21. Datenablage (Datenbasis, Repositorium) und Austauschformate

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de
Version 11-0.5, 17.11.11

1) Ziele
2) Architektur
3) Beispiele für Repositorien
4) Austauschformate und Technikraum-Brücken
5) Frameworks zur Werkzeug-integration (PCTE)

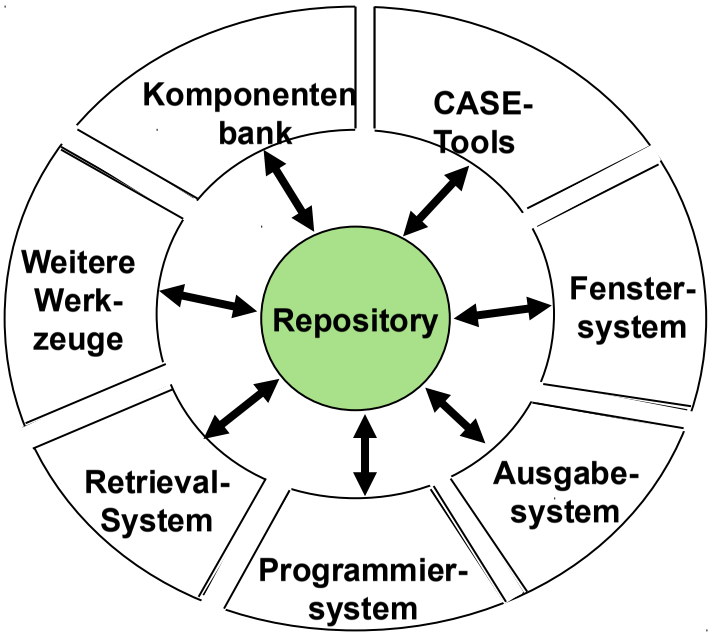
Obligatorische Literatur

- Netbeans Metadata repository (MDR) http://docs.huihoo.com/netbeans/netbeanstp.pdf
- D. Bäumer, D. Riehle, W. Silberski, M. Wulf. Role Object. Conf. On Pattern Languages of Programming (PLOP) 97. http://citeseer.ist.pst.edu/baumer97role.html
- Steffen Staab, Tobias Walter, Gerd Gröner, and Fernando Silva Parreiras. Model driven engineering with ontology technologies. In Uwe Aßmann, Andreas Bartho, and Christian Wende, editors, Reasoning Web, volume 6325, Lecture Notes in Computer Science, pages 62-98. Springer, 2010.
  - http://www.uni-koblenz.de/~staab/Research/Publications/2010/reasoningweb2010.pdf

21.1 Ziele und Aufgaben der Datenbasis (Repositoriums)

Aufgaben des Repository

Eine **Datenbasis (Datenablage, Repositorium, repository)** ist die *Ablage aller Informationen* eines Systems. In einer SEU enthält es alle passiven und aktiven Komponenten zur Handhabung der Basisinformationen der Werkzeuge [12, S. 121ff.].



Zielstellungen für Repositories

- **Transparente persistente Dokumentspeicherung:** Einfache Abbildung aller Datenobjekte (Artefakte) auf persistente Datenstrukturen im Repository auch bei Änderung der Objekte (Systemausfall)
- **Entkopplung:** Physische und logische Datenunabhängigkeit, d.h. Trennung der Programme (Werkzeuge) von physischer bzw. konzeptioneller Repräsentation der Daten
- **Unterstützung von Schemata** beschrieben in DDL**:**
  - **Datenmodell:** Transparent und über mehrere Metamodellebenen für die auszutauschenden Daten zwischen Repository und Werkzeugen leicht handhabbar
  - **Externe Schemata:** Für Steuerung des Zugriffs, in dem zu unterschiedlichen Rollen verschiedene externe Schemata für ein Dokument def nierbar sind
  - **Schema-Änderungen:** Sollen bestehende Daten in der Datenbasis erhalten
  - Flexible **Schnittstellen:** Programmierbare APIs (Anfragesprachen, prozedurale Programmierschnittstellen), möglichst aus Metadaten generiert
  - Ref ektive Schnittstellen: Programmierbare APIs, die typunabhängig sind und unabhängig vom Schema arbeiten
- **Administration:** Verfügbarkeit allgemeiner Verwaltungsfunktionen und auch für die Erstellung administrativer Ausgaben (Statistiken, Metriken, Software-Leitstände)

Quelle: nach Habermann, H.-J., Leymann, F.: Repository - eine Einführung; Oldenbourg Verlag 1993
und Däberitz, D.: Der Bau von SEU mit NDBMS; Diss. Uni Siegen 1997

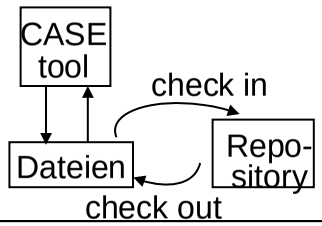
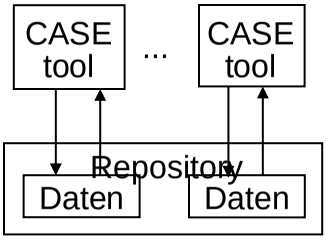
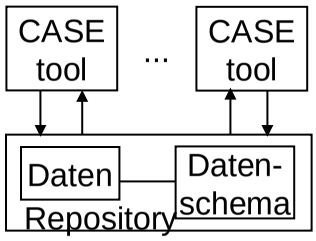
Zielstellungen für Repositories (2)

- Effektive Arbeitsweise:
  - **Verteilung:** Zugriff von verschiedenen Orten aus und Bereitstellung einheitlicher Basisfunktionen für die kooperierende Arbeit mehrerer Benutzer
  - **Transaktionen:** Dienen der Synchronisation von Metadaten und Objektdaten. Sie können dazu benutzt werden, um temporär inkonsistente Zwischenzustände von Dokumenten zu verwalten
  - **Leistungsfähigkeit:** schneller, eff zienten Zugriff auf umfangreiche Dokumente, großer Durchsatz (Transaktionen pro Sekunde)
  - **Konf gurationsmanagement:** Verwaltung unterschiedlicher, zu einem Projekt gehörender Ent-wicklungsstände von Dokumenten
  - **Usability:** Einfache Bedienung durch einheitliche ergonomische Benutzungsoberf ächen
- Entwicklungsqualitäten:
  - **Adaptierbarkeit:** Einbettung des Repository in gewünschte Entwicklungsrichtung (kommerzielle oder technische Informationssysteme, Standardsoftware)
  - **Portabilität:** Zukunftssicherheit - Standardkonformität – Aufwärtskompatibilität

Quelle: nach Habermann, H.-J., Leymann, F.: Repository - eine Einführung; Oldenbourg Verlag 1993
und Däberitz, D.: Der Bau von SEU mit NDBMS; Diss. Uni Siegen 1997

21.2 Architektur von Datenablagen

Repository - Datenintegrationsstufen

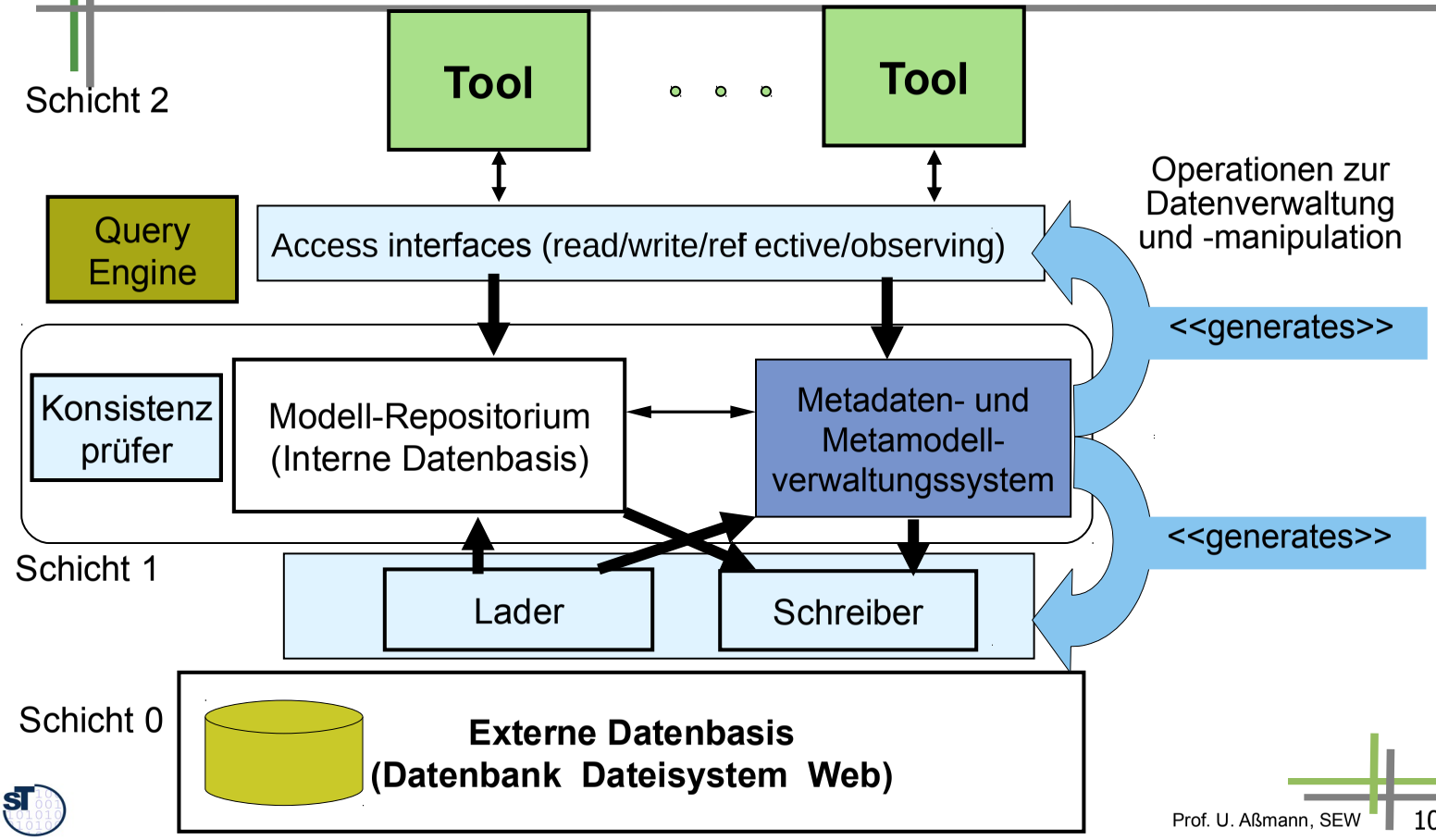
Integrationsart	Schema	Wirkungsweise
Black-box-Integration (schwach)		• Werkzeug arbeitet isoliert auf eigenen Datenstrukturen • Repositorium stellt Daten bereit (check out) • Nach Arbeit Ablage in Repositorium (check in) • checkout/in oft manuell
Grey-box-Integration (mittel)		• Werkzeugzugriffe durch Repository-Dienste ersetzt • Unterstützung von Datenintegrität und Interoperabilität zwischen Werkzeugen • Keine Offenlegung essentieller Bestandteile der Werkzeug-Implementierung • Verkapselung in einem Zugriffsmodul
White-box-Integration Datenteilung (stark)		• Def nition einheitl. Datenschema für alle Werkzeuge • Alle Werkzeuge setzen über Zugriffsdienste auf • Einfache Sicherung von Konsistenz, Datenintegrität und Datensicherheit • Werkzeuge sind bei Änderung an Datenschema anzupassen

Quelle: [Bal-II, S. 608]

