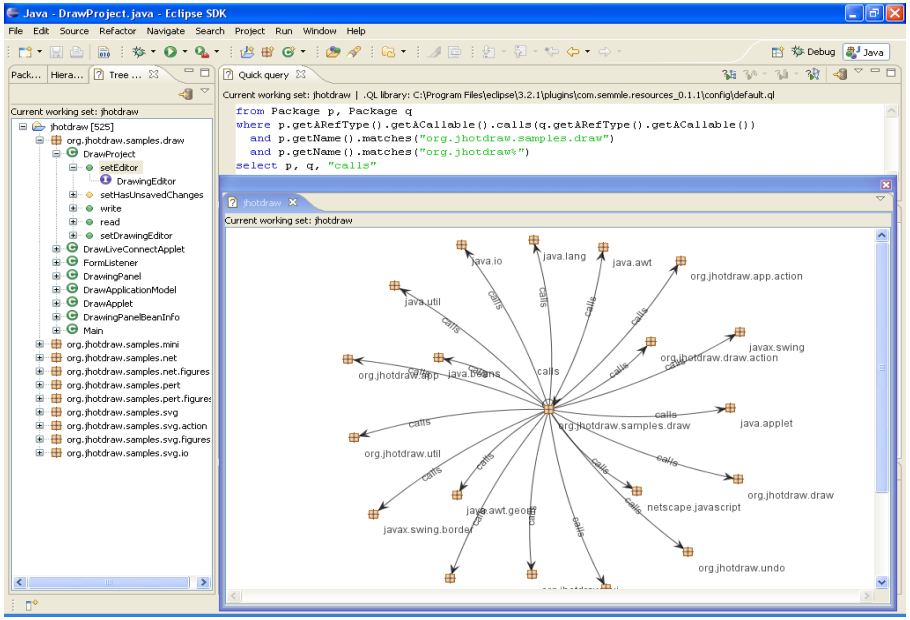
Code Display



Statistics



Graph Visualization



SemmleCode – Query Language

- Select Statements
- Lokale Variablen
- Nichtdeterministische Methoden
- Casts
- Chaining
- Aggregationen
- Eigene Klassen

Select Statements (1)

Finde alle Klassen c die zwar **`compareTo`** implementieren, aber **`equals`** nicht überschreiben

from Class c
where
c.declaresMethod("*compareTo*")
and not (c.declaresMethod("*equals*"))
select
c.getPackage(), c

Select Statements (2)

Finde alle **`main`**-Methoden die in einem Paket deklariert sind, welches auf „demo“ endet

from Method m
where
m.hasName(“*main*")
and m.getDeclaringType().getPackage().getName().matches("%demo")
select
m.getDeclaringType().getPackage(),
m.getDeclaringType(),
m

Lokale Variablen

Finde alle Methoden die `System.exit(…)` aufrufen

```
from Method m, Method sysexit, Class system
where
    system.hasQualifiedName("java.lang", "System")
    and sysexit.hasName("exit")
    and sysexit.getDeclaringType() = system
    and m.getACall() = sysexit
select m
```


Nichtdeterministische Methoden

Erzeuge einen Aufrufgraph zwischen den Paketen eines Projekts

```
from Package caller, Package callee
where caller.getARefType().getACallable().calls(
          callee.getARefType().getACallable())
    and caller.fromSource()
    and callee.fromSource()
    and caller != callee
select caller, callee
```


Casts

Finde alle Abhängigkeiten – auch Nutzung von Typen zwischen den Paketen eines Projekts

```
from MetricPackage p
select p, p.getADependencySrc().getARefType()
```


Chaining (Multiple Source - Multiple Target Graph Reachability Problem, MSMT)

Finde alle Paare (s,t), so dass

- t eine direkte Oberklasse von s ist
- und beide Oberklassen von `org.jfree.data.gantt.TaskSeriesCollection`
- und t nicht `java.lang.Object` ist

```
from RefType tsc, RefType s, RefType t
where
    tsc.hasQualifiedName("org.jfree.data.gantt","TaskSeriesCollection")
    and s.hasSubtype*(tsc)
    and t.hasSubtype(s)
     and not(t.hasName("Object"))
select s,t
```


Aggregationsfunktionen

Ermittle die durchschnittliche Anzahl an Methoden pro Typ und Paket

```
from Package p
where p.fromSource()
select p, avg(RefType c |
          c.getPackage() =p |
    c.getNumberOfMethods())
```

Andere Aggregationsfunktionen: count, sum, max, min

Orientiert sich an der „Eindhoven Quantifier Notation“ (Dijkstra et al.)

Eigene Klassen (1)


```
class VisibleInstanceField extends Field {
  VisibleInstanceField() {
    not (this.hasModifier("private")) and
    not (this.hasModifier("static"))
  }

  predicate readExternally() {
    exists (FieldRead fr |
          fr.getField()=this and
          fr.getSite().getDeclaringType() != this.getDeclaringType())
  }
}
```


Eigene Klassen (2)


```
from VisibleInstanceField vif
where vif.fromSource() and not (vif.readExternally())
select vif.getDeclaringType().getPackage(),
          vif.getDeclaringType(),
          vif
```

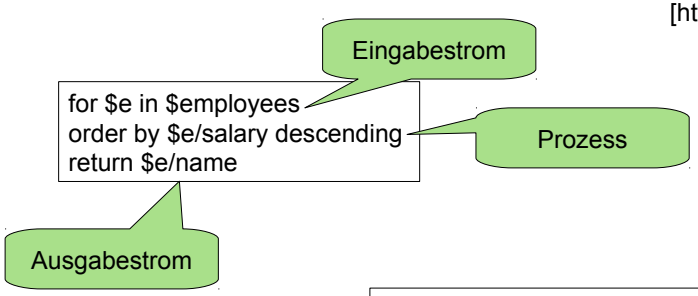

12.3.2 XQuery

The standard from W3C

Xquery

- http://www.w3.org/XML/Query/
- Eine Anfragesprache des W3C für XML queries
- In Schleifen werden Muster ausgewertet
  - Die Schleifen ähneln DFD-Prozessen

[http://www.w3.org/TR/xquery/]




```
for $b in $books/book
stable order by $b/title
      collation "http://www.example.org/collations/fr-ca",
  $b/price descending empty least
return $b
```


