

14. Einführung in die Architektur von Software-Werkzeugen

1

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und
Multimediatechnik
<http://st.inf.tu-dresden.de>
Version 13-0.1, 31.10.13

- 1) Grobarchitektur von Werkzeugen
- 2) Datenablage (Repositoryum)
 - 1) Metamodellsteuerung
- 3) Werkzeuge als Tools and Materials
- 4) Beispiele für Repositorien
 - 1) Master Data Management



Obligatorische Literatur

2

- ▶ 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>

14.1 Grobarchitektur von Werkzeugen

3

vgl. Architekturstile von Anwendungen aus der ST-II:
Algebraische Anwendungen: aufrufbasiert
Coalgebraische Anwendungen: strombasiert

Architektur eines repository-basierten Werkzeugs (aufrufbasiertes Repository)

4



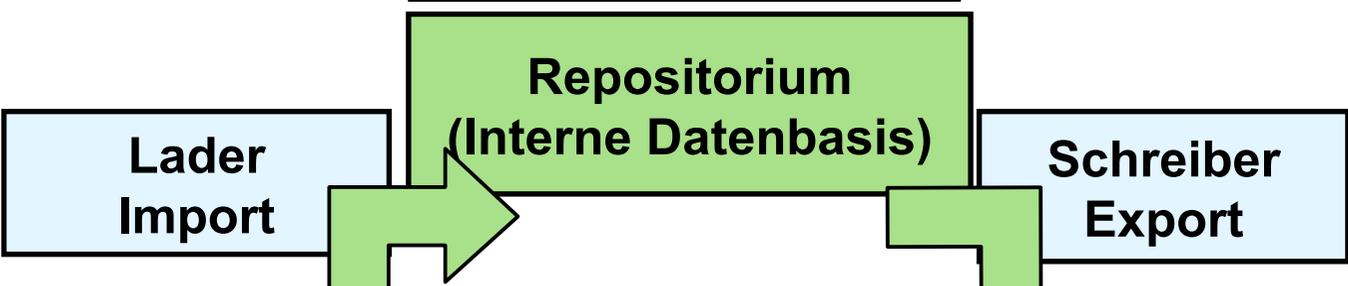
Schicht 3



Schicht 2



Schicht 1



Schicht 0



Tool-Repository-Schnittstelle

System-schnittstelle

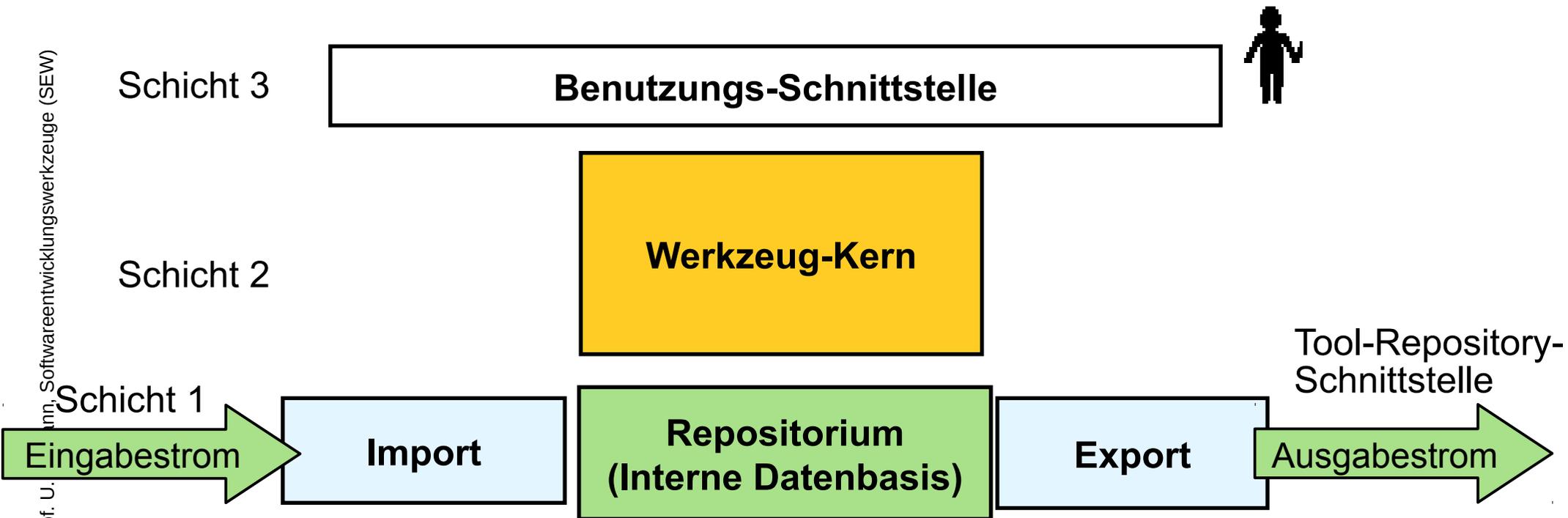
Prof. U. Alßmann, Softwareentwicklungswerkzeuge (SEW)



Architektur eines datenflussgesteuerten, strom-basierten Werkzeugs

5

- ▶ Arbeit wird stückweise erledigt; meist pro gelesenem Datenpaket.
- ▶ Eine DFD- oder Workflow- Sprache verknüpft (komponiert) die Werkzeuge durch ein DFD oder Workflow (Mashup) zu komplexeren Werkzeugen



14.2. Datenablage (Repository)

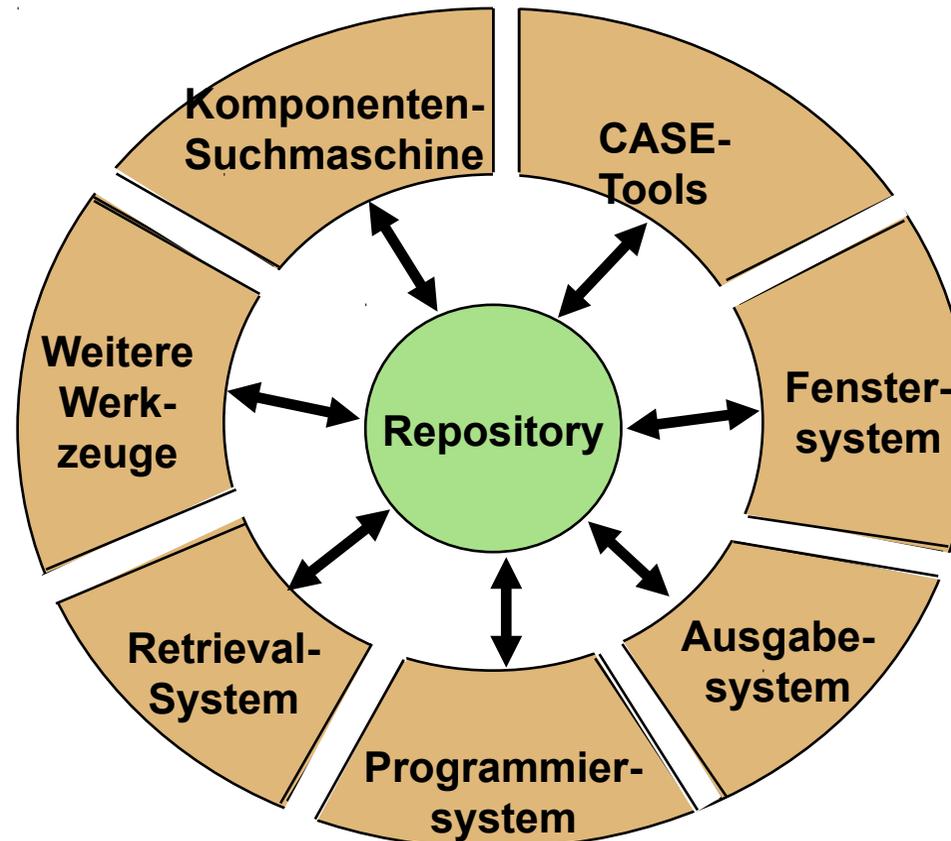
6

Prinzipiell entsprechen Repositorien den Speichern im DFD. Allerdings spricht man nur dann von einem Repository, wenn mehrere Prozesse darauf lesen und schreiben.