Metamodellgesteuerte Repositorien

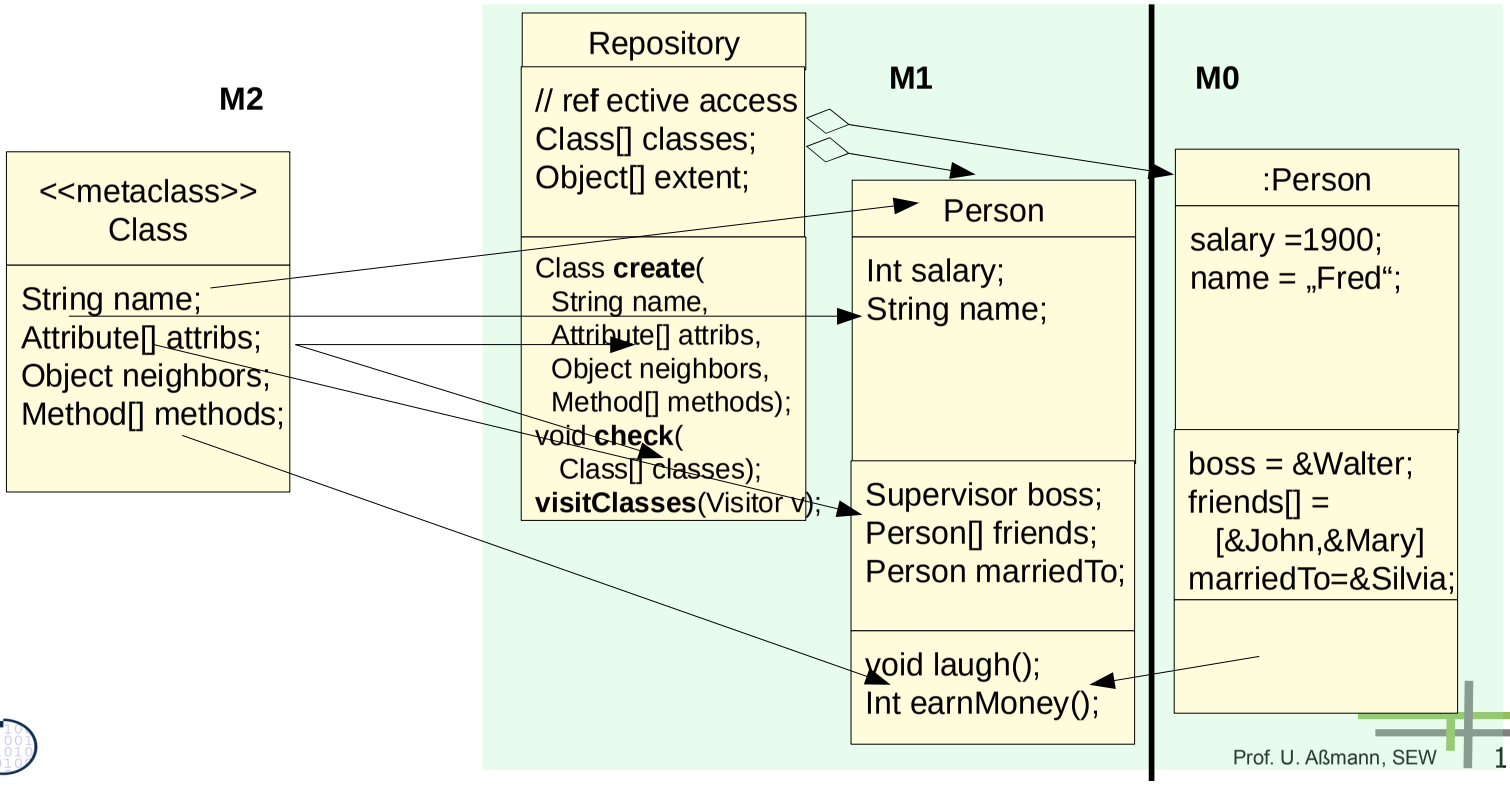
- Ein **metamodellgesteuertes Repositorium** erlaubt es, auch gleichzeitig, verschiedene Metamodelle zu laden
  - Ist zugeschnitten auf eine Metasprache (oder geliftete DDL)
  - Speicherung von Metadaten möglich (metadata repository), aber auch Modellen (model repository)
- Vorteile:
  - **Ableitung von Strukturinformation für Modelle:** Die strukturellen Eingeschaften der Modelle sind durch das Metamodell definiert und bekannt
    - Sind die Collection der Attribute einer Klasse eine Menge, Liste, Bag?
    - Was sind atomare Attribute? Referenzattribute? Mengenattribute?
    - Was sind Knoten und Kantentypen?
    - Kardinalitäten der Nachbarmengen?
  - Aus diesen Informationen können für alle Klassen eines Modells **Zugriffsschnittstellen** generiert werden

Grobarchitektur bei Datenteilung (Metamodellgesteuertes Repositorium)



Generierung von Typisierten Schnittstellen

- Aus den Metaklassen wird die Struktur der Typen der Schnittstellen und vieler Code-Pakete auf M1 abgeleitet:

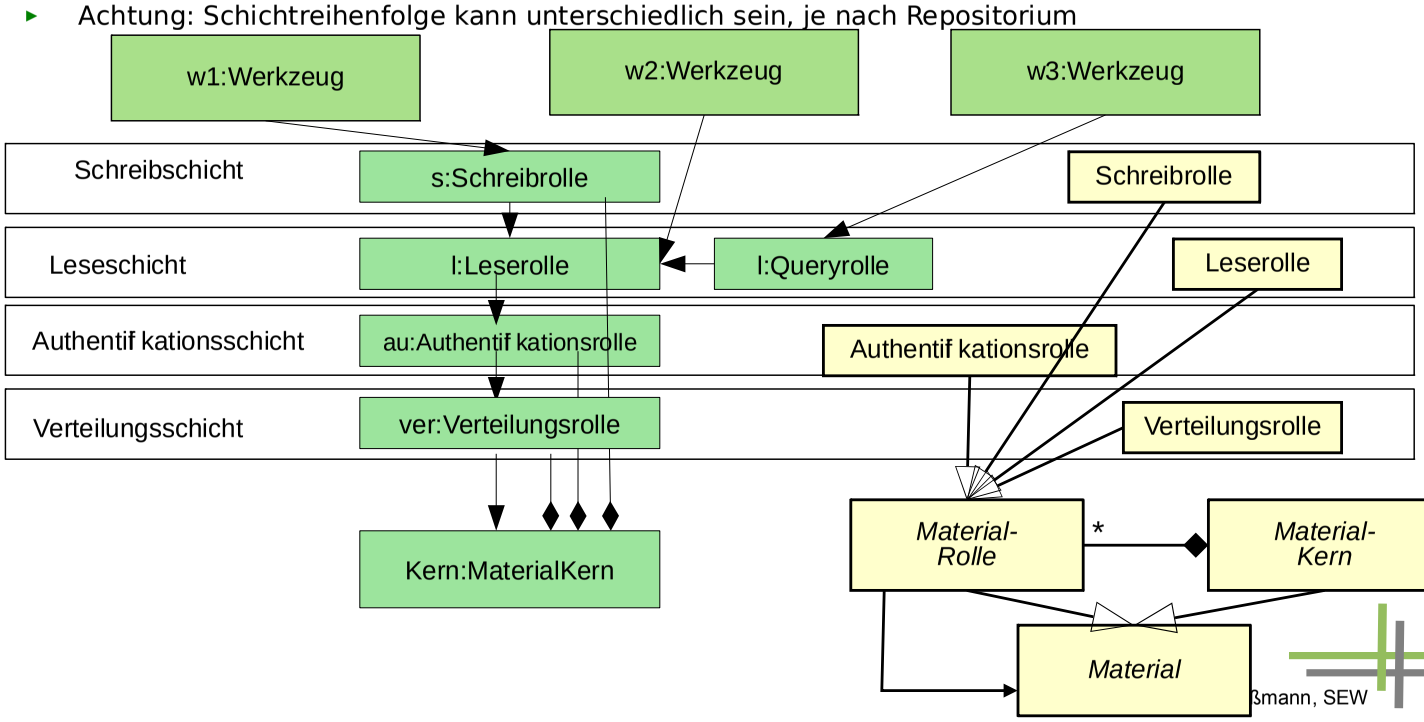


Generierung von Schnittstellen und Paketen in einem metamodellgesteuerten Repositorium

- Generierung von **statisch typisierten Zugriffsschnittstellen** zu den Modellen
  - Lese- und Schreib-Schnittstellen, inkl. deren Implementierung mit Leser/Schreiber-Synchronisations- bzw. Transaktionsprotokollen
  - Typisierung wird von dem Metamodell aus M2 geliefert (spezifziert in DDL)
  - Strukturinformation
- **Queryschnittstellen** zu den Modellen (statisch typisiert)
  - Implementierung mit Query-Interpreters, der die DQL kennt und interpretiert
- Generierung von **Konsistenzprüfern**, um über die Struktur hinaus weitere Konsistenzbedingungen (Wohlgeformtheit, Integrität) zu prüfen (aus DCL-Teil des Metamodells)
- Generierung von **Observer-Schnittstellen**: Mit ihnen wird die Observation der Modelle möglich (durch Einhängen von Observern) (aus DDL)
- Generierung von **Visitor-Paketen**: Mit ihnen wird die Anwendung von verschiedenen Algorithmen auf die Modelle möglich (durch Visitor-Pattern) (aus DDL)
- Generierung von **Datenaustausch-Schnittstellen:** Importern und Exportern, die in Austauschformate wandeln (aus Technikraumbrücken und Datenverbindungs-Abbildungen auf M2)
  - Implementierungen von persistenten Objekten (activation, passivation)
- Bereitstellung von **reflektiven Zugriffsschnittstellen** zu den Modellen (schwache Typisierung)
  - Zugriff auf die Knoten und Kanten als Graphknoten und Graphkanten (untypisiert)
  - Zugriff auf *Extent eines Modells:* Zugriff auf alle Modelle eines Metamodells oder alle Objekte eines Modells
  - aus der Metasprache oder gelifteter DDL abgeleitet (aus dem Graphschema)

Kollaboration von Werkzeugen auf dem Repositorium mit Rollenobjekten

- Ein gutes Entwurfsmuster für Repositorien und ihre Zugriffsschichten bietet das geschichtete *Role Object Pattern* von Bäumer et.al. (Siehe Kapitel in Design Patterns and Frameworks)
- Dekorator-Ketten regeln den Zugriff über Schichten
- Viele Schichten, vielen Rollen-Klassen können generiert sein
- Achtung: Schichtreihenfolge kann unterschiedlich sein, je nach Repositorium



21.3 Beispiele für Datenablagen

Historische Beispiele für Repositories

Bezeichnung	Kurzcharakteristik
IBM Repository Manager	Repository für AD/Cycle, offene Architektur für leichten Anschluss von Tools, teamorientiert
PCTE Object Management System	innerhalb einer PCTE-Umgebung
Pirol	Sichtenbasiertes Repository der TU Berlin www.objectteams.org/publications/Diplom_Florian_Hacker.pdf

Beispiele für Firmenweite Repositories

Bezeichnung	Kurzcharakteristik
WebSphere Repository Database von IBM	Administration-Tools zur Installation, Überwachung und Pf ege von Datenquellen und Enterprise Applications
SAP R/3-System	Eigentlich R/3-Data-Dictionary für Ablage von Metadaten auf Basis tabellarischer Datenstrukturen. DD-Tabellen ursächlich für Speicherung kommerzieller Daten aber auch für R/3-Entwicklungsumgebung
SAP NetWeaver Master Data Management (MDM)	Verteilte Stammdatenverwaltung zur Pf ege, Konsolidierung und Harmonisierung integrierter Informationen über gültige Stammdatenattribute

Quelle: Habermann, H.-J., Leymann, F.: Repository - eine Einführung; Oldenbourg Verlag 1993