Hamlet, this time Marked-Up


```
<?xml version="1.0"?>
<!DOCTYPE PLAY SYSTEM "play.dtd">
<PLAY> <TITLE>The Tragedy of Hamlet, Prince of Denmark</TITLE> <FM>
<P>Text placed in the public domain by Moby Lexical Tools, 1992.</P>
<P>SGML markup by Jon Bosak, 1992-1994.</P> <P>XML version by Jon Bosak, 1996-1998.</P>
<P>This work may be freely copied and distributed worldwide.</P>
</FM>
<PERSONAE>
<TITLE>Dramatis Personae</TITLE>
<PERSONA>CLAUDIUS, king of Denmark. </PERSONA>
<PERSONA>HAMLET, son to the late, and nephew to the present king.</PERSONA>
<PERSONA>POLONIUS, lord chamberlain. </PERSONA>
<PERSONA>HORATIO, friend to Hamlet.</PERSONA>
</PERSONAE>

<ACT><TITLE>ACT I</TITLE>
<SCENE><TITLE>SCENE I.  Elsinore. A platform before the castle.</TITLE>
<STAGEDIR>FRANCISCO at his post. Enter to him BERNARDO</STAGEDIR>
<SPEECH> <SPEAKER>BERNARDO</SPEAKER> <LINE>Who's there?</LINE> </SPEECH>
<SPEECH> <SPEAKER>FRANCISCO</SPEAKER> <LINE>Nay, answer me: stand, and unfold yourself.</LINE> </SPEECH>
<STAGEDIR>Exeunt</STAGEDIR>
</SCENE>
</ACT>
<ACT><TITLE>ACT II</TITLE>
...
</ACT>
</PLAY>
```

[Wikipedia]

Xquery is a Mixed Language

The following script produces a list of speakers of the hamlet plot

```
<html><head/><body>
 {
   for $act in doc("hamlet.xml")//ACT
   let $speakers := distinct-values($act//SPEAKER)
   return
     <div>
       <h1>{ string($act/TITLE) }</h1>
       <ul>
       {
         for $speaker in $speakers
         return <li>{ $speaker }</li>
       }
       </ul>
     </div>
 }
 </body></html>
<?xml version="1.0"?>
```

[Wikipedia]

Resume Anfragesprachen

- Anfragesprachen können eingesetzt werden, um
  - Anfragen an Artefakte in Repositorien abzusetzen
  - Transformationen von Strömen zu organisieren
  - Werkzeuge zu beschreiben, die einfach zu komponieren sind
- Sie eignen sich daher für die Spezifikation von Funktionen, Aktionen und Prozessen, z.B. in DFD.
- Sie eignen sich auch, Bedingungen zu prüfen, auch Konsistenzbedingungen

12.4 Datenkonsistenzsprachen (Data Constraint Languages, DCL)

Werkzeugunterstützung für DDL (Rpt.)

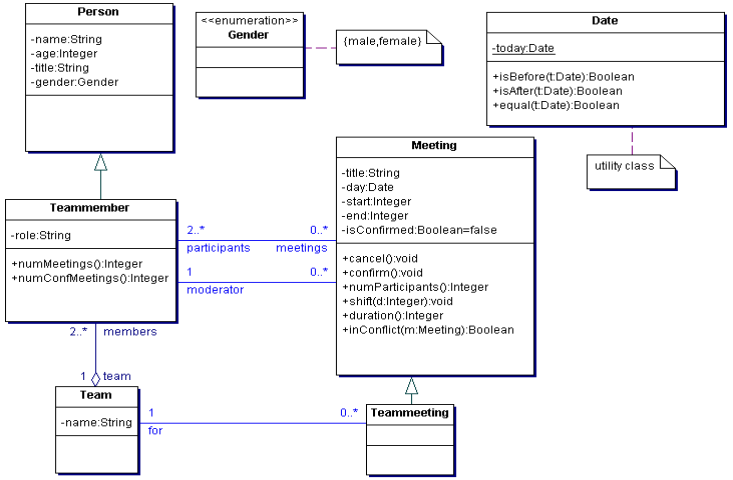
Wir hörten, Prüfung der allgemeinen Integritätsregeln für relationale Datenmodelle, ERD und UML-CD, sei notwendig für:

- Bereichsprüfungen
- Eindeutigkeit
- Minimalität
- Fremdschlüssel-Verbindung
- Referentielle Integrität
- Verbotene Kombinationen von Daten

- Anstatt diese fest im Werkzeug zu verdrahten, d.h. fest einzuprogrammieren, kann man sie mit Konsistenzprüfungssprachen (DCL) spezifizieren
  - und dann vom Werkzeug aufrufen
- Man spricht von **Invarianten** der Artefakte, die durch eine DCL festgelegt werden
- DCL bauen oft auf DQL auf und prüfen bestimmte Anfrageresultate

12.4.1: OCL für Invarianten von UML-Klassendiagrammen

- Mehr in Vorlesung ST-II



Invariante - Beispiele

Beispiel

```
context Meeting inv: self.end > self.start
```

Äquivalente Formulierungen

```
context Meeting inv: end > start
    -- self bezieht sich immer auf das Objekt, für das das Constraint
    -- berechnet wird

context Meeting inv startEndConstraint:
self.end > self.start
    -- Vergabe eines Namens für das Constraint
```

- Sichtbarkeiten von Attributen u.ä. werden durch OCL standardmäßig ignoriert.
- Mehr Info in der Vorlesung “Konsistenzprüfung mit OCL” in der ST-II, Dr. Birgit Demuth

12.4.2. Spider Diagramme

- http://en.wikipedia.org/wiki/Spider_diagram
- S. Kent. Constraint Diagrams: Visualizing Invariants in OO Modelling. Proceedings of OOPSA 97, ACM Press, Oct. 97, pp. 327-341.
- S. Kent and J. Howse. Mixing Visual and Textual Constraint Languages, UML 99, IEEE press, Oct 1999.

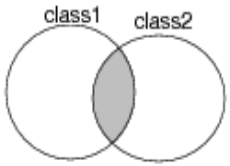
- Spider-Diagramme sind äquivalent zu monadischer Logik 2. Stufe (monadic second order logic, MSOL).
  - Sie beinhalten damit OCL, das Logik 1. Stufe modellieren kann
- Die folgenden Diagramme stammen aus der Diplomarbeit: J. Lövdahl, Towards a Visual Editing Environment for the Semantic Web. Linköpings universitet, 2002.