Ziele und Aufgaben des Repository

7

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

8

- ▶ **Transparente persistente Dokumentenspeicherung:** Einfache Abbildung aller Datenobjekte (Artefakte) auf persistente Datenstrukturen im Repository auch bei Änderung der Objekte (Systemausfall)
- ▶ **Entkopplung von physischen Speicherformaten:** Physische und logische Datenunabhängigkeit, d.h. Trennung der Programme (Werkzeuge) von logischer Struktur und physischer Repräsentation der Daten
- ▶ **Unterstützung von Metamodellen (Schemata)** beschrieben in DDL:
 - **Logisches Datenmodell:** Anwendungsnahes Datenmodell, in ausdrucksstarker DDL
 - **Physische Datenmodell:** Wie werden die Daten gespeichert? meist in Relational Schema
 - **Externe Schemata:** Für Steuerung des Zugriffs, in dem zu unterschiedlichen Rollen verschiedene externe Schemata für ein Dokument definierbar sind
 - **Schema-Änderungen:** Sollen bestehende Daten in der Datenbasis erhalten
 - **Typisierte Zugriffsschnittstellen:** Programmierbare APIs (Anfragesprachen, prozedurale Programmierschnittstellen), möglichst aus Metadaten generiert
 - **Refektive Zugriffsschnittstellen:** Programmierbare APIs, die typunabhängig sind und unabhängig vom logischen oder physischen 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)

9

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



14.2.1 Metamodell-Steuerung von Datenablagen



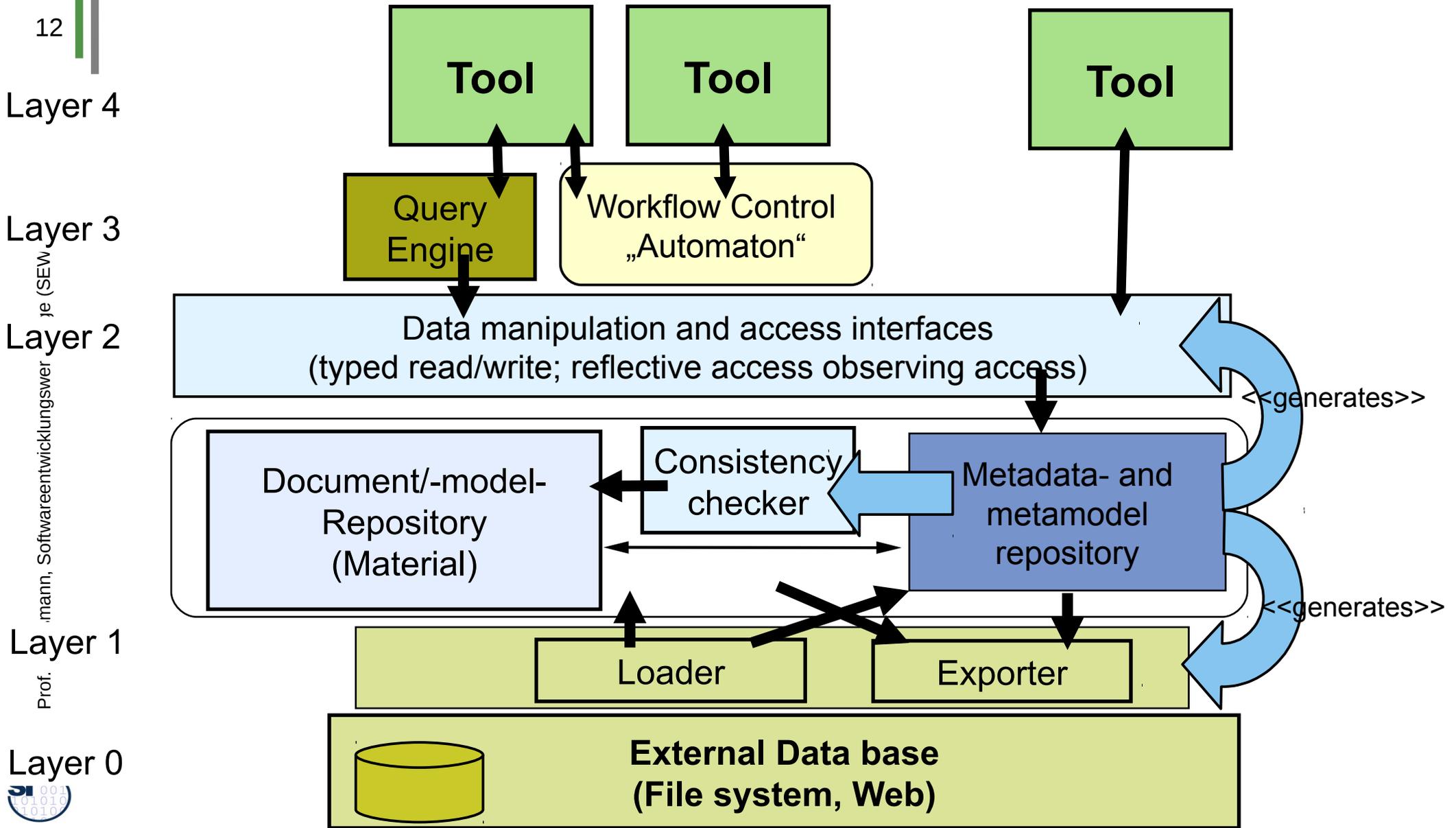
10

Metamodellgesteuerte Repositorien

11

- ▶ Ein **metamodellgesteuertes Repository** 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 Eigenschaften 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

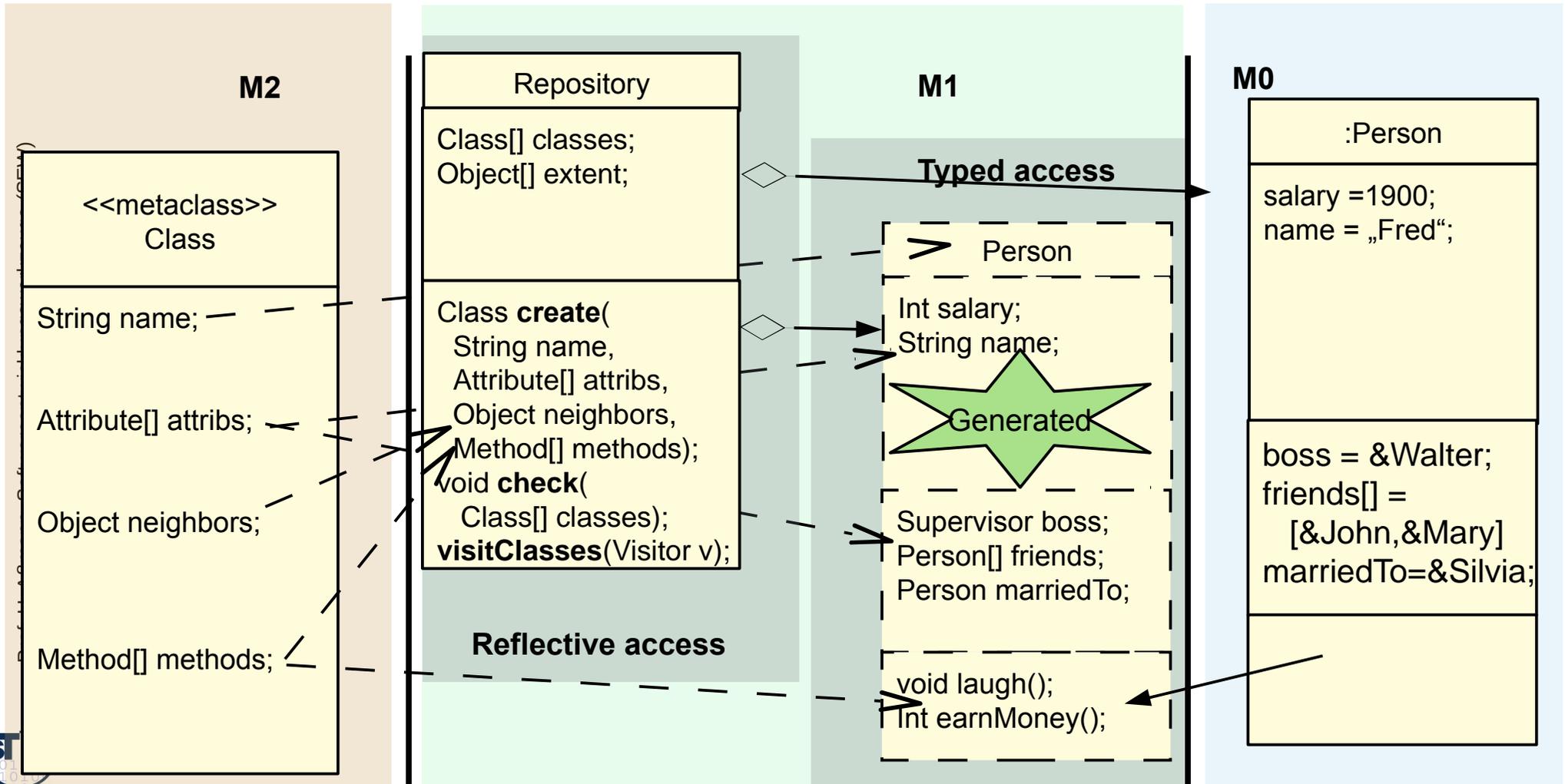
Grobarchitektur bei Datenteilung (Metamodellgesteuertes Repository)



Generierung von typisierten Schnittstellen

13

- Aus den Metaklassen wird die Struktur der Typen der Zugriffsschnittstellen und vieler anderer Code-Pakete auf M1 abgeleitet (compartments):