Beispiele für metadatengesteuerte Repositories

Bezeichnung	Kurzcharakteristik
Hibernate	Persistente einzelne Anwendungen mit object-relational mapper (ORM); Abbildung von Java-Objekten auf Relationen, analog zu ERD-Abbildung, Verwendung von verschiedenen SQL-Datenbanken
Netbeans Metadata Repository (MDR)	Metasprache MOF; Laden von Metamodellen möglich; Generierung von Modell-Zugriffsschnittstellen; reflektiver Zugriff auf Modelle; Mapping auf Filesystem möglich
Eclipse EMF	Metasprache EMOF; ansonsten wie oben
ModelBus	Repository für MOF-basierte Modelle

Beispiele Dateisystem-basierter Datenablagen

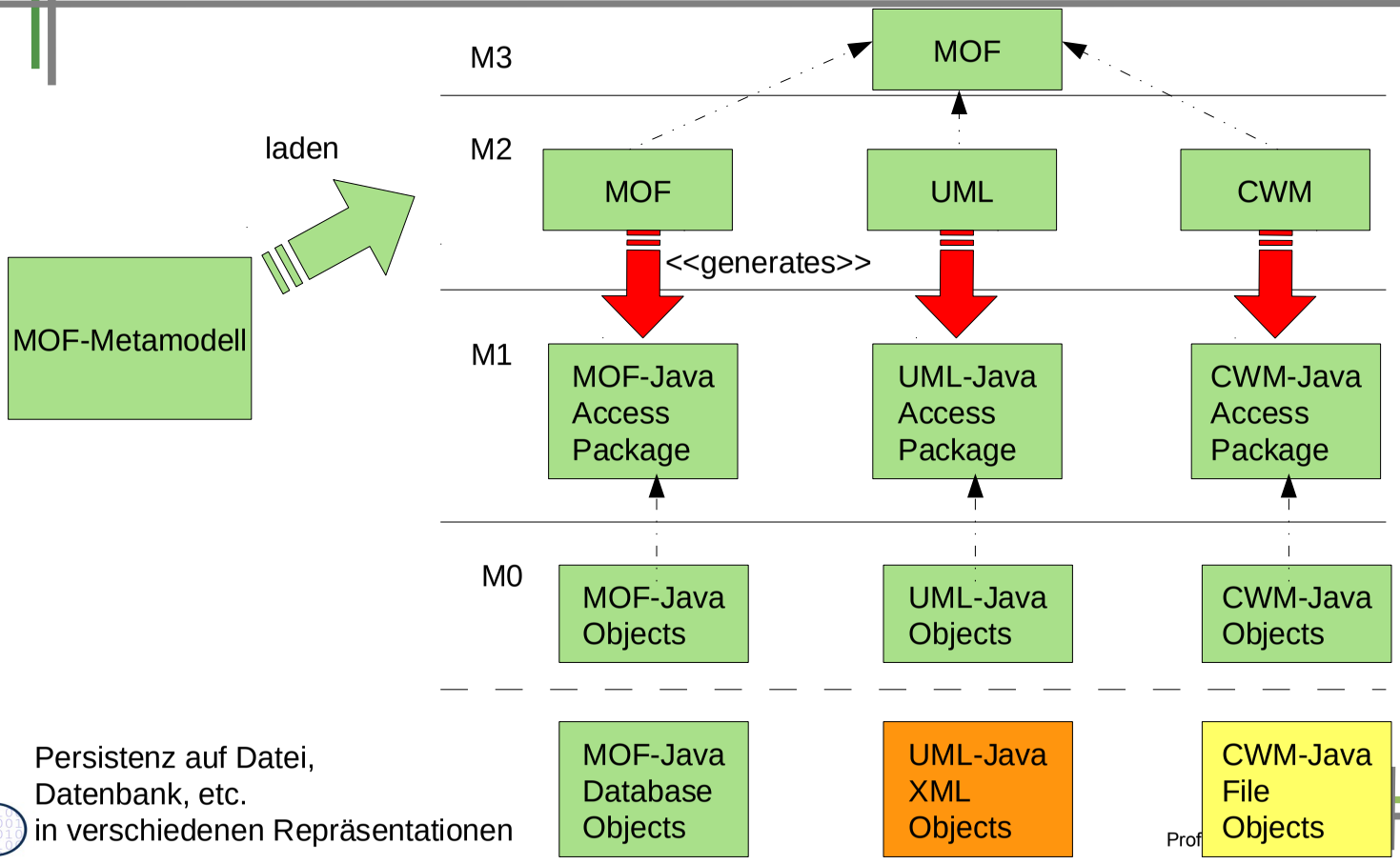
Bezeichnung	Kurzcharakteristik
Microsoft Sharepoint	Webbasiertes, filesystem-basiertes Repository
WebDAV	Webprotokoll zum verteilten Datenmanagement
Subversion/CVS/git	Versionsverwaltungssysteme auf File-Basis; Verwaltung von Versionen aller Files und Verzeichnisse; über spezielle Clients vom Browser aus bedienbar
DropBox, GoogleDocs	Cloud-basiertes Filesystem mit Rechteverwaltung

- Keine Metamodelle, sondern nur einfache Metadaten (markup tags)

Netbeans MDR

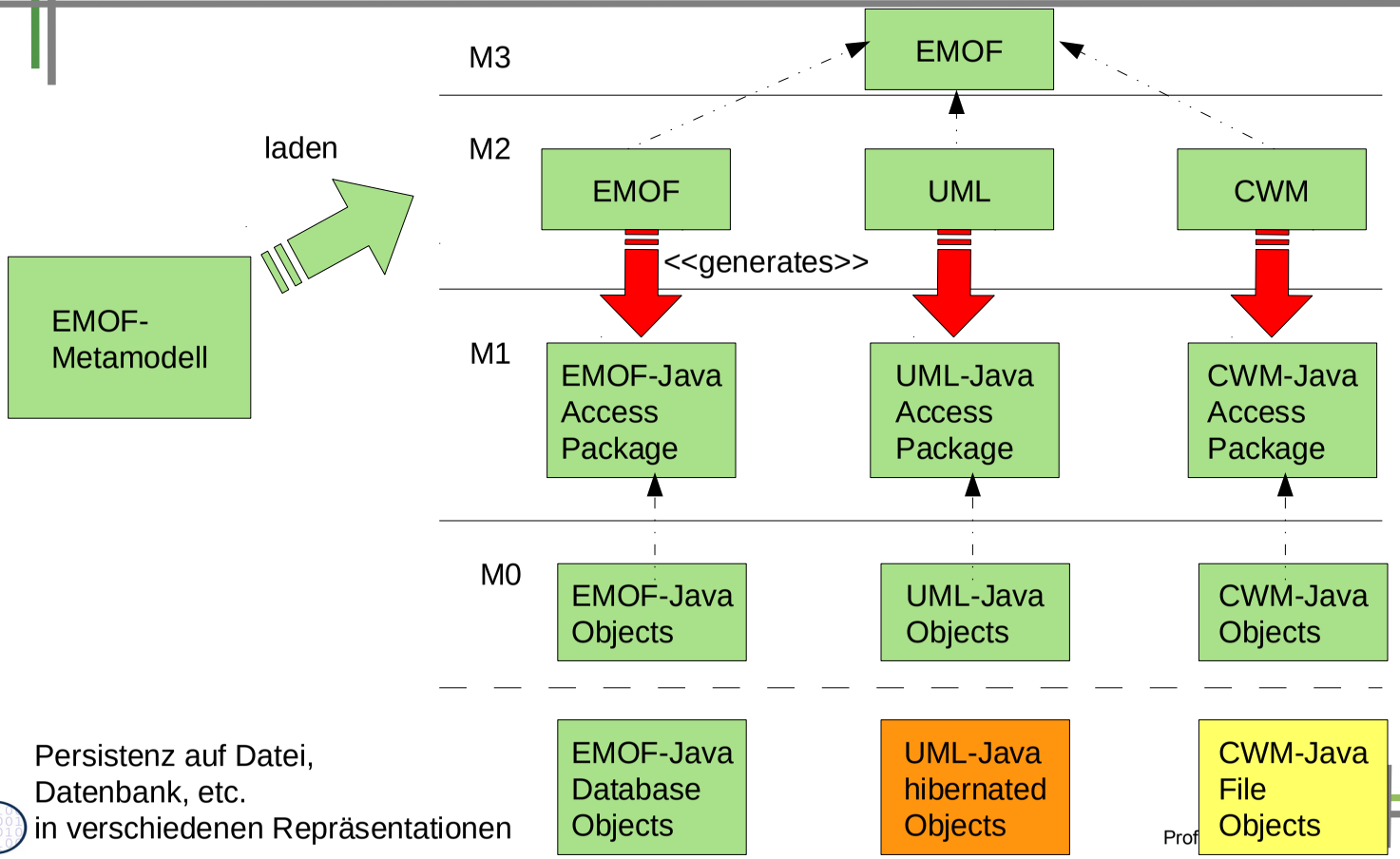
- Ein metamodellgesteuertes Repositorium für die Netbeans SEU http://www.netbeans.org, Metasprache MOF
  - Speicherung von MOF-Metamodellen und -Metadaten möglich (metadata repository), aber auch Modellen (model repository)
- Vorteile:
  - Generierung von Java-Zugriffsschnittstellen zu den Modellen (starke Typisierung), via JMI (MOF-Java-Mapping)
  - Verwendung von reflektiven Zugriffsschnittstellen zu den Modellen (schwache Typisierung)
    - Via Reflektion auf MOF-Konzepten (Klassen, Attributen, Assoziationen, Vererbung)
    - Zugriff auf *Extent*
  - Observation der Modelle möglich
    - Zugriffsschnittstellen können observiert werden
  - Transparente Speicherung im Hauptspeicher, Filesystem oder in einer Datenbank
  - Austauschformat XMI (MOF-XML-Mapping)

Netbeans MDR



Persistenz auf Datei, Datenbank, etc. in verschiedenen Repräsentationen

Eclipse EMF (basierend auf EMOF)



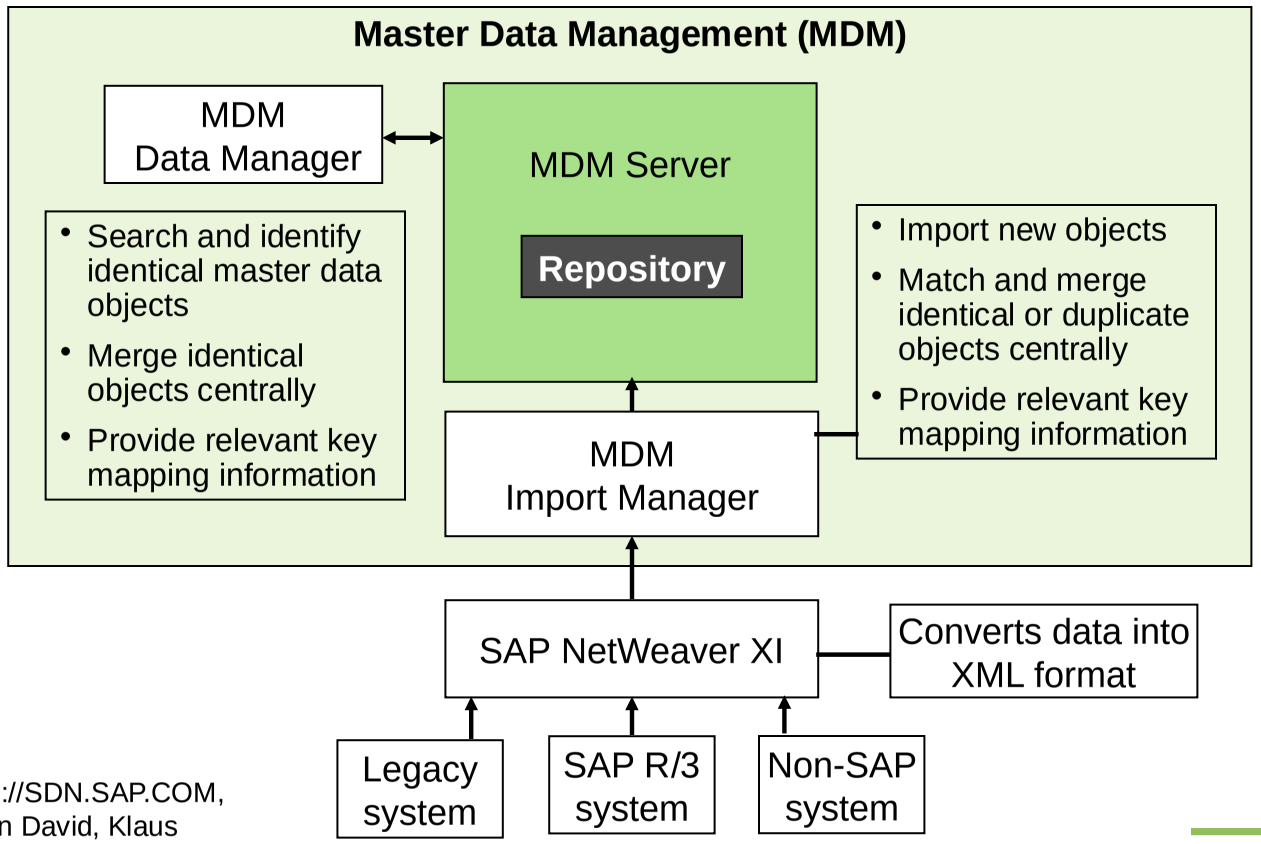
Persistenz auf Datei,
Datenbank, etc.
in verschiedenen Repräsentationen