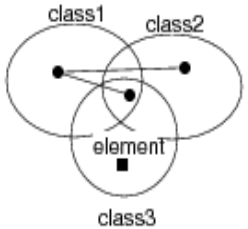
Simple Spider Diagrams

- Existential Logic (propositional logic with existential quantifiers)

An object of class1 has an object of class2
and an object in class1^class2^class3
and class3\class1\class2 is not empty

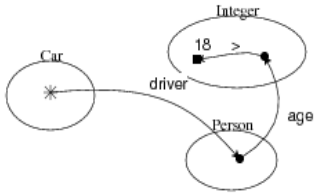
class1^class2



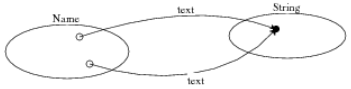


- All quantifiers are possible (star symbol)

All cars must be driven
by a person older than 18

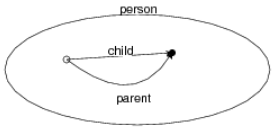


There are no two names that have the same string

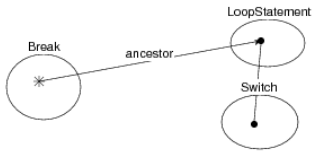


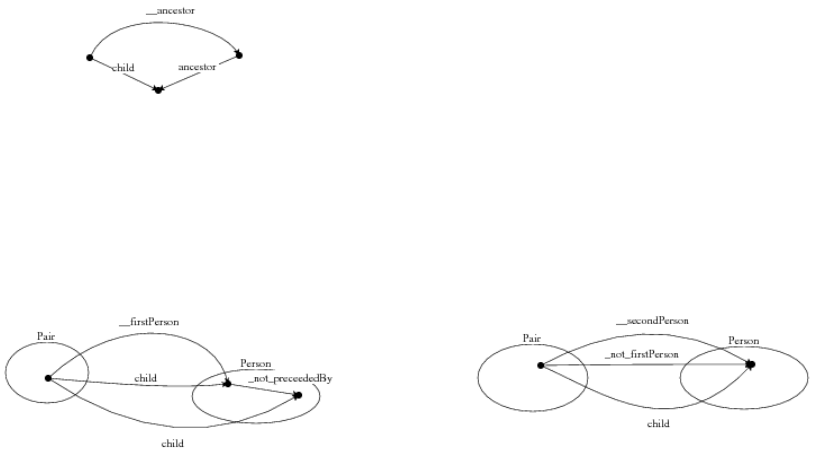
Other constraints

For every person, there is no child that has no parent



All Break statements must have a LoopStatement as ancestor,
which is related to a Switch statement

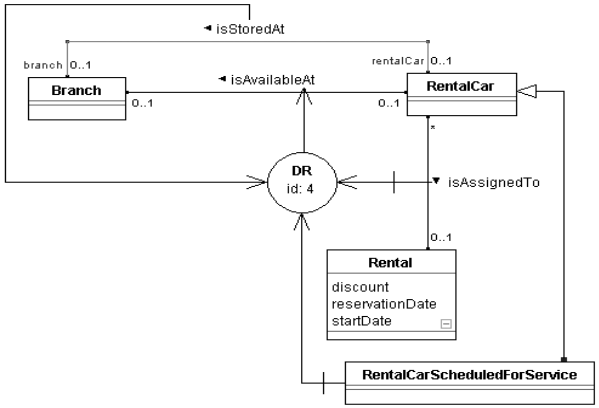




12.4.3. URML

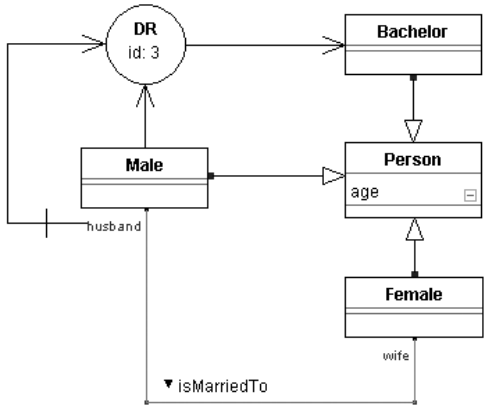
- URML http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=URML
- Beispiel: Modeling a Derivation Rule for Defining an Association

If a rental car is stored at a branch, is not assigned to a rental and is not scheduled for service, then the rental car is available at the branch.



Modeling a Derivation Rule with a Role Condition

A bachelor is a male that is not a husband.



12.5 Data Transformation Languages (DTL)

Text, XML, Term, and Graph Rewriting
see separate Chapter

DTL und DML

- Mit DML (Datenmanipulationssprachen) formt man Daten um.
- **Deklarative DML (Datentransformationssprachen, DTL)** bestehen aus Regeln, die ein Repositorium ohne die Spezifikation weiteren Steuerflusses transformieren.
  - Termersetzungsregeln, die Bäume oder Dags transformieren (Kap. 35)
  - Graphersetzungsregeln, die Graphen und Modelle transformieren (Kap. 36)
- **Imperative DML (allgemeine DML)** kennen Zustände und Seiteneffekte.

- Beispiele von deklarativen DML (DTL):
  - Term Rewriting:
    - Xquery
    - Xcerpt als Strom-Manipulationssprache
  - Graph Rewriting:
    - EARS, XGRS
    - Fujaba

DRL

- Restrukturierung von Daten bedeutet, sie zu transformieren, und bestimmte Invarianten zu erhalten.
- Daher ist eine DRL eine spezielle DML, mit der speziellen Eigenschaft, dass nach bestimmten Transformationsschritten Invarianten mit einer DCL überprüft werden.
- Beispiel:
  - Man transformiert eine ER-Datenbank mit Hilfe von DFD, Xcerpt, oder XGRS in eine zweite Datenbank
  - und prüft ihre Konsistenz nach jedem Transformationsschritt mit einer DQL.

12.6 Data Manipulation Languages (DML) and Behavioral Specification Languages (BSL)

Sprachen zur Manipulation von Daten, mit globalem Zustand

12.6.1 Datenflussdiagramme (DFD)

Wiederholung aus ST-II

DFD entsprechen speziellen Petrinetzen bzw. Workflowsprachen, die *keinen globalen Zustand* verwalten.

DFD vermeiden globale Speicher, sondern arbeiten mit partionierten Repositorien. Daher sind sie für die Modellierung von Parallelität sehr gut geeignet, denn sie beschreiben die Compute-Data-Lokalität.