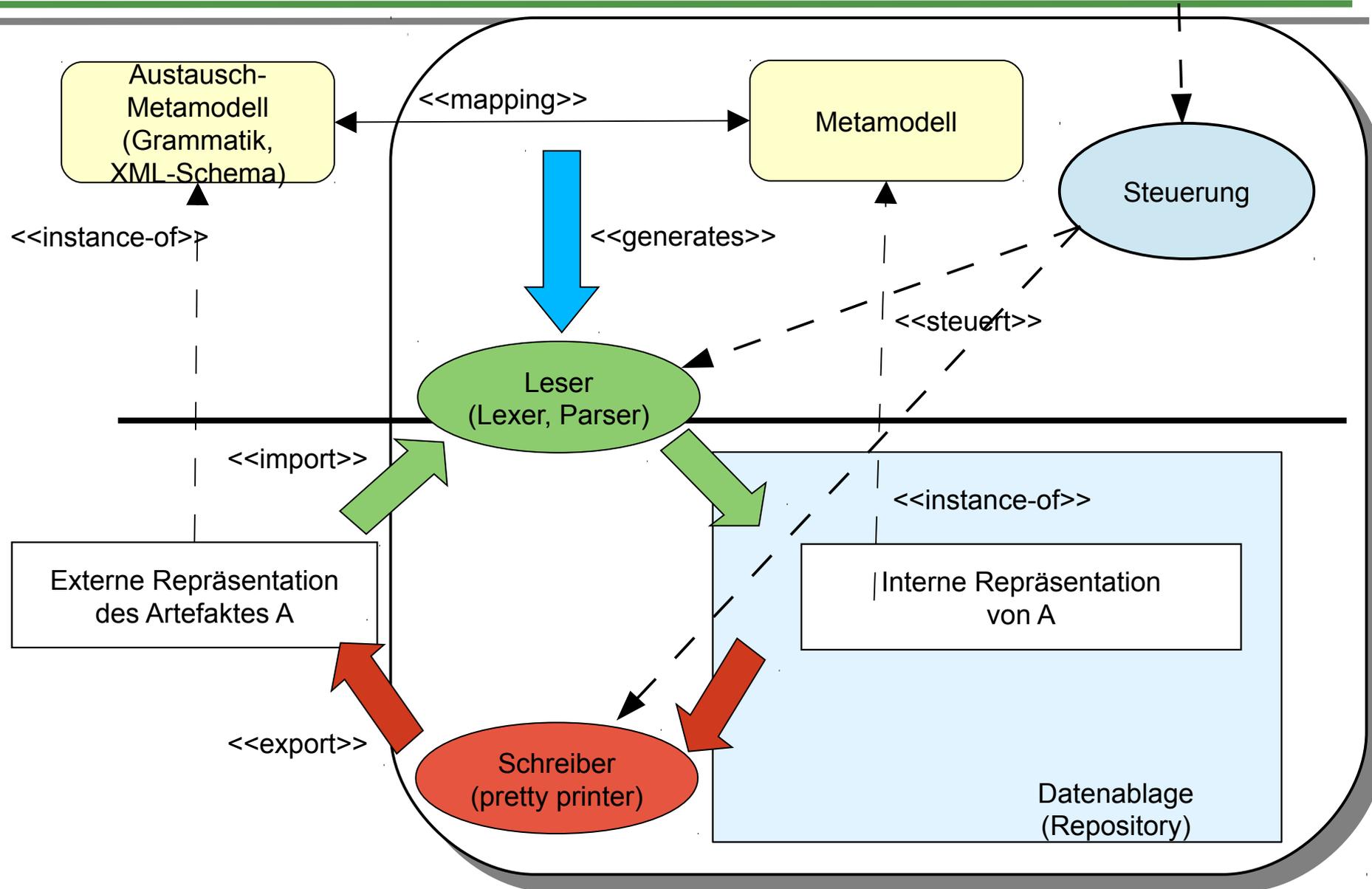
Generierung von Schnittstellen und Paketen in einem metamodellgesteuerten Repository

14

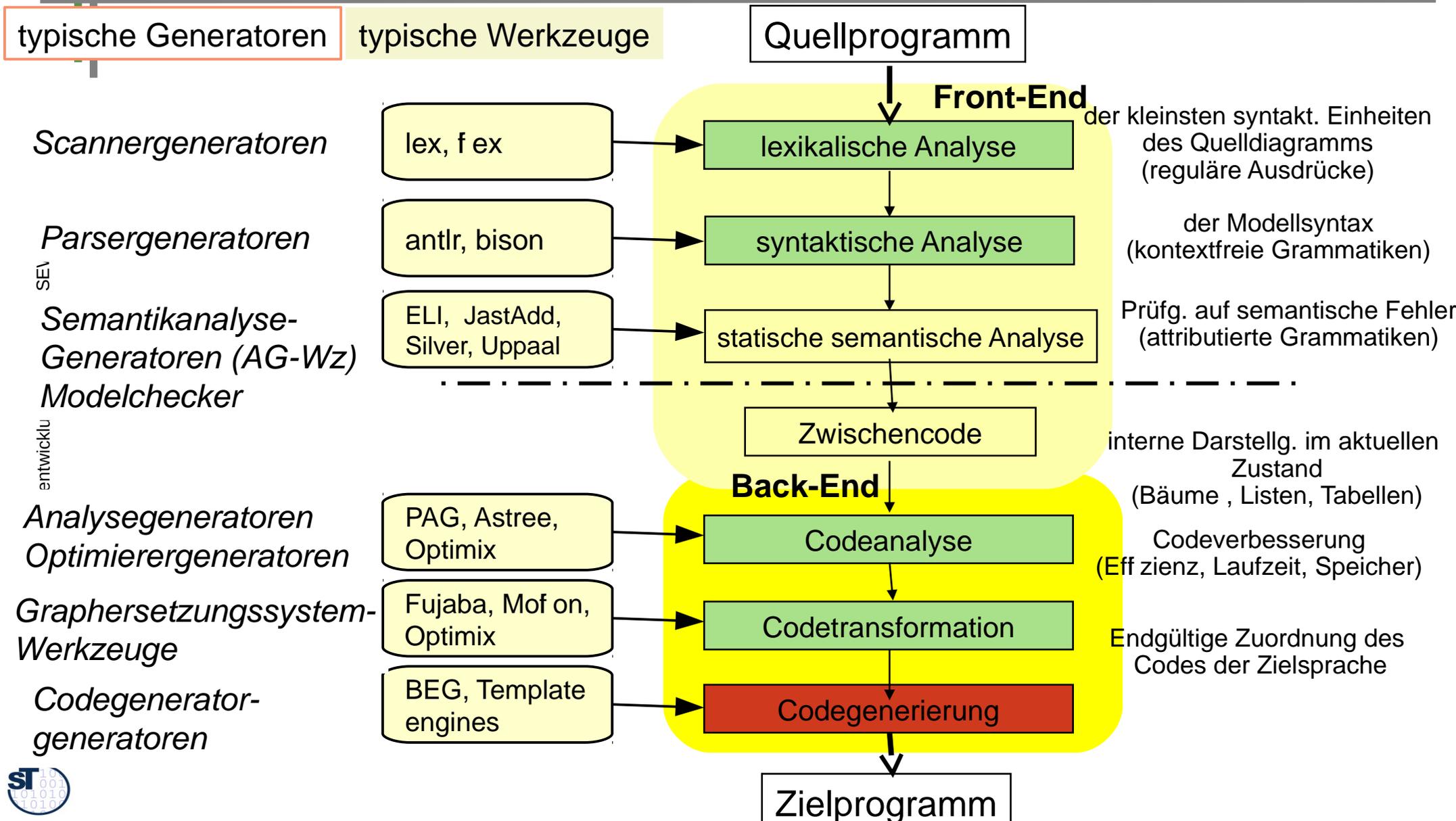
- ▶ Generierung von **statisch typisierten Zugriffsschnittstellen** zu den Modellen
 - **Lese- und Schreib-Schnittstellen**, inkl. deren Implementierung mit Leser/Schreiber-Synchronisations- bzw. Transaktionsprotokollen
 - **Typisierung** wird von dem Metamodell aus M2 geliefert (spezifiziert in DDL)
 - **Strukturinformation** (atomare Attribute, Strukturattribute, etc.)
- ▶ **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 **Navigatinos-Paketen**: Zur Navigation auf den Objekten (depth-first, breadth-first, inside-out, outside-in, layer-wise, etc.)
- ▶ Generierung von **Datenaustausch-Schnittstellen**:
 - Generierung von **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)