21.3.2 Master Data Management (MDM)

MDM

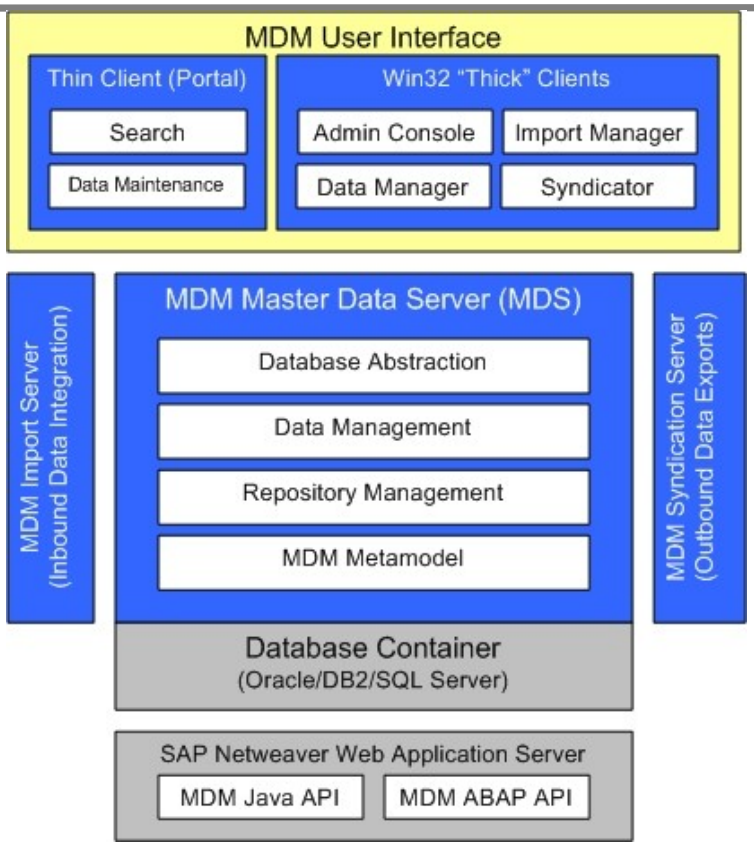
- Verwaltet alle Daten eines Unternehmens in einer Datenablage
  - Föderierte verteilte Datenbank
  - Nicht konsistent gehalten durch Transaktionen
  - Allerdings mit Werkzeugen zur Analyse, Query, Konsistenzprüfung, -erhaltung und -wiederherstellung, Normalisierung
  - http://en.wikipedia.org/wiki/Master_Data_Management
  - Entsteht oft aus Firmenfusionen und muss danach “entschlackt” werden

Bsp.: SAP NetWeaver MDM



Quelle:
URL: http://SDN.SAP.COM,
Artikel von David, Klaus

http://searchsap.techtarget.com/resources/SAP-MDM-software



http://searchsap.techtarget.com/generic/0,295582,sid21_gci1232140,00.html

21.4 Datenverbindung mit Austauschformaten und Technikraum-Brücken

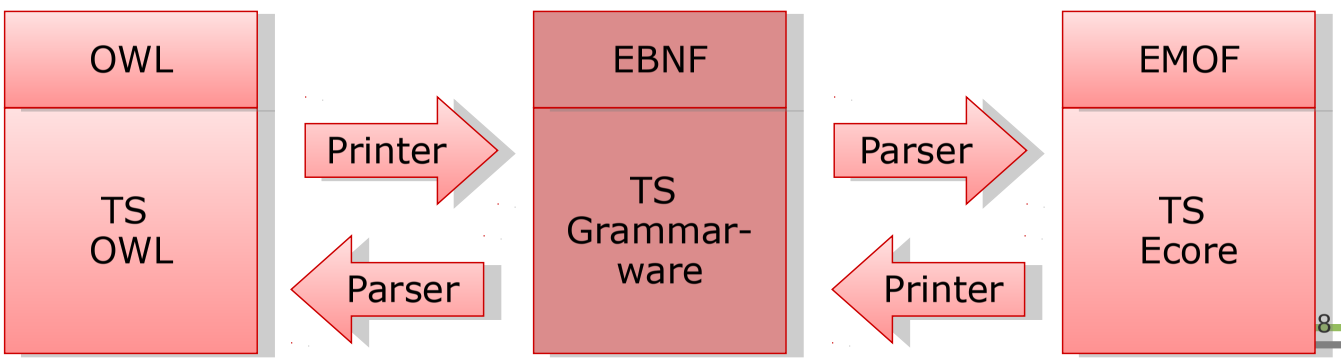
Einsatz in Transformations-Brücken zwischen Repositorien

Austauschformat konkrete Syntax

- **Datenverbindung** zwischen Repositorien beruht auf einer *semantischen Beziehung der Daten*
  - Z.B. Gleiche Sprache auf M2 ermöglicht Mapping zwischen Modellen auf M1
  - Sprach-Abbildung auf M2
- Für Datenverbindung ist als Austauschformat von jeher konkrete textuelle Syntax benutzt worden (mittels Technikraum Grammarware, mit Metasprache EBNF)
  - Parser lesen den Text und wandeln ihn in das interne Format
  - Prettyprinter schreiben das interne Format um auf die konkrete Syntax
- Man spricht von **normativer konkreter Syntax**, wenn diese normiert ist, also nicht beliebig

Transformative TS-Brücken mit konkreter Syntax

- Eine **transformative Technikraum-Brücke** (TS-Brücke, TS bridge) bietet
  - ein Austauschformat in konkreter Syntax (via dem TS Grammarware)
  - eine Generierungstechnik für Printer und Parser
- Am besten: normative konkrete Syntax
  - EMFText: normative konkrete Syntax for Ecore
  - Xtext: normative konkrete Syntax for Ecore and OAW
  - CDIF: normative konkrete Syntax für ERD



Austauschformat CDIF: CASE Data Interchange Format

- CDIF ist ein Austauschformat für CASE-Werkzeuge, basierend auf
  - Metasprache auf M3: ERD
- Austauschformat einfachen, transparenten Aufbaus zwischen (Modell-)Tools und Repositories
  - hersteller- und methodenunabhängig
  - unterstützt die Kooperation zwischen verschiedenen Tools und Projekten

Quellen: OMG-Dokument ad/98-07-09; http://www.cdif.org

Textuelle CDIF-Beschreibung eines DFD

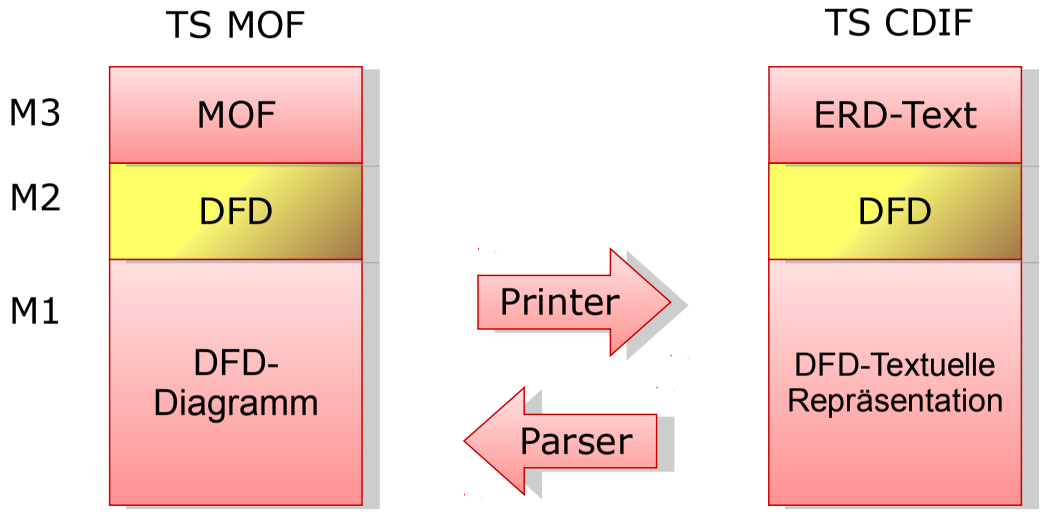
CDIF nutzt zur Spezif kation von Grammatiken nicht EBNF, sondern eine textuelle Notation von ERD (ERD-Text).

DFD-Spezif kation in ERD-Text (Schlüsselwörter in boldface, benutzte Nichtterminale in italics):

```
obj_dfd          (dataFlowDiagram dfd_title {dfd_element} )
dfd_title        (dfdTitle @dfd_title_id dfd_title_name )...
dfd_element      dfd_bubble |dfd_store | dfd_term | dfd_tb |
                            dfd_csc | dfd_flow
dfd_bubble       (process  pt pt @process_id process_name
                           inst_num [process_type] )
@process_id      (processID string )
process_name     (processName string )
process_type     (processType string )
dfd_store        (store pt pt store_name inst_num )
store_name       (storeName string )
```

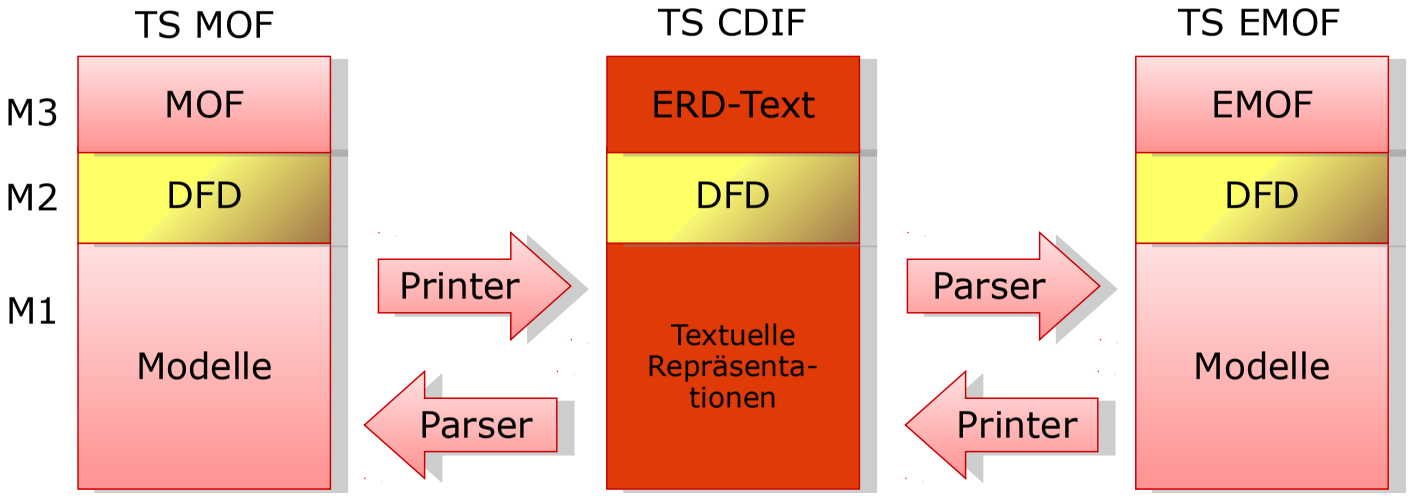
Austauschsyntax CDIF

- CDIF definiert eine textuelle Syntax für ERD (ERD-Text)
  - normative textuelle Repräsentation (für alle Sprachen auf M2 gleich)