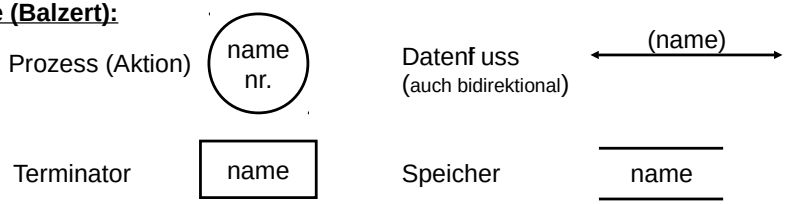
Datenflußmodellierung

- **Datenfluss-Modellierung:** Prozesse (Iterierte Aktionen) auf Datenflüssen, ohne gemeinsames Repository
  - Datenfluss (Datenströme, streams, channels, pipes) zwischen Prozessen (immerwährenden Aktivitäten auf einem Zustand)
  - Datenflussdiagramme werden für strukturierte Prozesse (Geschäftsprozesse, technische Prozesse, Abläufe in Werkzeugen) eingesetzt
- Datenfluss-Modellierung ist Hauptbestandteil der **Strukturierten Analyse (SA)**

DFD-Modellierung

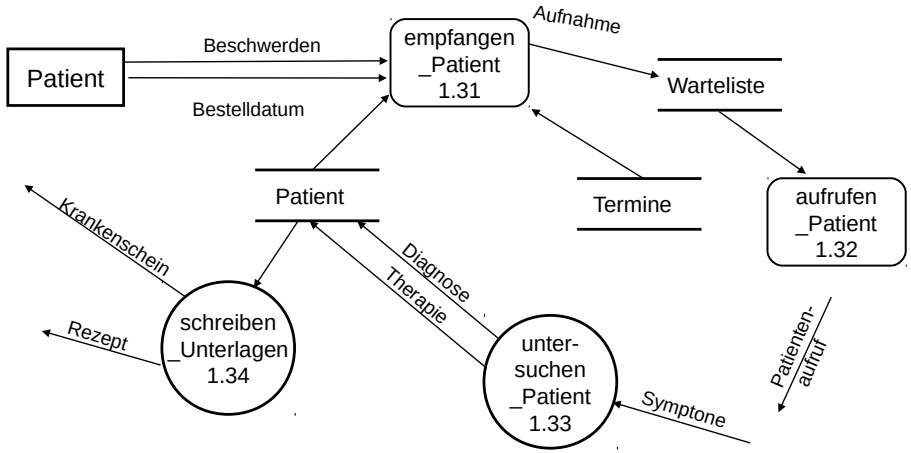
- Hierarchische (reduzible) Prozessspezifikationen:
  - Kontextdiagramm (oberstes Diagramm, mit Terminatoren)
  - Parent-Diagramme
  - Child-Diagramme (Verfeinerte Prozesse)
- Datenkatalog wird benutzt zur Typisierung (spezifiziert in einer DDL)
- Minispezifikationendienen der Beschreibung der in Elementarprozessen durchzuführenden Transformationen.
  - mit Pseudocode
  - mit einer Transformationssprache wie Xcerpt

Symbole (Balzert):

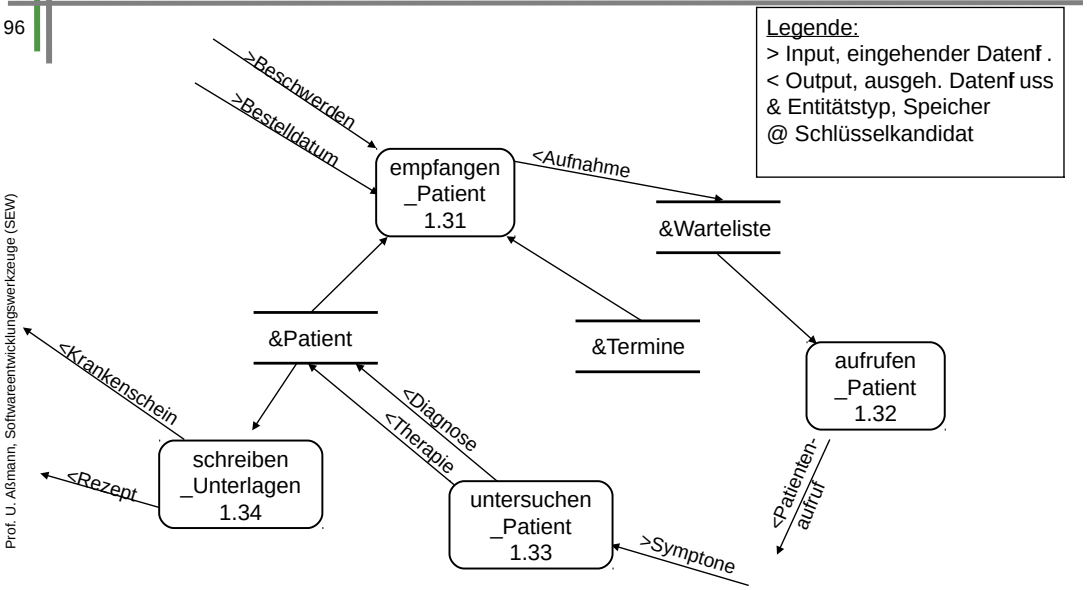


DFD-Beispiel "behandeln_Patient"

- Prozesse auf Datenströmen, auch Geschäftsprozesse
- Kein zentrales Repositorium, lokale Daten, explizite Definition des Datenflusses
- UML notiert Aktivitäten und Prozesse mit Ovalen, SA/Balzert mit Kreisen



Verfeinertes DFD-Beispiel "behandeln_Patient"



Wohlgeformtheitsregeln (Integritätsregeln) von DFD

- **Syntaktische Regeln** zur graphischen DFD-Darstellung:
  - Jeder Datenfluss muß mit mindestens einem Prozess verbunden sein.
  - Datenflüsse zwischen Terminatoren und zwischen Speichern sind nicht erlaubt.
  - Datenspeicher, die nur einseitig beschrieben (ohne zu lesen) und nur einseitig gelesen (ohne zu beschreiben) werden, sind nicht erlaubt.
  - Prozesse, die Daten ausgeben, ohne sie erhalten zu haben oder umge- kehrt, die Daten erhalten, ohne sie auszugeben oder zu verarbeiten, sind nicht erlaubt.
  - Im Kontextdiagramm darf es keine Speicher geben, in Verfeinerungen keine Terminatoren
  - Jeder Prozess, Speicher und Datenfluss muss einen Namen haben. Nur in dem Fall, wo der Datenfluss alle Attribute des Speichers beinhaltet, kann der Daten- flussname entfallen.
- **Semantische Konsistenzregeln** zur Wohlgeformtheit der Namensgebung:
  - Prozessnamen: Verb_Substantiv zur aussadgekräftigen Beschreibung (z.B. berechne_Schnittpunkt)
  - Datenflussnamen: [<Modifier>]Substantiv beschreibt momentanen Zustand des Datenflusses (z.B. <neue>Anschrift)
  - Speichernamen: Substantiv, das den Inhalt des Speichers (identisch Entity im DD) beschreibt (z.B. Adressen)

Weiterführende Literatur: [BAL]

Integritätsregeln der DFD-Erstellung: Balancieren zwischen DFD und anderen Sprachen

- **Vertikales Balancing** zwischen Knoten und Verfeinerungen
  - Alle Komponenten der im Vater referenzierten Flüsse sind zu benutzen.
- **Horizontales Balancing** zwischen DFDs und Minispezifikationen:
  - Jede Minispezifikation muß genau einem (Primitiv-)Knoten zuordenbar sein und umgekehrt
  - Alle Schnittstellen zu Knoten müssen in der MSpec referenziert sein und umgekehrt.
  - Alle Ausgaben jedes Prozesses müssen aus seinen Eingaben erzeugbar sein (korrekte Nutzung von Speichern!).
- **Balance von DFDs zum Data Dictionary:**
  - Zusammensetzung jedes Datenflusses und Speichers vollständig im DD beschrieben
  - Jedes Datenelement im DD muß in anderem Datenelement oder DFD vorkommen (Vollständigkeit)
- **Balance von ERD zu DFDs und Minispezifikationen:**
  - Jeder Speicher und Typ eines Kanals in einem DFDs muß einem Entitytyp des ERD entsprechen.