Nutzung von generierten Parsern zum Import in Repositorien

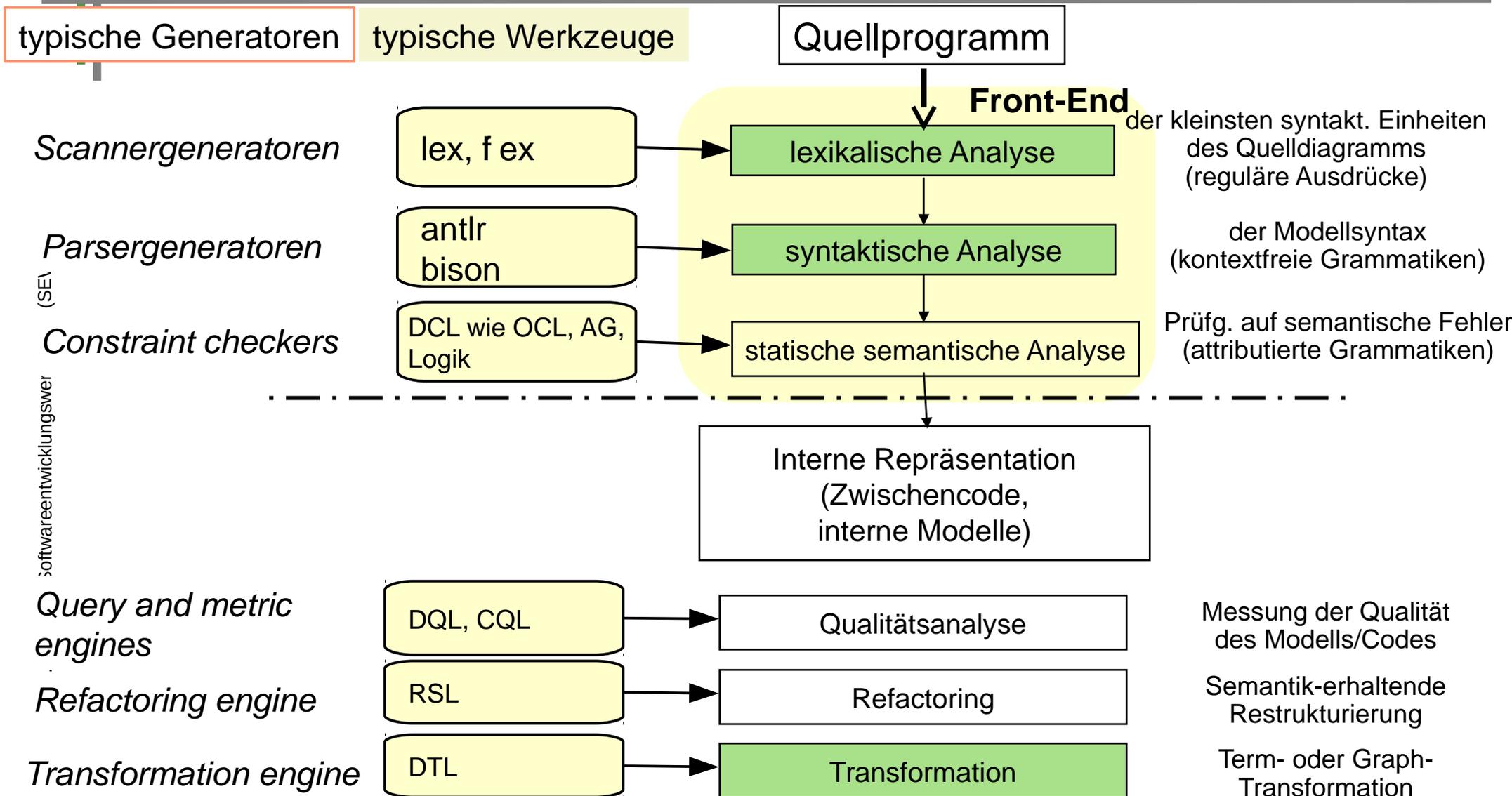
15



Phasen eines Werkzeugs und ihre erzeugenden Werkzeuge



Phasen eines Importers in ein Repository und die erzeugenden Werkzeuge



14.3 Werkzeuge als Tool-Objekte, die auf Materialien in der Datenablage zugreifen

18

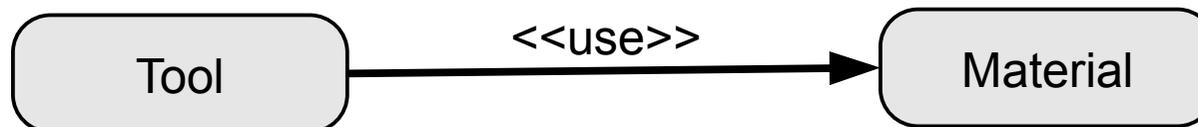
- Objekt-orientierte Sicht auf Werkzeuge:
- Ein Werkzeug (tool) kann als ein Objekt gesehen werden, das einen externen Zustand über Material-Objekten in einem Repository verwaltet.
- Siehe “Design Patterns and Frameworks”, Kapitel “Tools and Materials (TAM)”



Tool-Material Collaboration Pattern

19

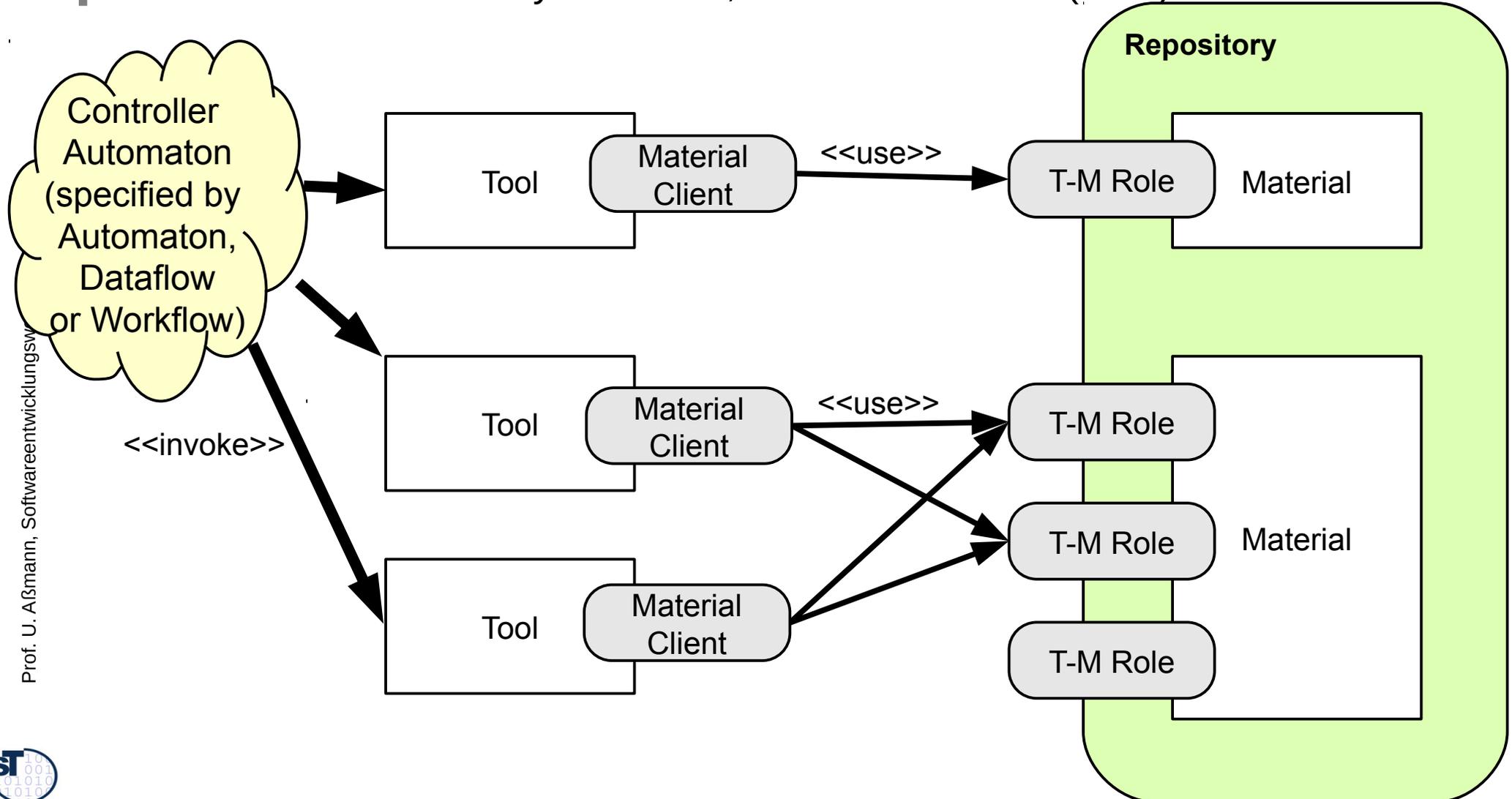
- ▶ A *tool-material collaboration* (T&M role model, T&M access aspect) expresses the relation of a tool and the material
 - The tool is active, has control; the material is passive and contains data
 - Characterizes a tool in the context of the material, and the material in the context of a tool
- The tool's access of the material. The tool has a view on the material, several tools have different views
 - The tool sees a *role* of the material, in collaboration with a tool
 - Roles of a material define the necessary operations on a material for one specific task
 - They reflect usability: how can a material be used?
 - They express a tool's individual needs on a material