Transformative TS-Brücken via CDIF

- TS-Brücken (via CDIF) generieren Parser und Printer
- Normative konkrete Syntax in der CDIF-Spezifikation festgelegt

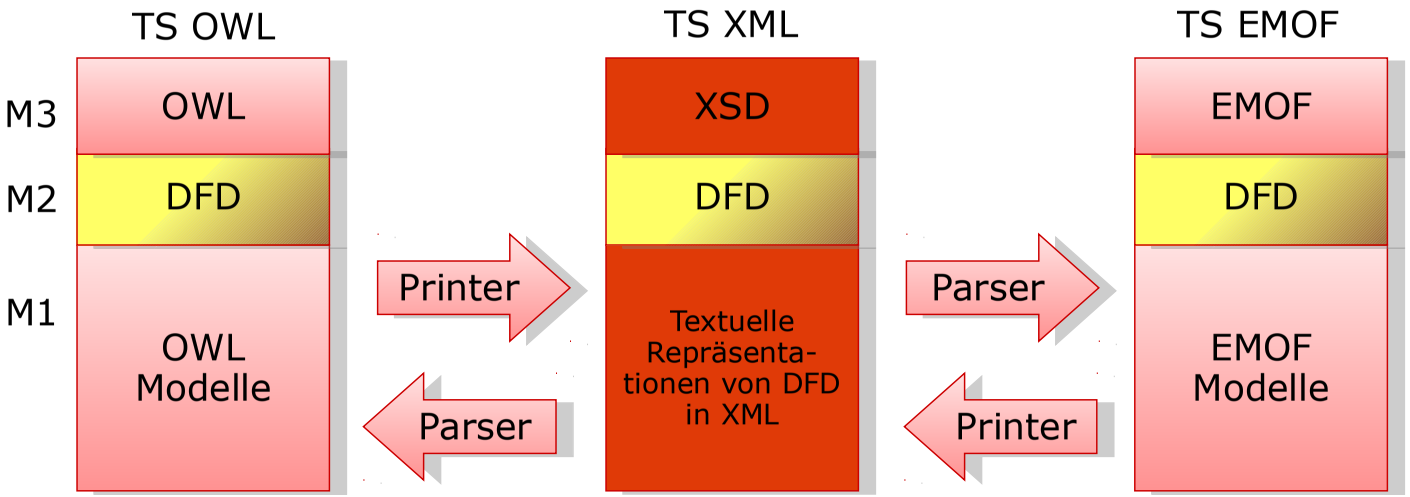


Austauschformat XMI
XML Metadata Interchange Format

- Ziel: anbieterneutrales offenes Austauschformat für Metadaten/Modelle in verteilten Umgebungen
  - Metasprache MOF
  - generisches „Stream“-Format
  - lose gekoppelte Architektur, einfach für Anbieter zur Verarbeitung in aktuellen Produkten
  - überwindet Lücken zwischen inkompatiblen Tools, Repositories und Anwendungen
- Stand: OMG-Standard für XML Metadata Interchange (XMI) zwischen Repositories, Tools und Anwendungen Version 2.1 (formal/2005-09-01)
- Allerdings:
  - Wegen des Indeterminismus des spannenden Baumes keine volle Kompatibilität möglich

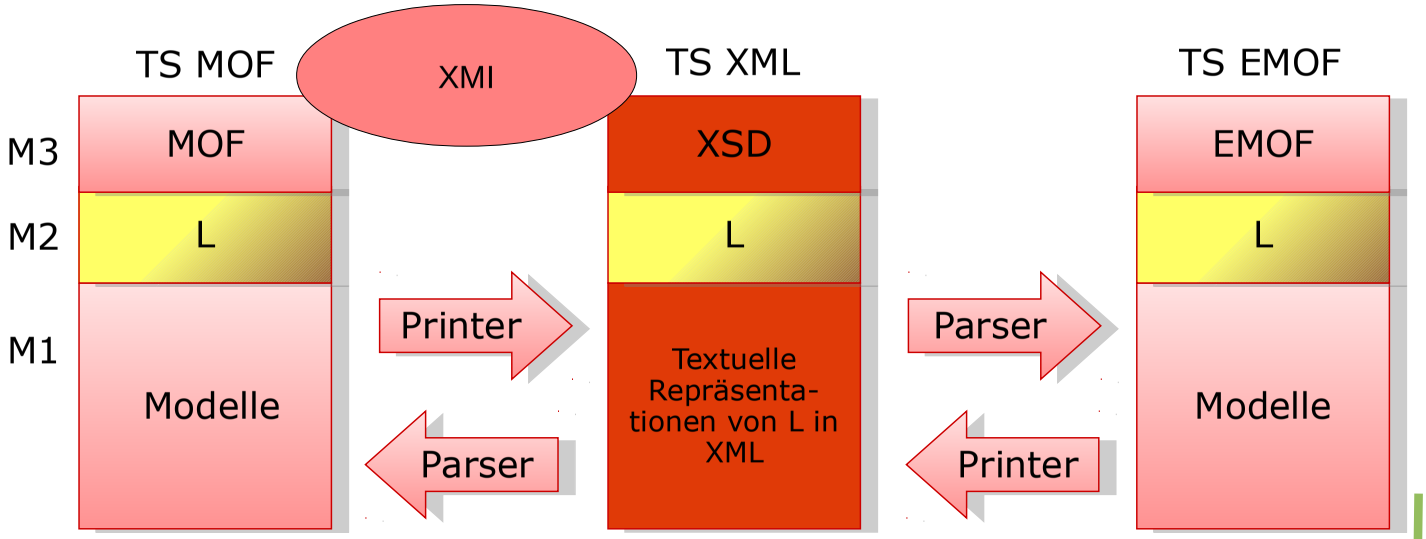
Transformative TS-Bridges via XML

- XML is a normalized concrete syntax
  - Because of trees, a linearized normalized concrete syntax is possible
- Good for exchange!