DFD als BSL mit privaten Daten

- DFD verzichten auf ein globales Repositorium, sondern spalten die Daten in “private” Speicher auf,
  - für die explizit spezifiziert wird, wohin ihre Daten fließen
- DFD sind sehr gut geeignet für die Spezifikation von Werkzeugverhalten
  - Datenabhängigkeiten sind immer klar, da explizit spezifiziert
  - Natürliche Parallelität
  - Einfache Komposition durch Anfügen von weiteren Datenflüssen und Teilnetzen

12.6.2 Deklarative Sprachen zur Verhaltensspezifikation (BSL)

Mit formaler Semantik, damit Beweise möglich werden

.. siehe ST-2 ..

Automaten, Petri-Netze und Workflowsprachen

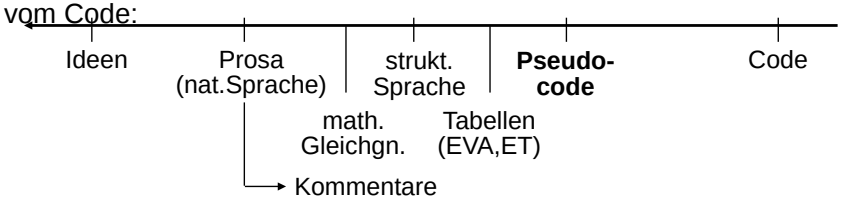
- **Zustandsorientierte Verhaltenssprachen** erlauben die Spezifikation von **Interpretern (operationale Semantik)**
  - Automaten wurden bereits in Softwaretechnologie-I behandelt.
  - Petri-Netze und Workflowsprachen werden ausführlich in Softwaretechnologie-II behandelt.
  - Petrinetze und Workflowsprachen kennen einen globalen Zustand, sind also allgemeine DML.
  - Bitte schlagen Sie dort die entsprechenden Kapitel nach.

12.6.3 Pseudocode

http://en.wikipedia.org/wiki/Pseudocode

Pseudocode

- **Pseudocode** besteht aus strukturiertem Text mit Schlüsselwörtern für Strukturkomponenten, z. B. seq, endseq, if, then, else, endif, while, endwhile, call, action, stop,...
  - "freisprachl. Text" ist als Kommentar eingeschlossen
- Werkzeugunterstützung:
  - Syntaxkontrolle mit Parsern für Pseudocode
  - Codeerzeugung (Codegerüst + Kommentare)
  - Dokumentationserstellung (Einrückdiagramme, PAP, Struktogramm)
- Pseudocode liegt auf der Hesse'schen Skala des Formalisierungsgrades links vom Code:



Beispiele für Pseudocode

- Die in Pseudocode vorkommenden formalen Namen sind :
  - **Titel** von Prozeduren und Prozessen
  - Im Datenkatalog erklärte Datenfluss- und Attributnamen (Referenzierung)
  - Pseudocode-Schlüsselwörter
  - lokale Namen und freisprachlicher Text zur Verbesserung der Lesbarkeit
  - Makros zur Zusammenfassung mehrerer Worte.

```
prozess empfangen_Patient 1.3.1
fuer &Patient
      mit >Bestelldatum = Datum in &Termine und >Beschwerden
               wenn Name*des Patienten* in &Patient
                      sonst "aktualisieren_Patient 1.1"
      wenn keine >Beschwerden und >Bestelldatum ungueltig
               dann „vergeben_Termin 1.2“
               sonst Uebernahme Patientendaten aus &Patient
          alle Unterlagen fuer Arzt aufbereiten
<Aufnahme Name*des Patienten* in &Warteliste
      wenn @Bestdat+Zeit = Kalenderdatum + Uhrzeit
               dann Terminpatient Platz m+1*
                    vorhergehender Terminpatient m*
               sonst Platz n+1*n Anzahl aller Patienten im Wartezimmer*
```


Beispiele für Pseudocode (2)


```
action empfangen_Patient
        while (Patienten oder Praxisoeffnung)
        seq Eingabe >Bestelldatum, >Beschwerden
        if (@Bestdat+Uhrzeit enth. &Termine)
        then Bestellpatient
        else if (@Gebdatum+Name enth. &Patient)
               then ziehen Patientenakte
               else call aktualisieren_Patientendaten
             endif
             if (>Beschwerden <> 0*vorhanden*)
               then Unbestellter_Patient
               else call vergeben_Termin
             endif endif
      Aufbereiten aller Unterlagen fuer Arzt endseq
        if (Bestellpatient)
               then <Aufnahme Platz m+1 in &Warteliste
               else <Aufnahme Platz n+1 in &Warteliste
        endif endwhile
stop
```


Unterstützung für Pseudocode

- LaTeX-Distributionen besitzen gute Style-Pakete für Pseudocode:
  - algorithms.sty
  - \usepackage{algpseudocode}
  - \usepackage{algorithmicx}
  - listings.sty
- ELAN, klartextähnliche Programmiersprache
  - http://de.wikipedia.org/wiki/ELAN
  - Teil von Betriebssystem L3, Vorgänger von L4

```
PACKET stack handling DEFINES push,pop,init stack:
  LET max = 1000;
  ROW max INT VAR stack;
  INT VAR stack pointer;
  PROC init stack:
    stack pointer := 0
  END PROC init stack;
  PROC push (INT CONST dazu wert):
    stack pointer INCR 1;
    IF stack pointer > max
      THEN errorstop ("stack overflow")
      ELSE stack [stack pointer] := dazu wert
    END IF
  END PROC push;

  PROC pop (INT VAR von wert):
    IF stack pointer = 0
      THEN errorstop ("stack empty")
      ELSE von wert := stack [stack pointer];
           stack pointer DECR 1
    END IF
  END PROC pop

END PACKET stack handling;
```