Controller Automaton, Tools and Their Views on Material

20

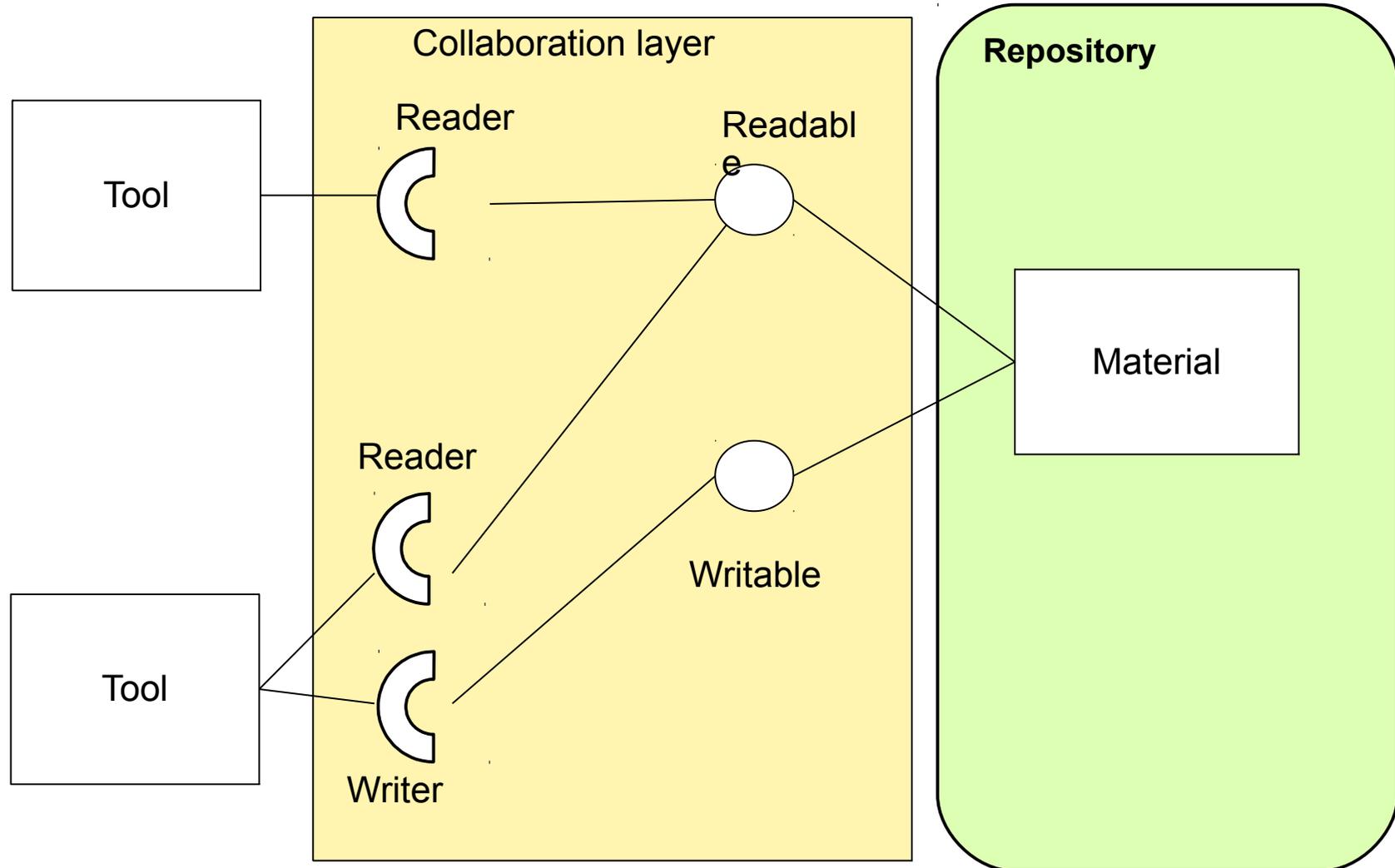
- ▶ Tools are controlled by automata, DFD or workflows (TAM)



Collaboration Layer in UML Component Diagram Notation

21

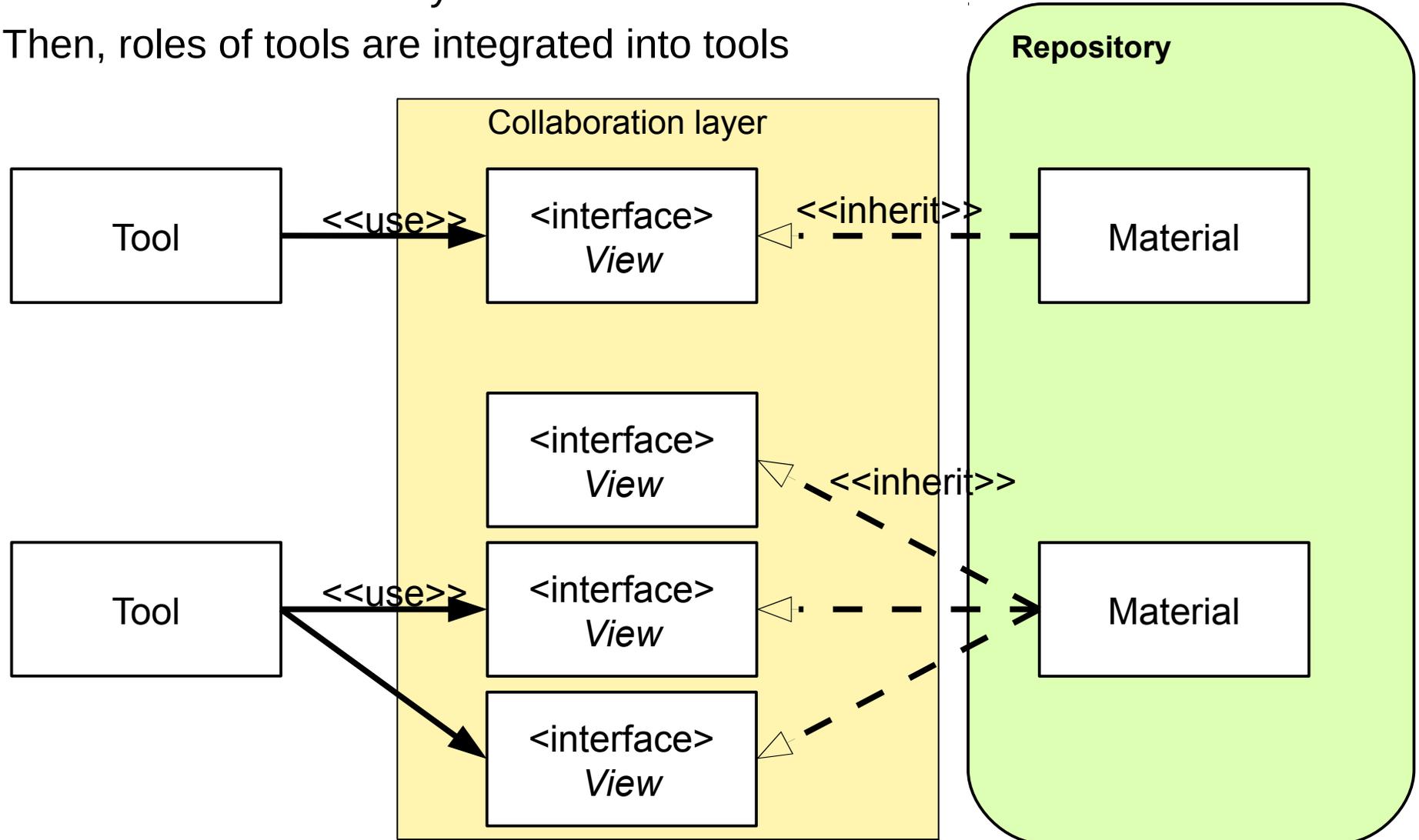
- ▶ In UML, roles are represented by “lollipops” and “plugs”



Implementing Tool-Material Roles in the Collaboration Layer With Interfaces in Java or C#

22

- ▶ Roles can be realized by mixin inheritance or interface inheritance
- ▶ Then, roles of tools are integrated into tools