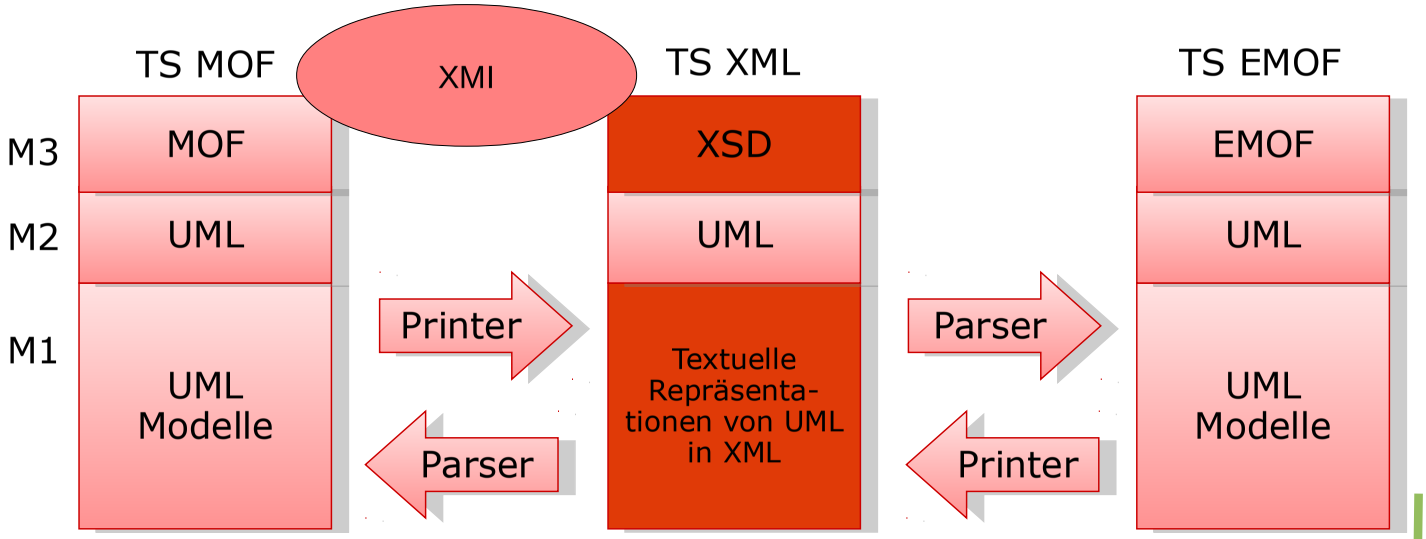
XMI: Transformative TS-Brücke

Im allgemeinen Sinne ist XMI eine Brücke zwischen MOF und anderen TS via XSD/XML



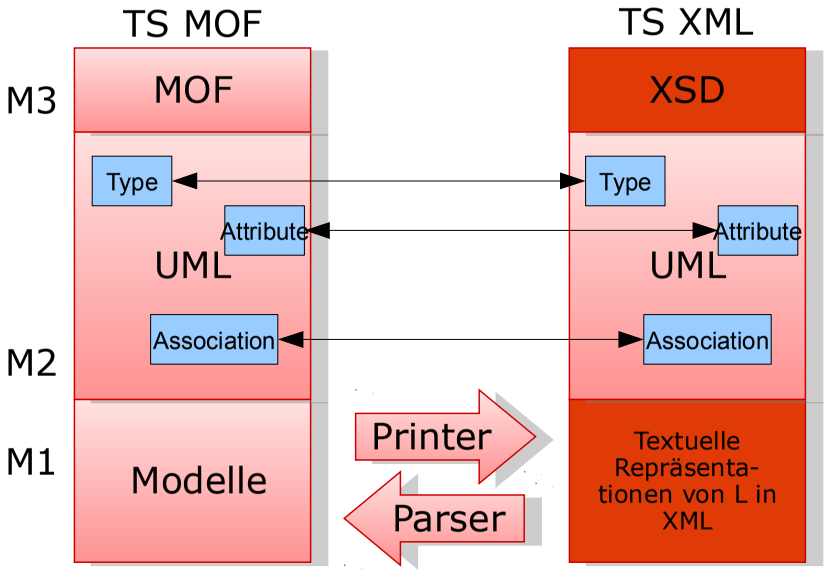
XMI: Transformative TS-Brücke für UML

Meist wird allerdings das nur für UML ausgeprägt. Dann ist XMI eine UML-Brücke



Zusammenhang XMI - UML

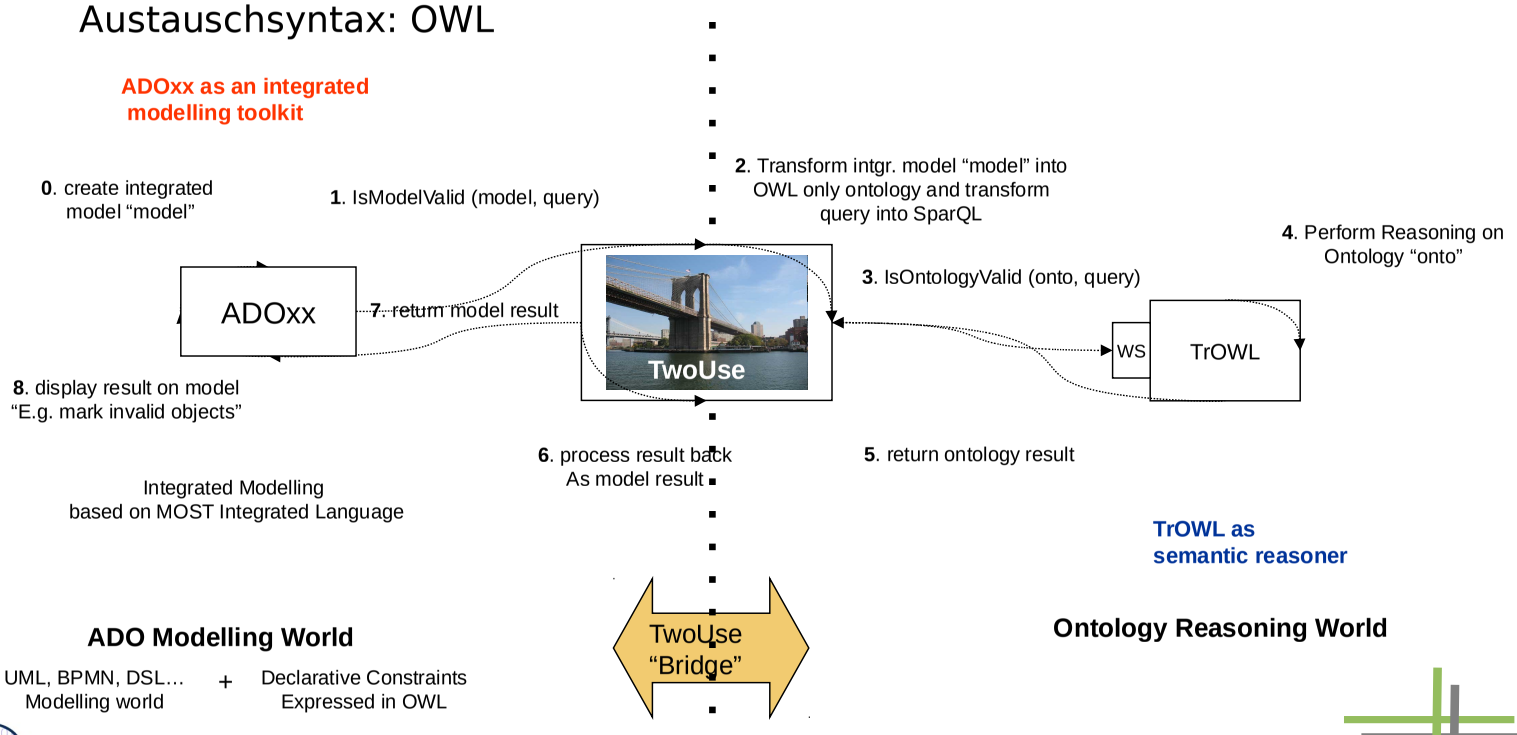
- XMI liegt ein Metamodell der UML zugrunde, das zweimal, in MOF und XSD, spezifiziert wird
  - Zwischen beiden Metamodellen wird ein **Sprachabbildung (language mapping)** angegeben
  - Aus diesem werden Printer und Parser generiert



Transformationsbrücke zwischen ADO und OWL

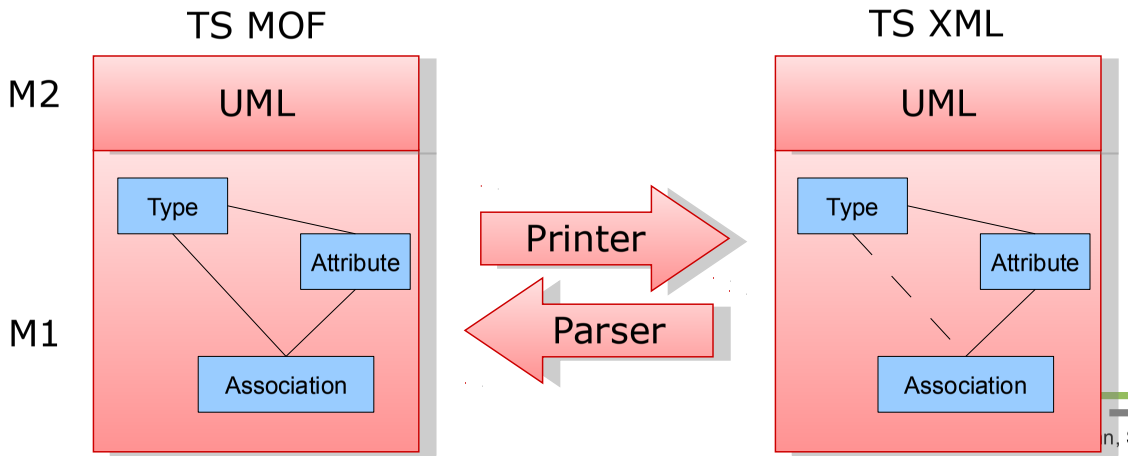
TwoUse (U Koblenz) ist eine Transformationsbrücke zwischen TS ADO (BOC Wien) und TrOWL (OWL, Uni Aberdeen)

Austauschsyntax: OWL



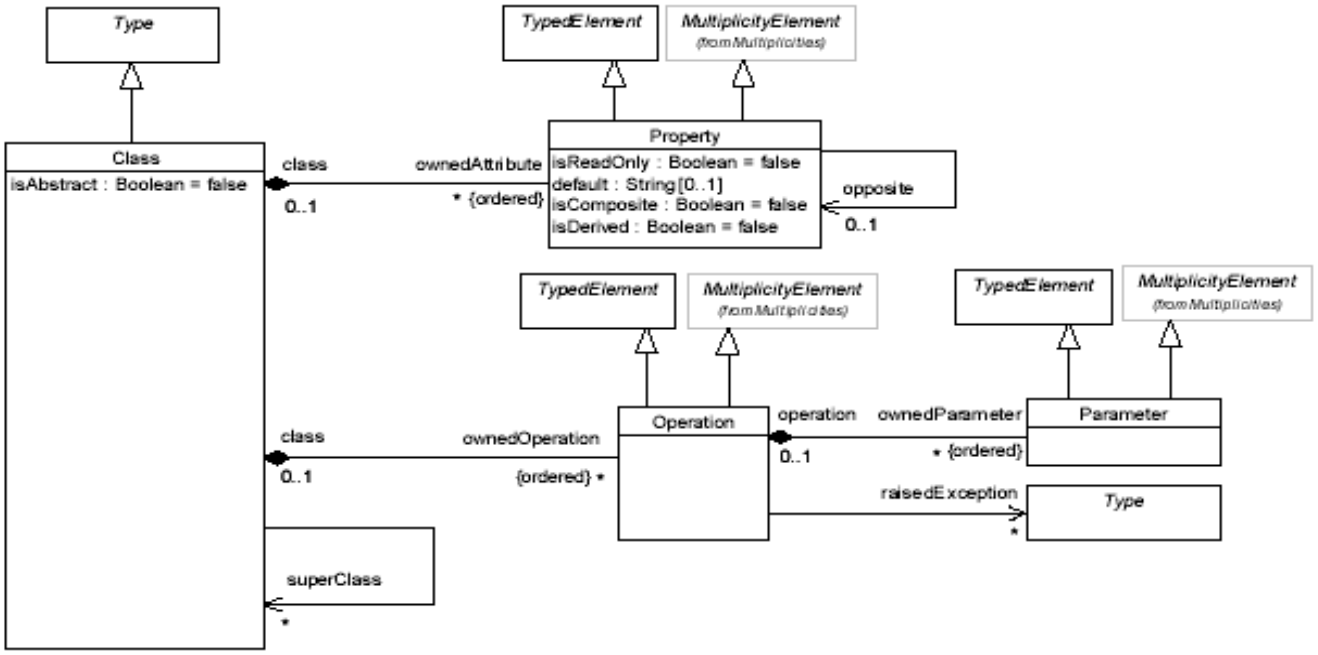
Problem: normativer spannender Baum für Graphmodelle

- UML bzw. MOF sind graphbasiert, XML baumbasiert
- XML muss bestimmte Links in Namensreferenzen auflösen
  - Dazu wird über UML oder MOF-Modell ein Spannender Baum gelegt (z.B. entlang der Aggregation)
  - Alle Links, die nicht im spannenden Baum vorkommen, werden mit Namensreferenzen dargestellt, und **nicht** im XML Baum
- Da der spannende Baum nicht deterministisch ist, entstehen Inkompatibilitäten



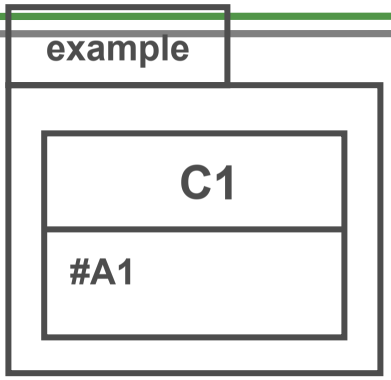
Erinnerung: Classes and Properties in UML

Definieren einer Klasse als Typ und Festlegung der weiteren Elemente zur klassenbasierten Modellierung



Quelle: UML 2.0 Infrastructure Specification; OMG Adopted Specification ptc/03-09-15