- http://os.inf.tu-dresden.de/L3/usrman/node10.html

12.6.4 Metaprogramme

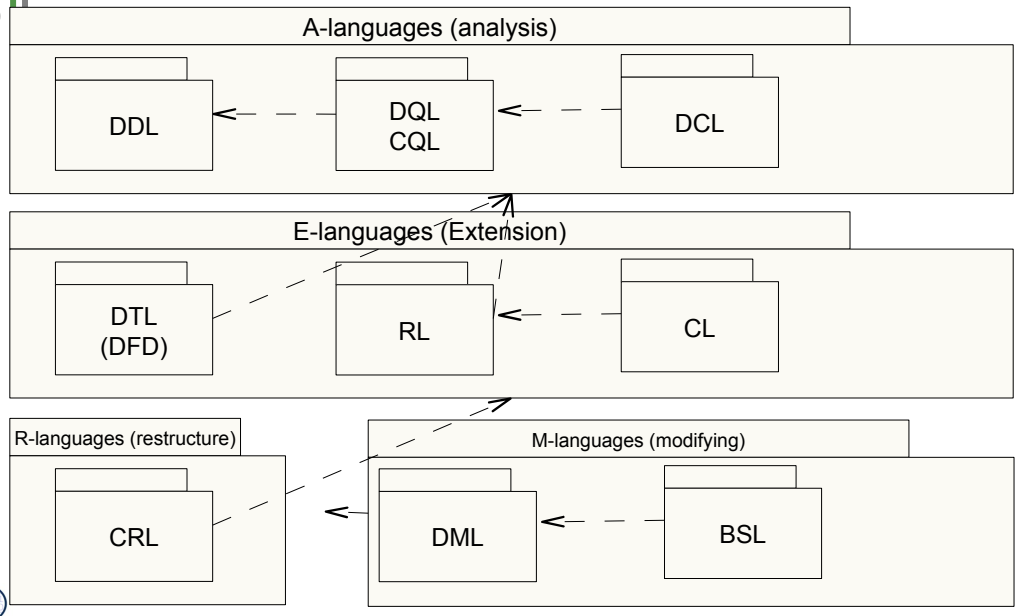
- **Metaprogramme** sind Programme, die Programme erzeugen oder transformieren.
  - Sie haben seit Lisp eine lange Tradition: Lisp erlaubt ungetypte Metaprogramme
- **Dynamische Metaprogramme** laufen ständig mit der Anwendung mit und regenerieren Teile neu.
  - Dynamische Metaprogramme sind allerdings laufzeitintensiv und verhindern eine statische Analyse der dynamisch produzierten Programme.
- **Introspektive Programme** inspizieren die Metadaten oder den Code anderer Programme und Komponenten und ziehen dadurch Schlüsse
- **Codegeneratoren** sind Metaprogramme, die mit Introspektion von Modellen oder Programmen neuen Code produzieren und alten Code nicht invalidieren
- **Statische Metaprogramme** produzieren ein Programm, in dem sie selbst nicht enthalten sind.
  - Sie funktionieren also als reiner Code-Generator.

12.7 Language Hierarchies on M2 (Structure of M2)

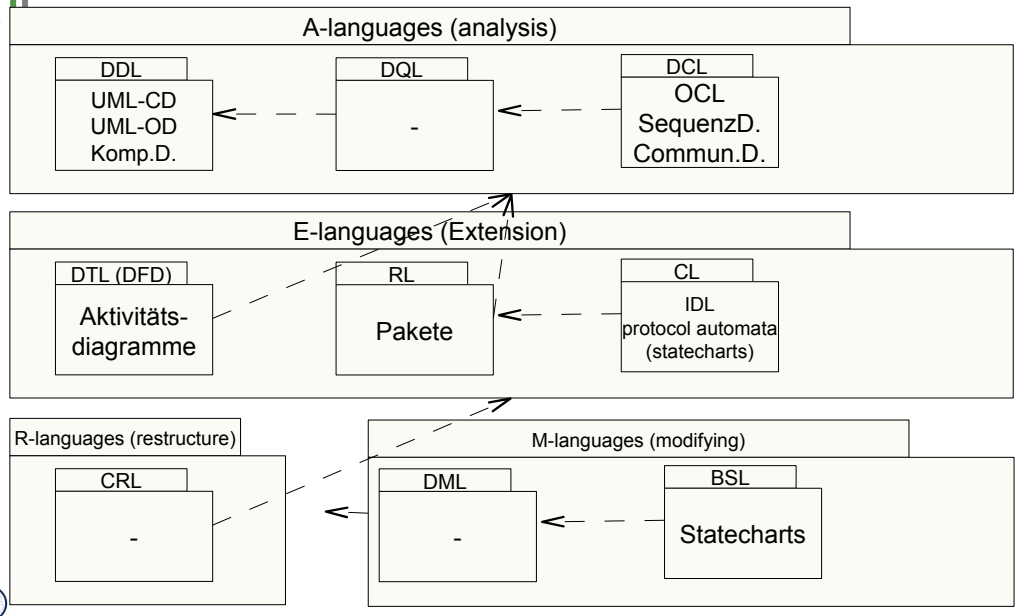
Every technical space has a language hierarchy on M2 with a similar, layered structure.

Every IDE has an underlying language family.

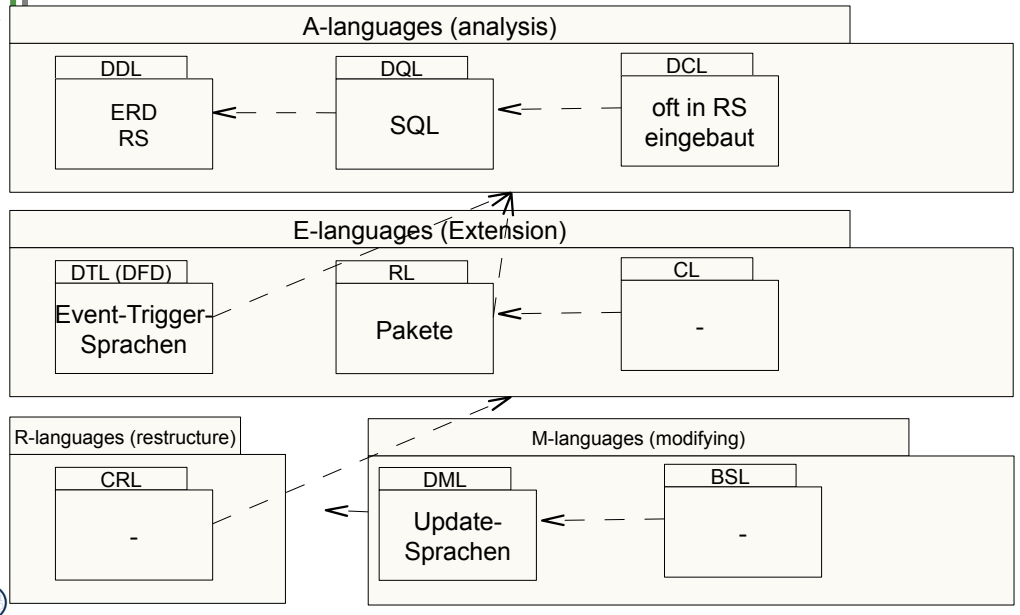
Grundlegende Sprachfamilien (Struktur von M2)