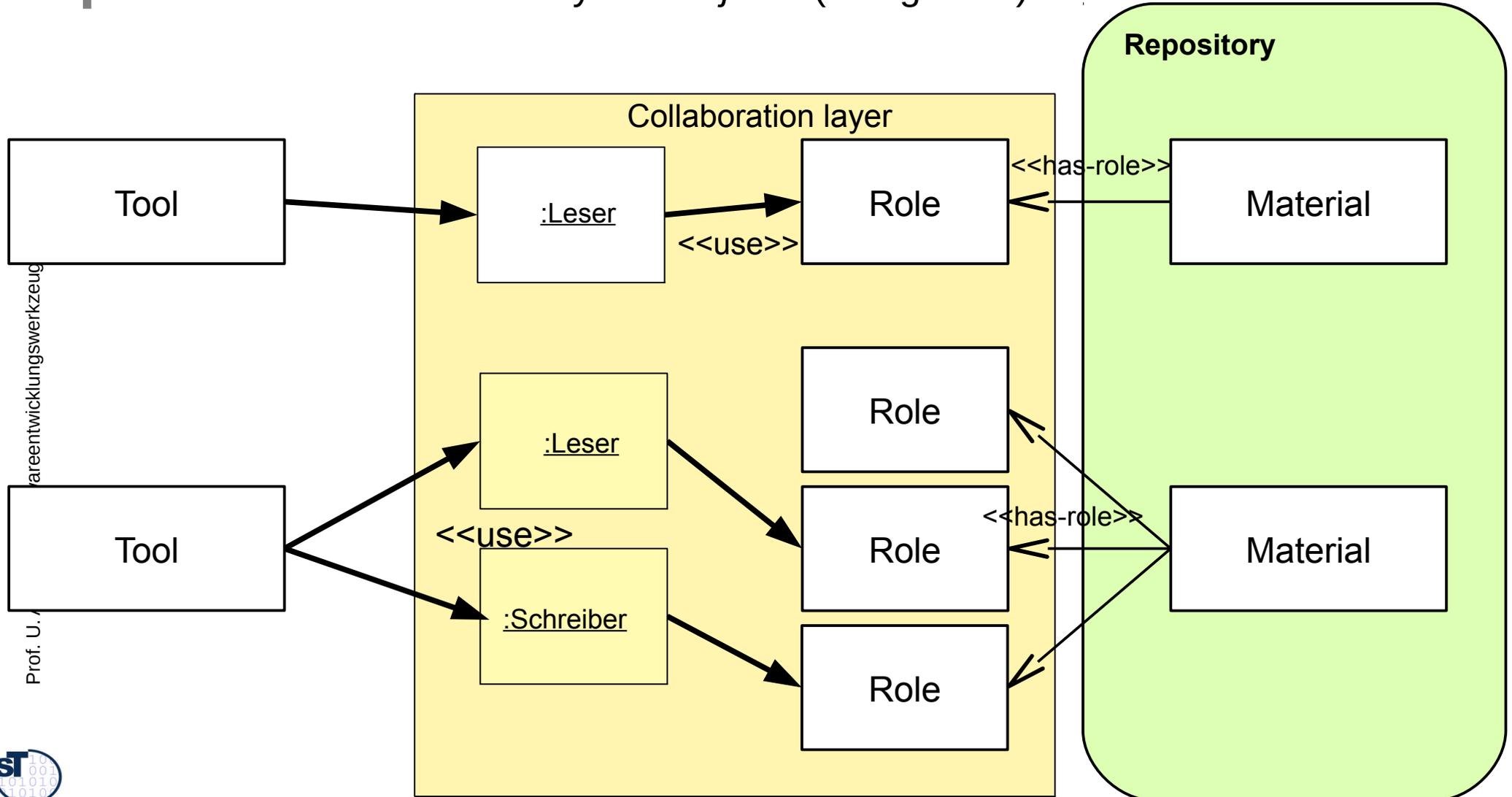


Implementing Tool-Material Roles in the Collaboration Layer

With Role Objects

23

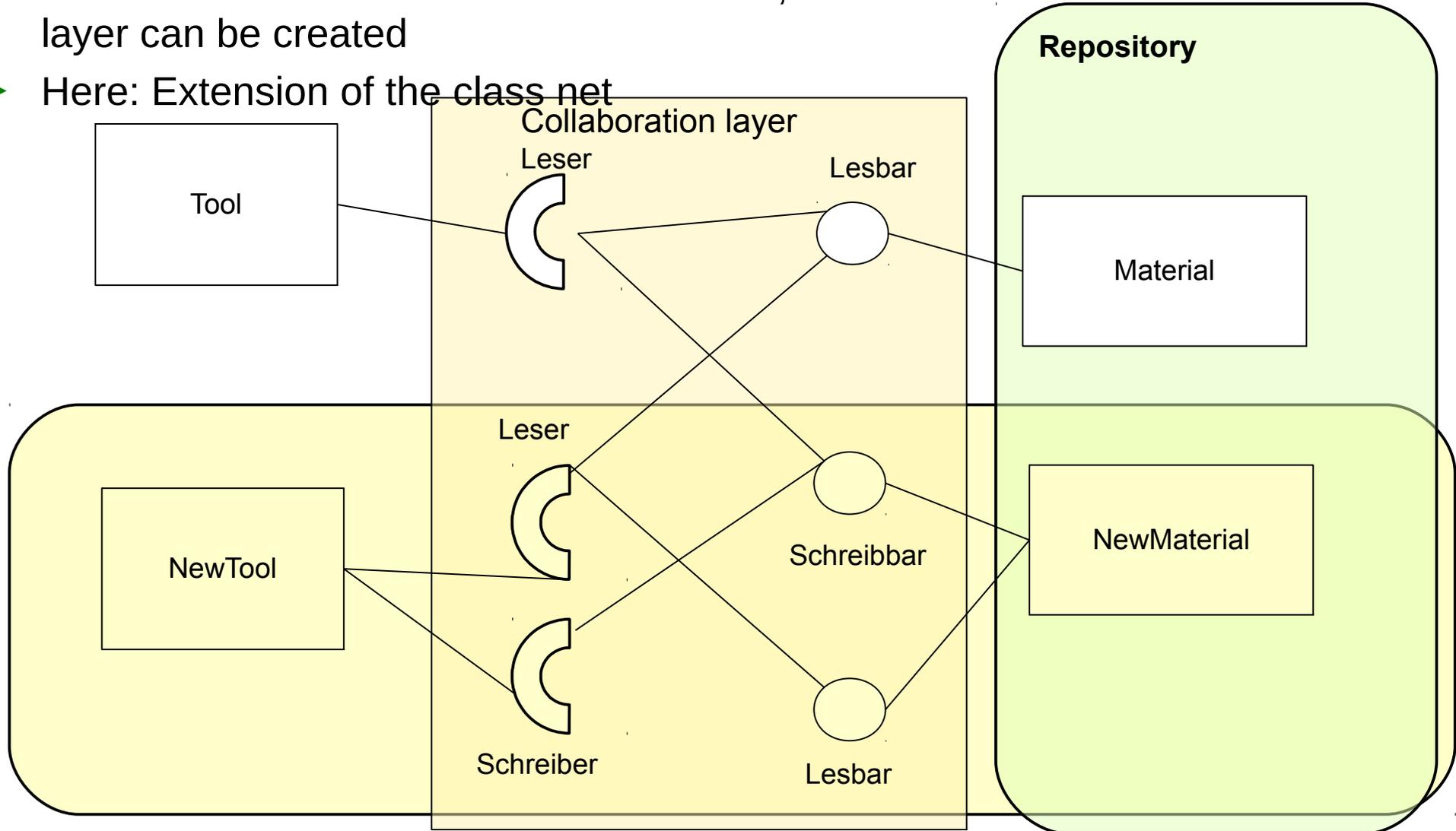
- ▶ Roles can be realized by *role objects* (delegates)



Extension of a Collaboration Layer

24

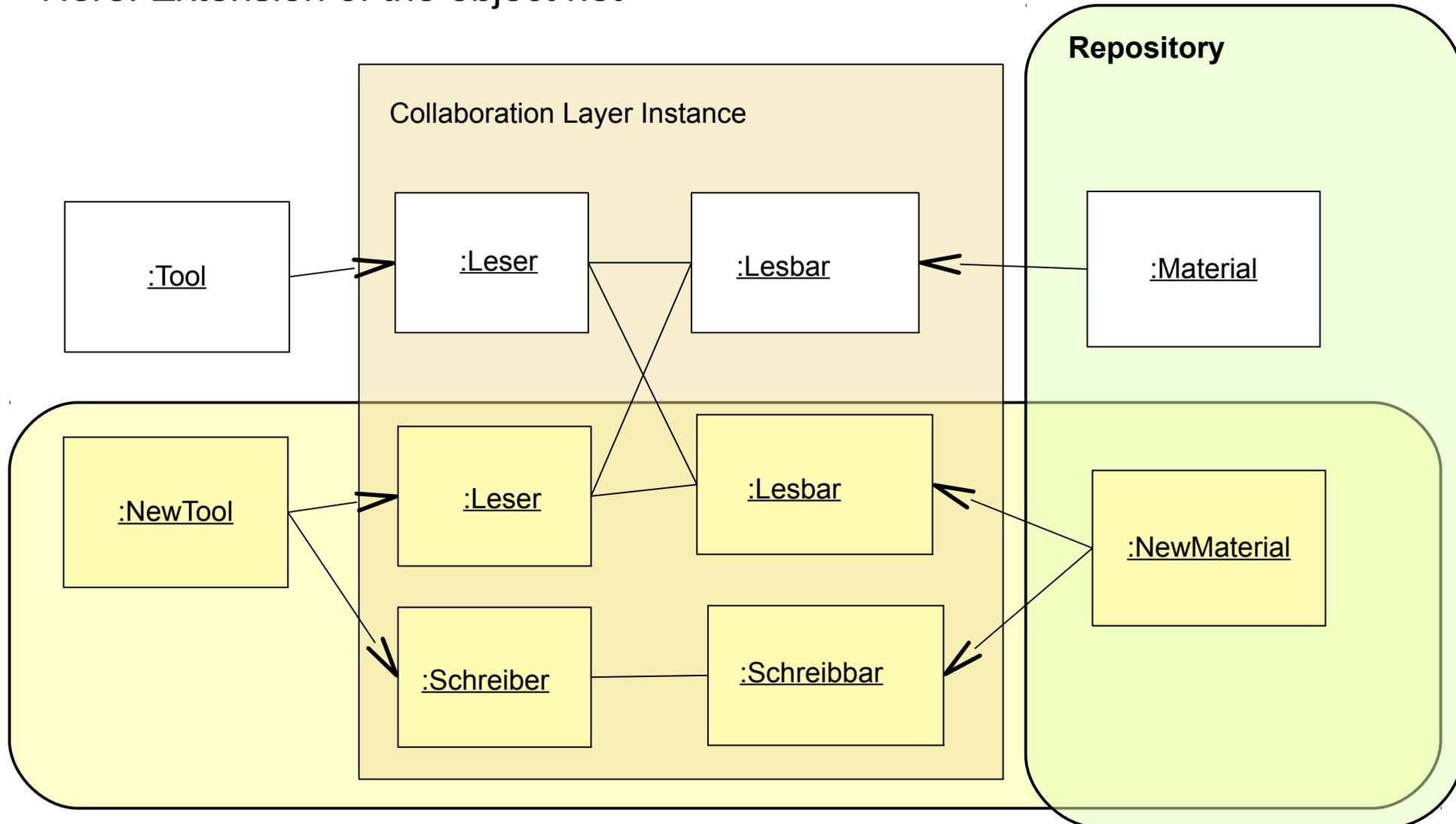
- ▶ If a set of tools and materials is extended, new relations in the collaboration layer can be created
- ▶ Here: Extension of the class net