Beispiel einer XMI-Objektinstanz: Kodierung einer UML-Klasse

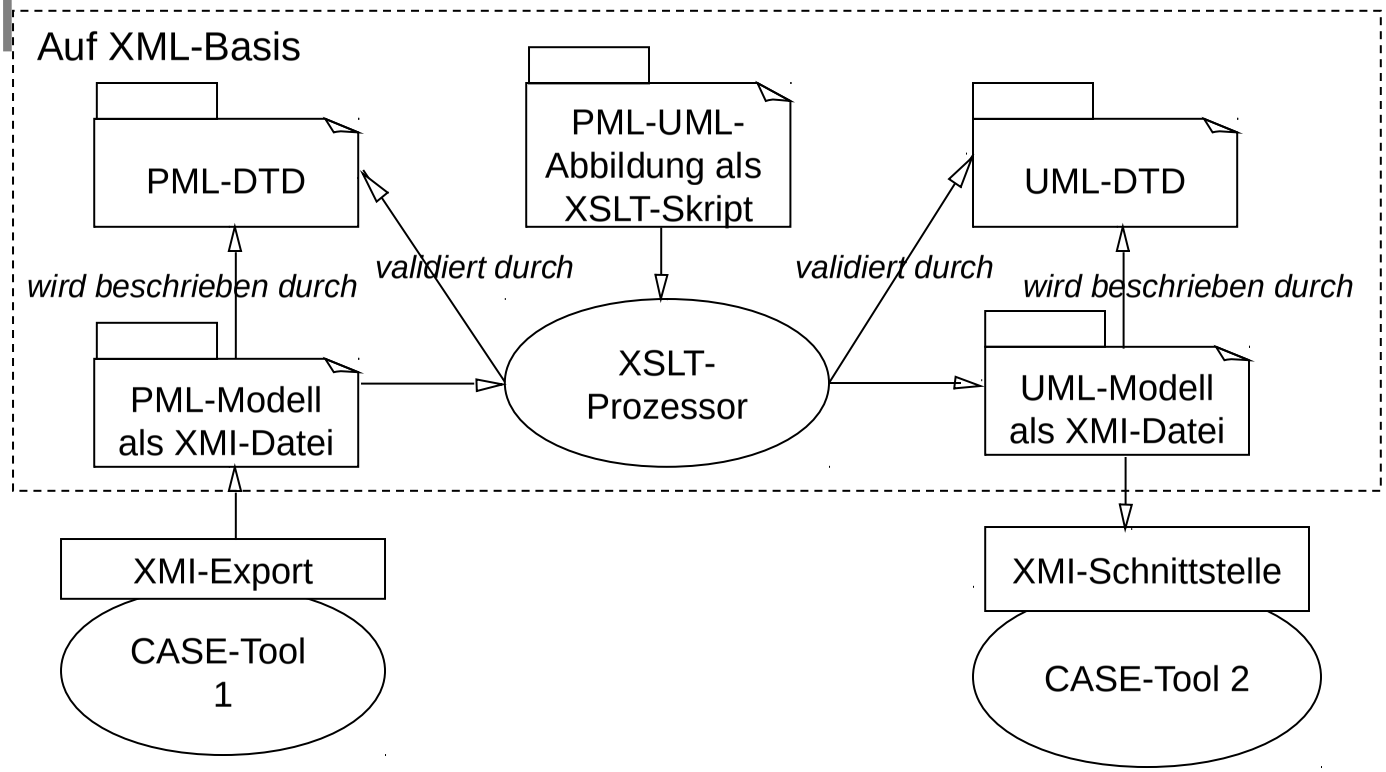



```
<?xml version = „2.0"?>
<!DOCTYPE XMI SYSTEM "uml.dtd">
<XMI xmi.version=„2.0">
<XMI.Header>
      <XMI.Metamodel name=„UML“ href=„UML.xml“/>
      <XMI.Model name=„example“ href=„example.xml“/>
</XMI.Header>
<XMI.Content>  <Core.Basic.NamedElement.name>example</Core.Basic.NamedElement.name>
   <Core.Basic.Class>
    <Core.Basic.NamedElement.name>C1</Core.Basic.NamedElement.name>
      <Core.Basic.feature>
        <Core.Basic.Property>
          <Core.Basic.NamedElement.name>A1</Core.Basic.NamedElement.name>
          <Core.Sasic.NamedElement.visibility xmi.value=“protected"/>
        </Core.Basic.Property>
        [<Core.Basic.Operation> ... </Core.Basic.Operation>]
      </Core.Basic.feature>
   </Core.Basic.Class>
</XMI.Content>
</XMI>
```

UML Typen

(ähnliches Beispiel siehe : www.jeckle.de/xmi_ex1.htm)

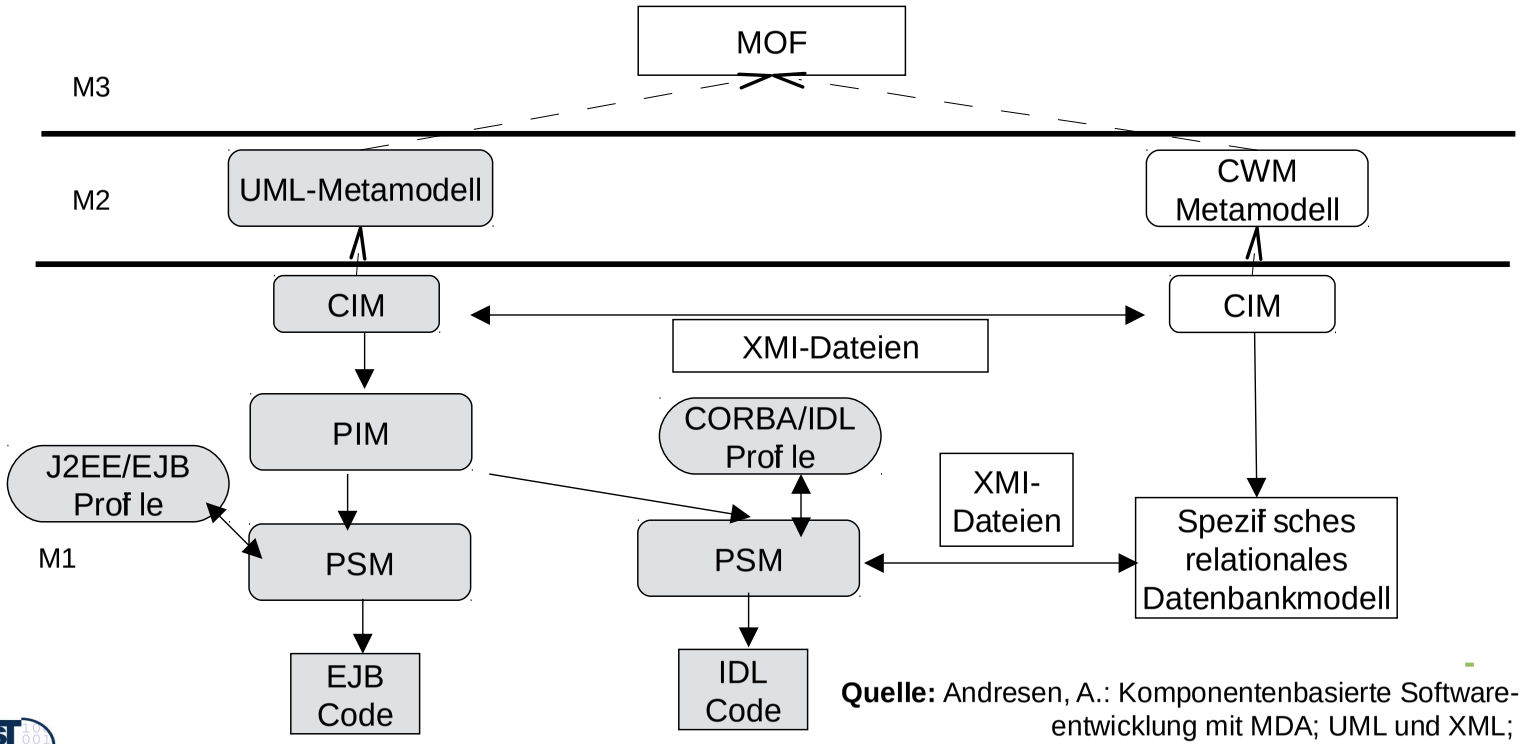
Bsp.: Adaption von XMI-basierten Modellen mit XML-DML wie XSLT



Quelle: Großmann, A.: XMI für prozedurale Programmstrukturen und Transformation in UML; Diplomarbeit an der Fakultät Informatik der TU Dresden, 2000

Bsp.: Datenaustausch mit XMI für CIM im Kontext von Model-Driven Architecture (MDA)

Computation independent model (CIM) ist eine Requirements-Spezifikation



Quelle: Andresen, A.: Komponentenbasierte Softwareentwicklung mit MDA; UML und XML; Hanser Verlag 2003, S. 251

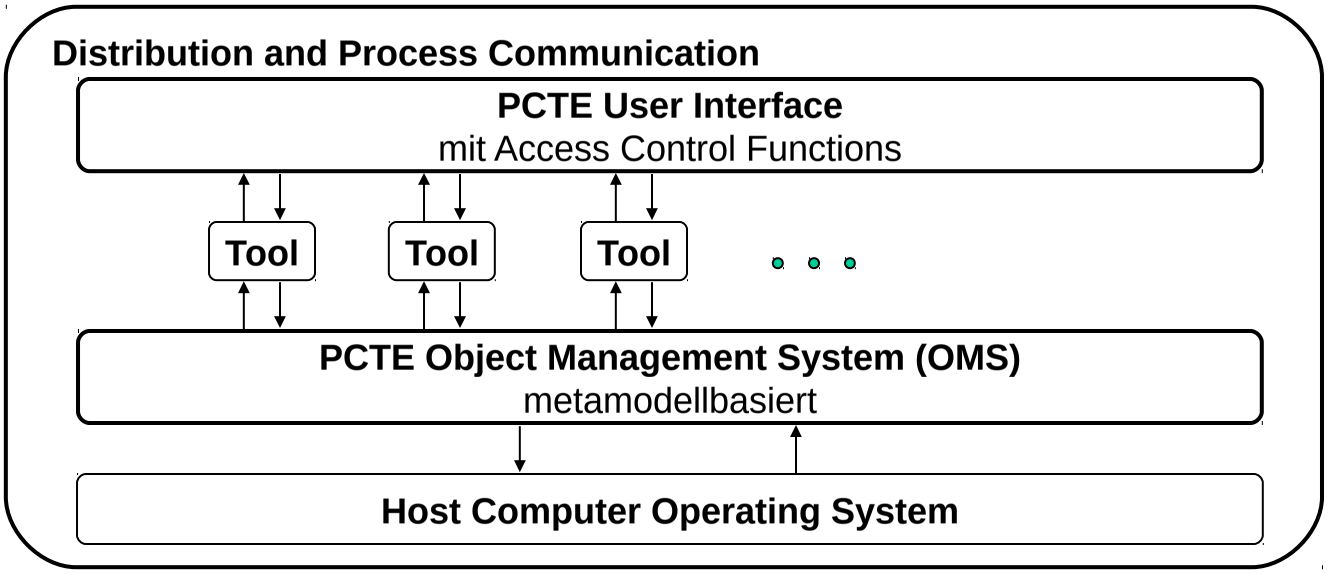
21.5 Ein Framework und Technikraum zur Werkzeugintegration (PCTE)

http://ieeexplore.ieee.org/iel3/2107/7595/00313508.pdf?arnumber=313508

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.8315&rep=rep1&type=pdf

Portable Common Tool Environment (PCTE+, HPCTE)

- PCTE erfüllt den Schnittstellenstandard der ECMA - unterstützt systemunabhängigen Zugriff auf Werkzeuge und Repository



Quelle: ECMA - Portable Common Tool Environment (PCTE), Abstract Specif cation; ECMA-149, 2nd Edition, Juni 1993

Technische Merkmale von PCTE

PCTE stellt eine Menge hochintegrierter **Basisdienste** bereit, die eine vielseitige Grundlage für verteilte Software-Entwicklungsumgebungen (SEU) bilden. Hauptmerkmale sind:

- **verteiltes DBMS** basierend auf dem ERD mit Erweiterungen, wie zusammengesetzte Objekte, Versionen, Mehrfachvererbung, dynamisch kreierte Sichten, eingebettete Transaktionen usw.;
- ein **exklusives Ausführungssystem**, welches Prozeßhierarchien, Vererbung von offenen Files, Prozeßkommunikation über Pipes und Nachrichtenwarteschlangen gestattet. Werkzeuge können als Shell-Skript geschrieben und in mehreren Fenstern unterstützt werden.
- **verteilte Dienste**, d.h. Objektbasis und Prozesse sind transparent verteilt, Replikation von Objekten sowie Schema-Management sind ebenfalls dezentralisiert.
- **erweiterbare Sicherheitsmerkmale**, wie Vertraulichkeit, geschützte Zugriffssteuerung für individuelle Objekte, Revisionsfähigkeit.