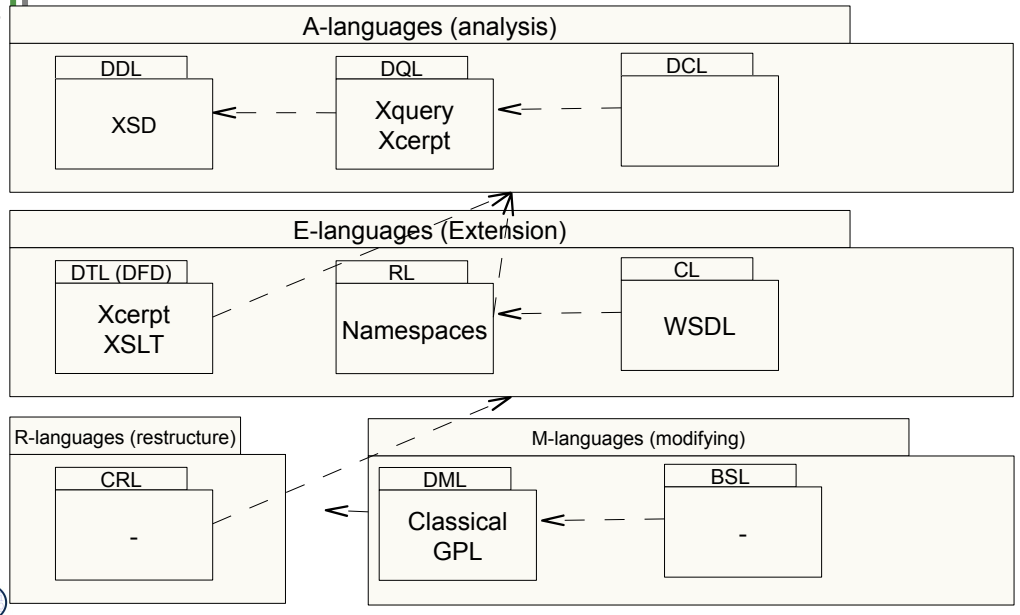
UML-Sprachfamilie (Struktur von M2)



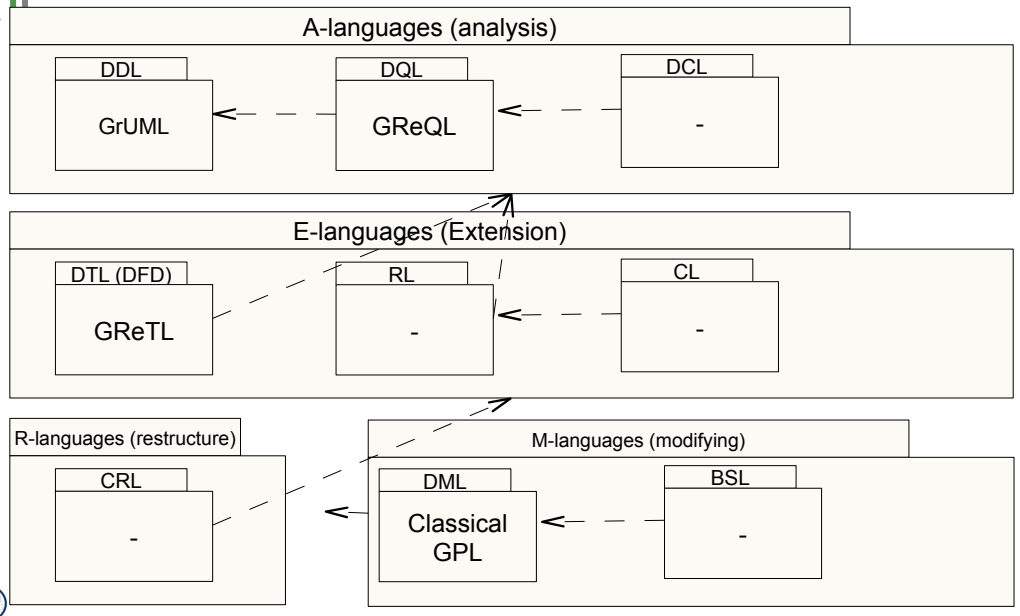
ERD/RS-Sprachfamilie (Struktur von M2)



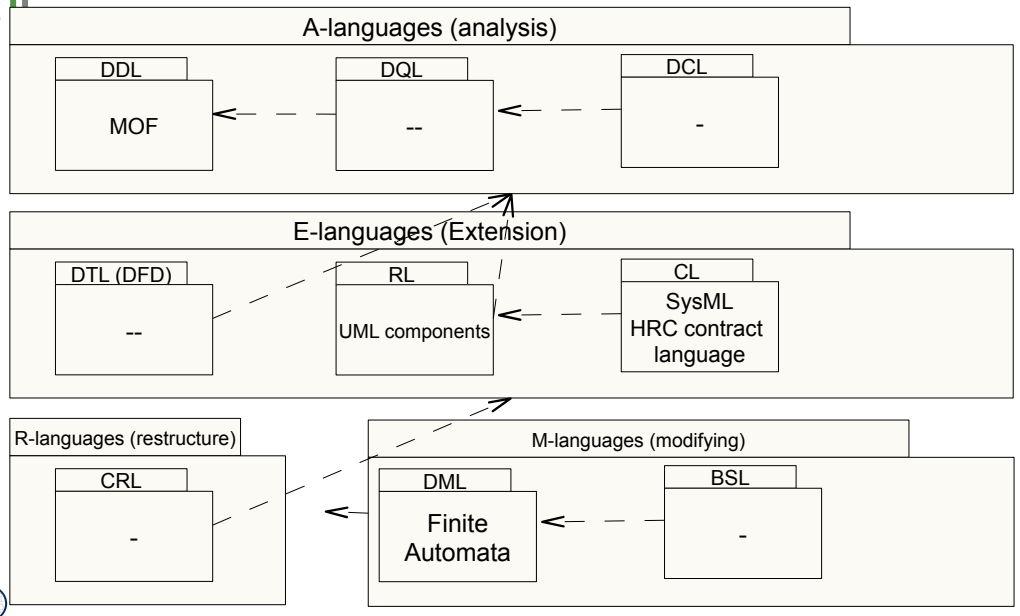
XML-Sprachfamilie (Struktur von M2)



GrUML-Sprachfamilie (Struktur von M2)



HRC-Sprachfamilie für Safety-Critical Embedded Software



Warum ist die genaue Kenntnis der M2-Struktur für Werkzeugnutzung wichtig?