Extension of the Object Net in the Collaboration Layer

25

- ▶ Here: Extension of the object net



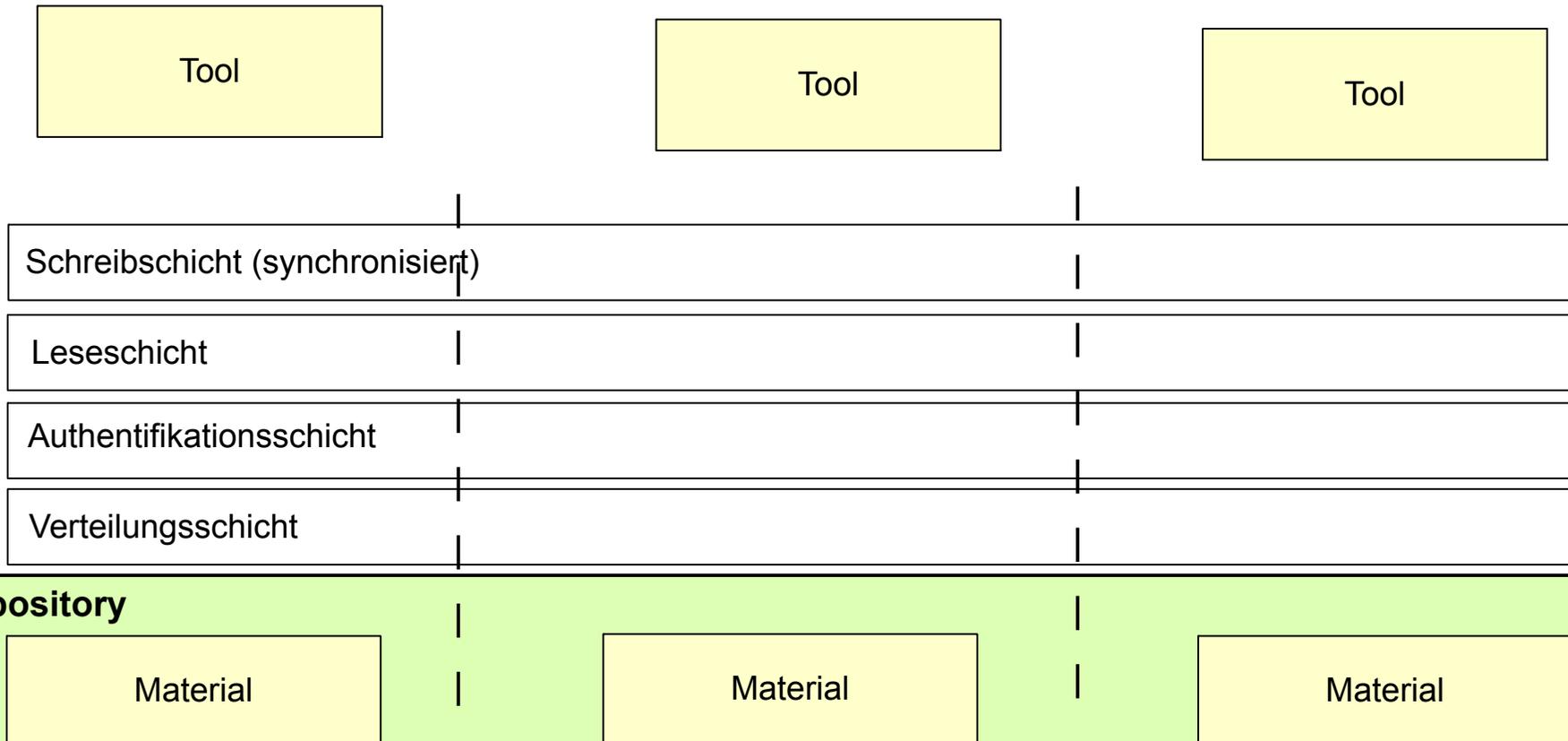
14.3.1 Schichtung der Material-Zugriffsrollen für Werkzeugobjekte (tool objects)

26

Überblick über die Zugriffsschichten des Repositories

27

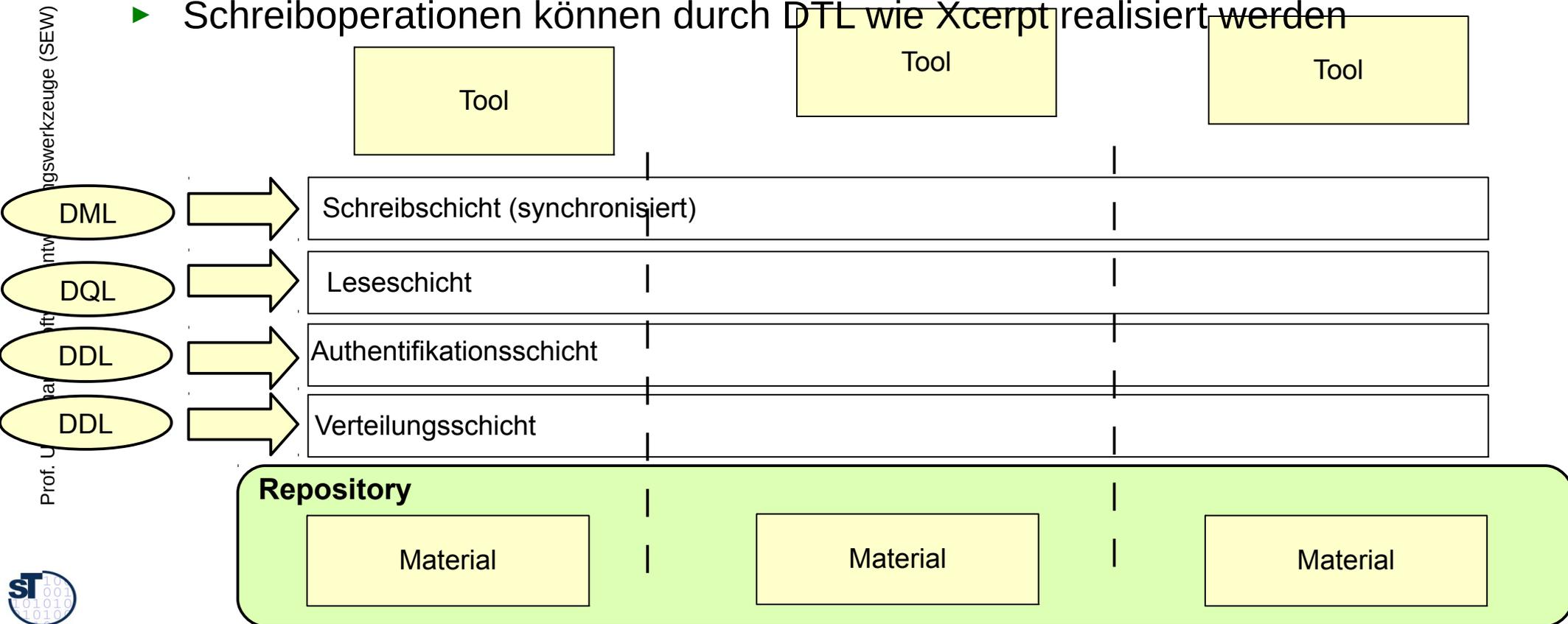
- ▶ In der Sicht von TAM, greifen Werkzeugobjekte durch mehrere Schichten von Rollenobjekten hindurch auf die Materialien zu.
- ▶ Die Rollenobjekte sind für bestimmte Zwecke (Belange) zuständig
- ▶ Es entsteht eine Belang-Materialobjekt-Matrix