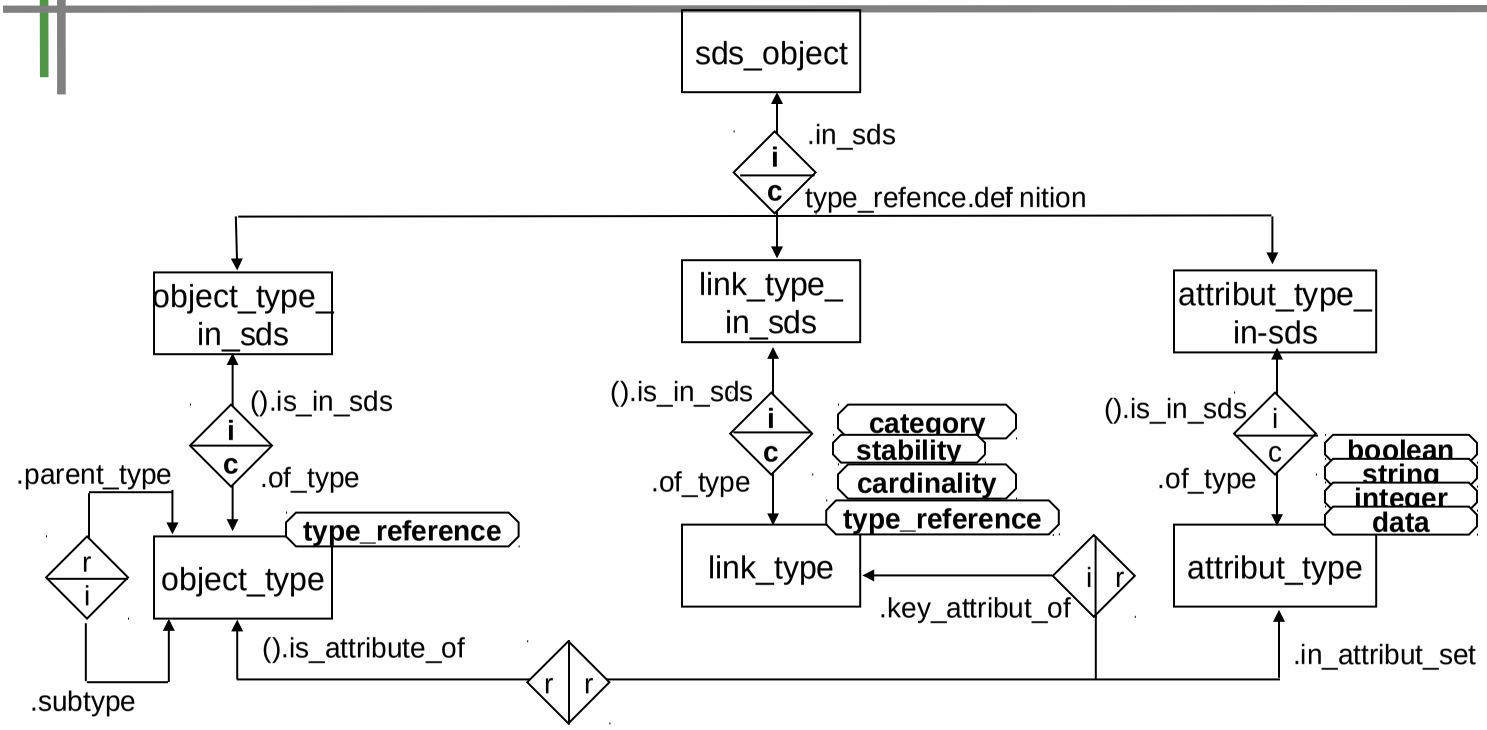
Quelle: http://pi.informatik.uni-siegen.de/pi/hpcte/hpcte.html

PCTE-OMS-Modell (Object Management System)

- Das OMS stellt Datentyp- und Datenspeichermöglichkeiten sowie Concurrency-Control-Mechanismen zur Verfügung,
  - def niert statische Informationen, die in der ***object base*** (Repositorium aller persistenten Daten) gehalten werden,
  - liefert Konzepte für die Softwareentwicklung, wie beispielsweise die (Typ-)Vererbung der objektorientierten Modelle.
- Das OMS ist metamodell-gesteuert. Es enthält **Typdef nitionen**, die in ***Schema Def nition Sets (SDS, Metamodelle)*** beschrieben werden:
  - **Objekte:** Entitäten, auf denen Operationen der Werkzeuge ausgeführt werden. Instanzen können Dokumente, Textf les, Quell- oder Objektcode, Task aber auch Geräte und Nutzer sein.
  - **Links (Assoziationen):** gerichtete Beziehungen zwischen (Ursprungs-) und (Ziel-)Objekt (bidirektional)
  - **Attribute:** beschreiben Objekte und Links näher. Sie enthalten einen bestimmten Wertetyp und können sowohl Schlüssel- als auch normales Attribut sein.
- Die DDL **PCTE-OMS** ist **selbstreferenzierend** (in sich selbst spezif ziert, DDL ist geliftet als Metasprache). Damit sind Typen, Prozesse, Teile der Objektbasis, Nutzer und Nutzergruppen werden einheitlich repräsentiert als Objekte.

Quelle: http://gille.loria.fr:7000/Emeraude/emeraude.html

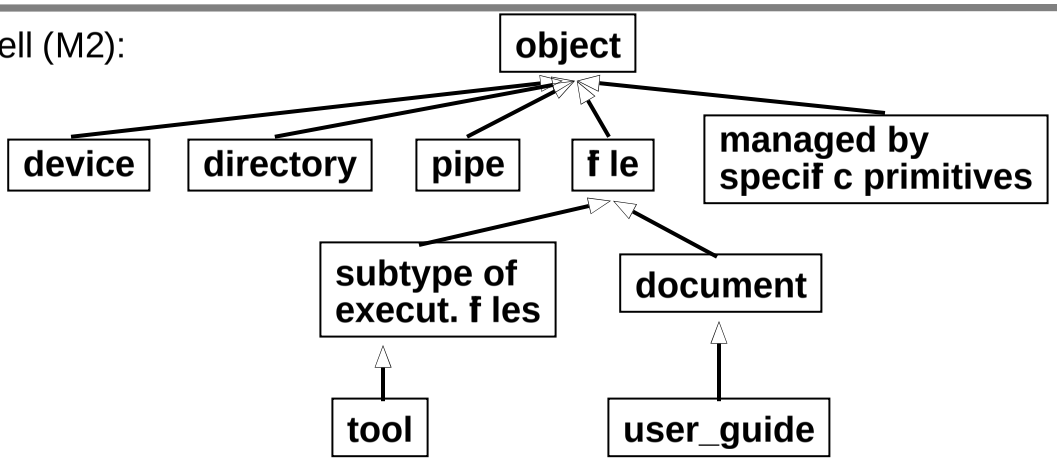
PCTE-OMS Metasprache (Erweiterung von ERD)



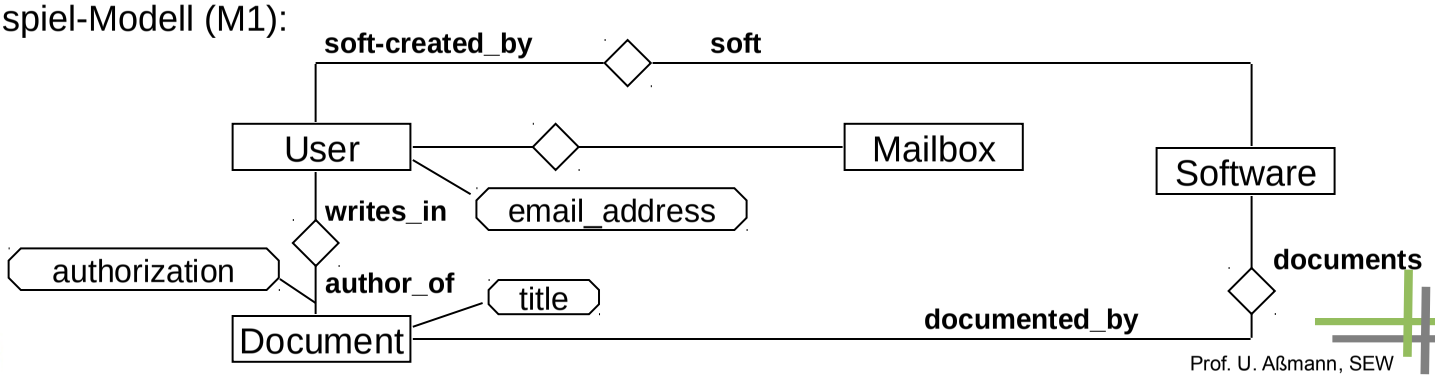
Link-Typ Kategorien: ***c composition*** **bei Anlegen eines neuen Objekts in Referenz zum existierenden.**
r reference **zwischen Ursprungs- und Zielobjekt**
i implicit **kann z.B. reverse link für einen angelegten Link sein**
s system implicit, **automatisch vom System (PCTE-OMS) gesetzt.**

PCTE-Objekt-Strukturen mit erweitertem ERD

PCTE DDL Metamodell (M2):

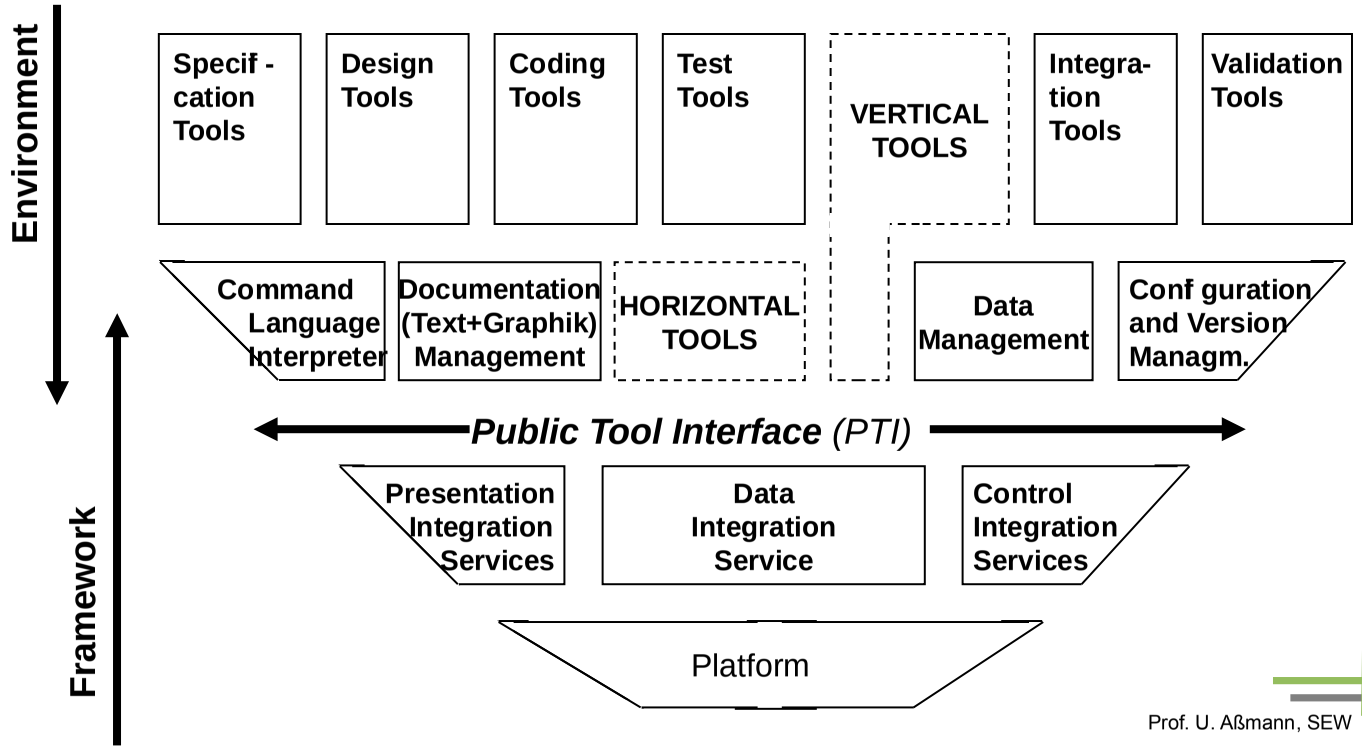


Beispiel-Modell (M1):



Emeraude PCTE Framework

http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=182066
http://www.springerlink.com/content/g111t41326211512/



More PCTE Implementations

- PACT PCTE Implementation
  - Thomas, Ian. Tool integration in the pact environment. In Proceedings of the 11th International Conference on Software Engineering, pages 13-22, May 1989.
- HPCTE implementation of University of Siegen
  - Java API
  - Supports views on the repository
  - http://pi.informatik.uni-siegen.de/pi/hpcte/hpcteapps.html