Sprachen, mit denen man Werkzeuge bedient, kombinieren verschiedene Sprachvarianten der Schichten von M2 (**M2-Mix**)

- ERD – MOF – XSD – UML-CD
- Xquery – XSLT – SQL – SPARQL
- OCL – SpiderDiagrams – OntologyLanguages
- Java – C++ – C#
- Petrinetze – DFD – WorkflowNets – BPMN

Domänenspezifische Sprachen bestehen immer aus einem M2-Mix

Methoden benutzen immer einen Mix aus Basistechniken

Warum ist die genaue Kenntnis der M2-Struktur für Werkzeugbau wichtig?

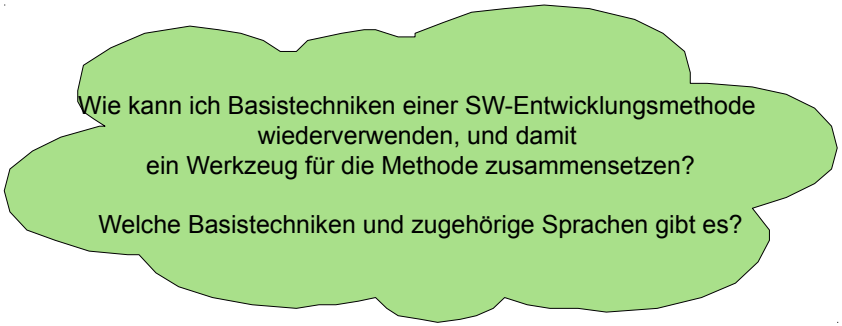
Wie kann ich Metamodelle von Sprachen komponieren, um den Werkzeugbau zu vereinfachen?

- Mit der Komposition der Metamodelle komponieren sich auch bestimmte Teile von Werkzeugen automatisch, z.b. das Repository
- Zur Komposition von Sprachen muss ein *Kompositionssystem* vorliegen
  - Einfaches Beispiel: UML-Paket-Merge-Operator
  - Xcerpt-Regeln sind komponiert aus einem Query-Teil (FROM clause) und einem CONSTRUCT-Teil

Sprachkomposition: Jenseits von Benutzungen von Sprachkonzepten aus tiefer liegenden Stufen der Benutzungshierarchie können Sprachkonzepte mit anderen *komponiert* werden, um zu neuen zu gelangen

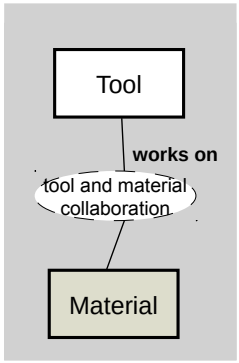
Wie kann ich Werkzeuge zu Basistechniken komponieren?

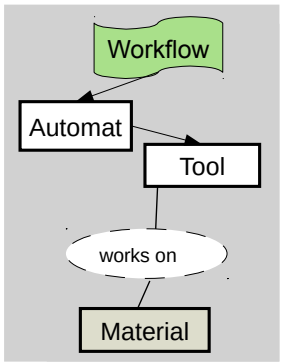
- In jedem Technikraum müssen Werkzeuge, Modellmanagement-Umgebungen und SEU gebaut werden
- Für ein Werkzeug, das eine Entwicklungsmethode unterstützt, oder eine SEU, müssen mehrere Werkzeuge für einzelne Basistechniken komponiert werden
- Wie geht das?
- Idee: Komponiere die Metamodelle der Sprachen/Basistechniken auf M2 und generiere die Werkzeuge!

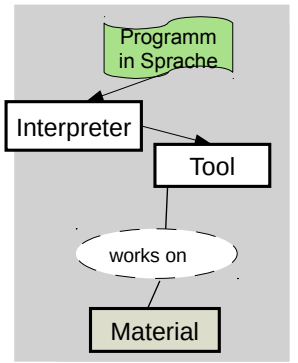


Tools, Automata and Interpreting Tools

- Ein **Werkzeug** ist ein kommando-orientiertes Objekt, mit den Material bearbeitet wird
- Werkzeuge, die einen Arbeitsablauf (Workflow) ausführen und während dessen weitere Werkzeuge anstoßen, nennt man **Automaten**
  - Kann einen Zustandsautomaten, Datenfluss, oder Workflow meinen
- Ein **Interpretierer** ist ein Automat, der eine Sprache interpretiert, um daraus einen Workflow zu gewinnen, mit dem es Material bearbeitet



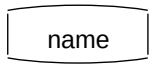
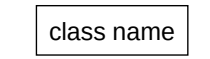
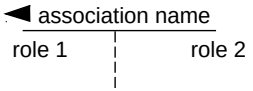
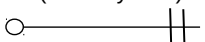
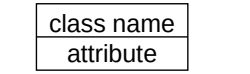




The End – Was haben wir gelernt?

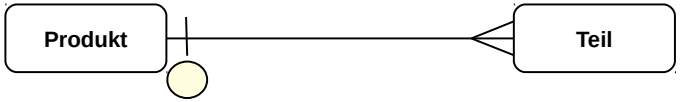
- Sprachfamilien lassen sich abgrenzen nach dem, was sie mit Daten tun.
  - Bestimmte Sprachklassen können einfach mit anderen komponiert werden
  - Werkzeuge, die bestimmte Sprachklassen verwenden, können einfach komponiert werden
  - DFD lassen sich leicht in Aspekte einteilen
- Für den Bau von Werkzeugen ist es wichtig, verschiedene Varianten einer Sprachklasse gegen eine andere austauschen zu könen (z.B. OCL gegen .QL).
- Die Paket- und Schichten-Struktur von M2
- Interpretierende Werkzeuge interpretieren die Programme einer Sprache, um Material zu bearbeiten.

Weitere ERD-Notationsformen

Modellelemente	DSA-Notation	**UML** Version 2.0 (Class Diagram)
Entitytyp		
Beziehungstyp assoziiertes Objekt/Class		
Attribut		
Kardinalität Multiplizität		

Alternative Notationen für Kardinalitäten

Krähenfuß-Notation (crow foot, DSA): Krähenfuß bedeutet „viele“



Schageter/Stucky-Notation (ARIS): Kardinalitätsangaben am Symbol des Beziehungstypes vertauscht

(min,max)-Notation: Die Eckwerte *min* und *max* bezeichnen Unter- und Obergrenze für Teilnahme in einer Beziehung

Vielzahl der Kardinalitätsformen kann verwirren. Entscheidend ist Funktionalität des Werkzeugs.