Generierung der Schichten aus speziellen Sprachen des Technikraums

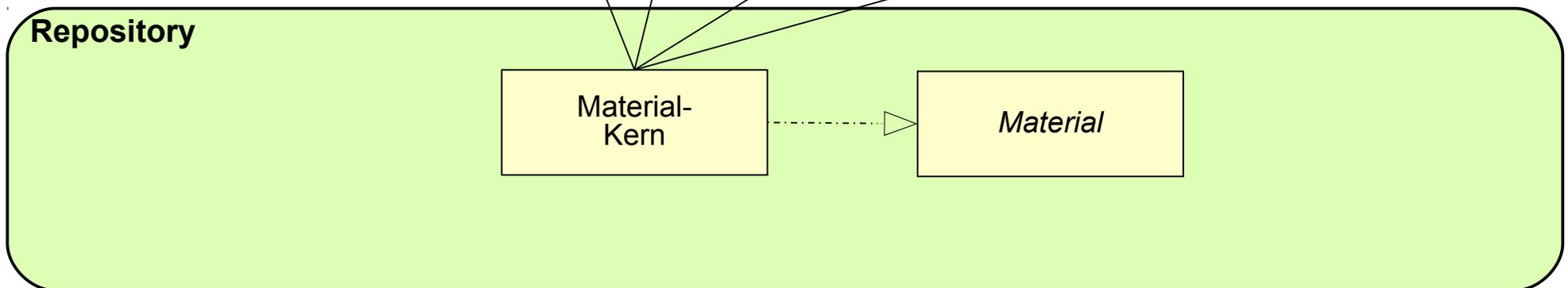
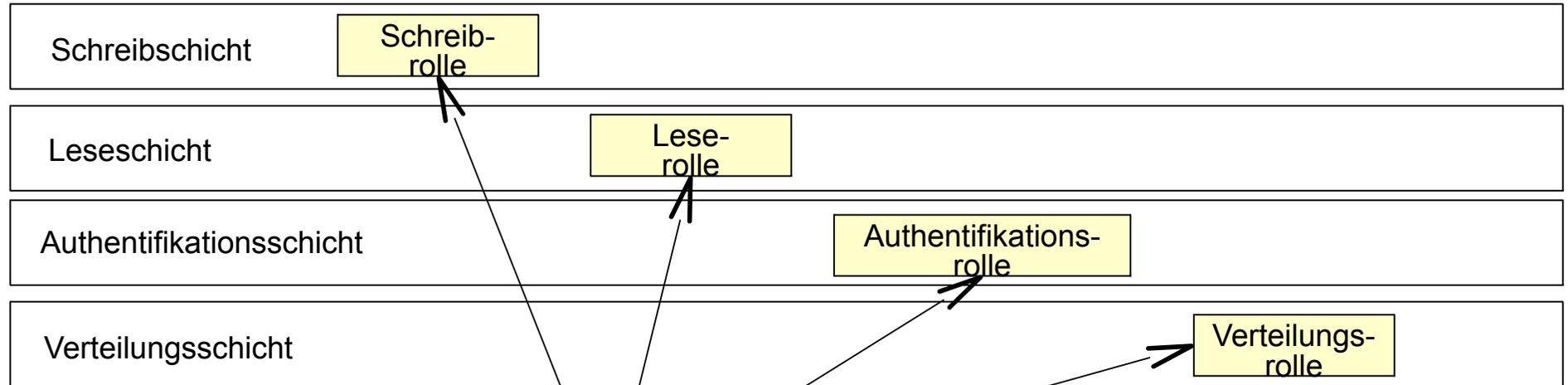
28

- ▶ Die Implementierung der Schichten kann durch (verschiedene) Sprachen spezifiziert sein
- ▶ OCL-constraints können Constraints in der Leseschicht spezifizieren
- ▶ Xquery kann als Anfragesprache in der Leseschicht verwendet werden
- ▶ Schreiboperationen können durch DTL wie Xcerpt realisiert werden



Material, realisiert als Kern mit Delegationen

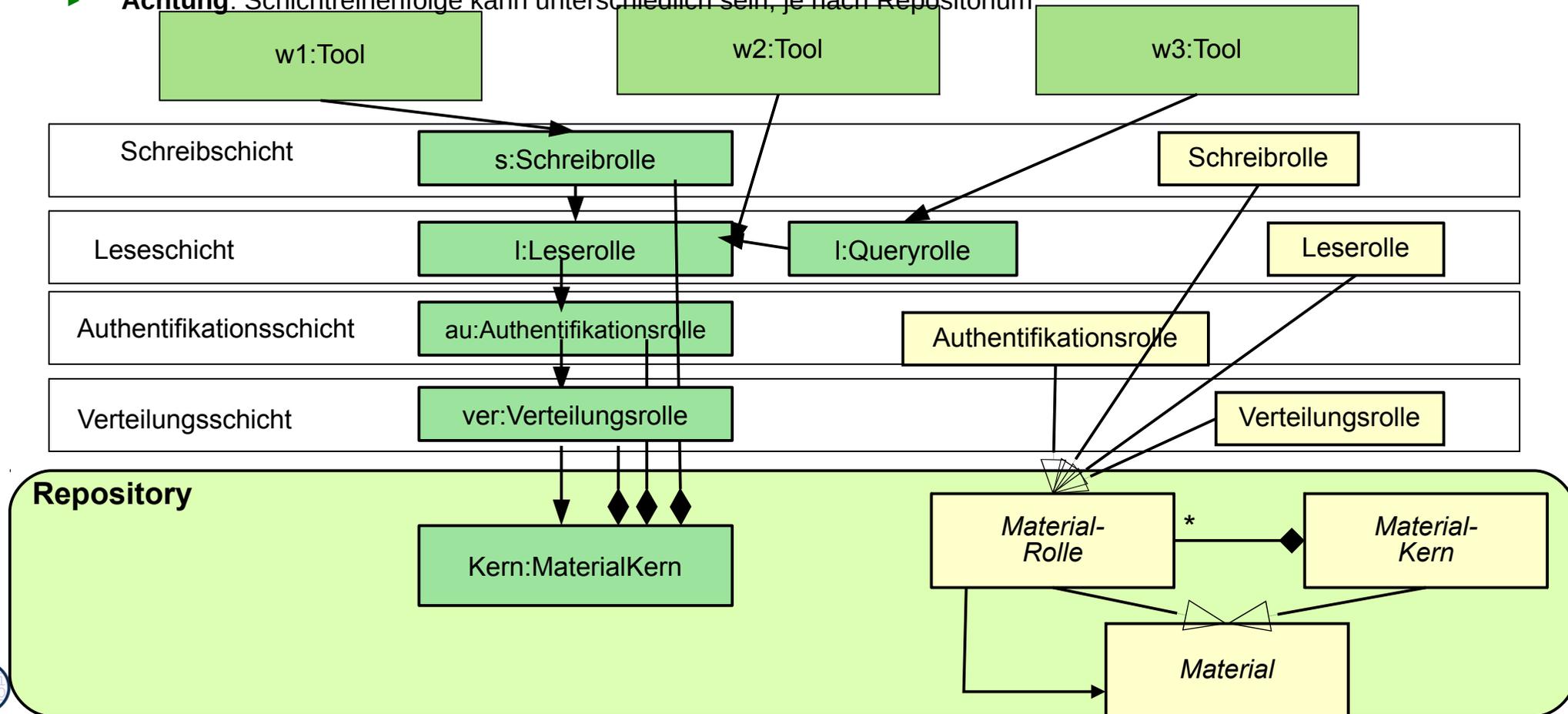
29



Kollaboration von Werkzeugen auf dem Repository mit Rollenobjekten

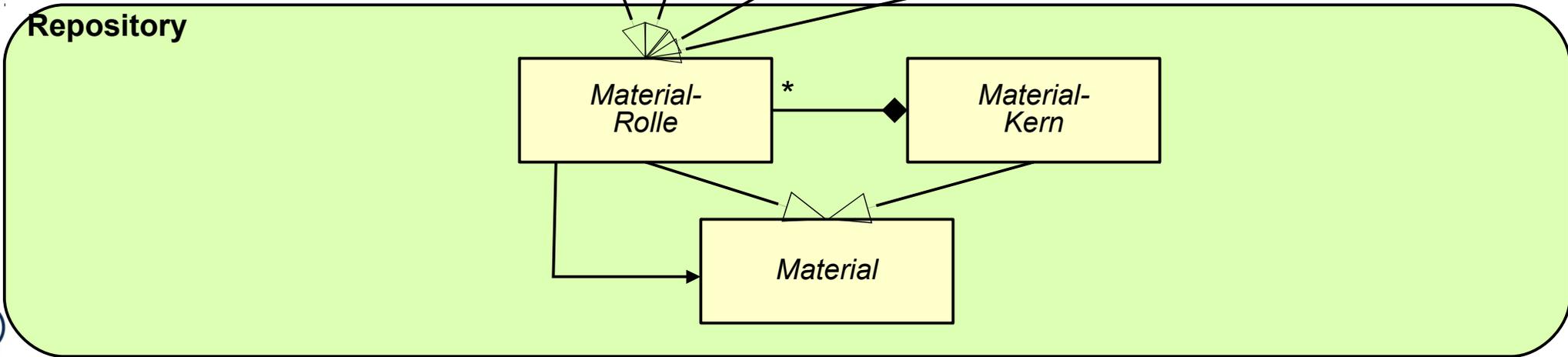
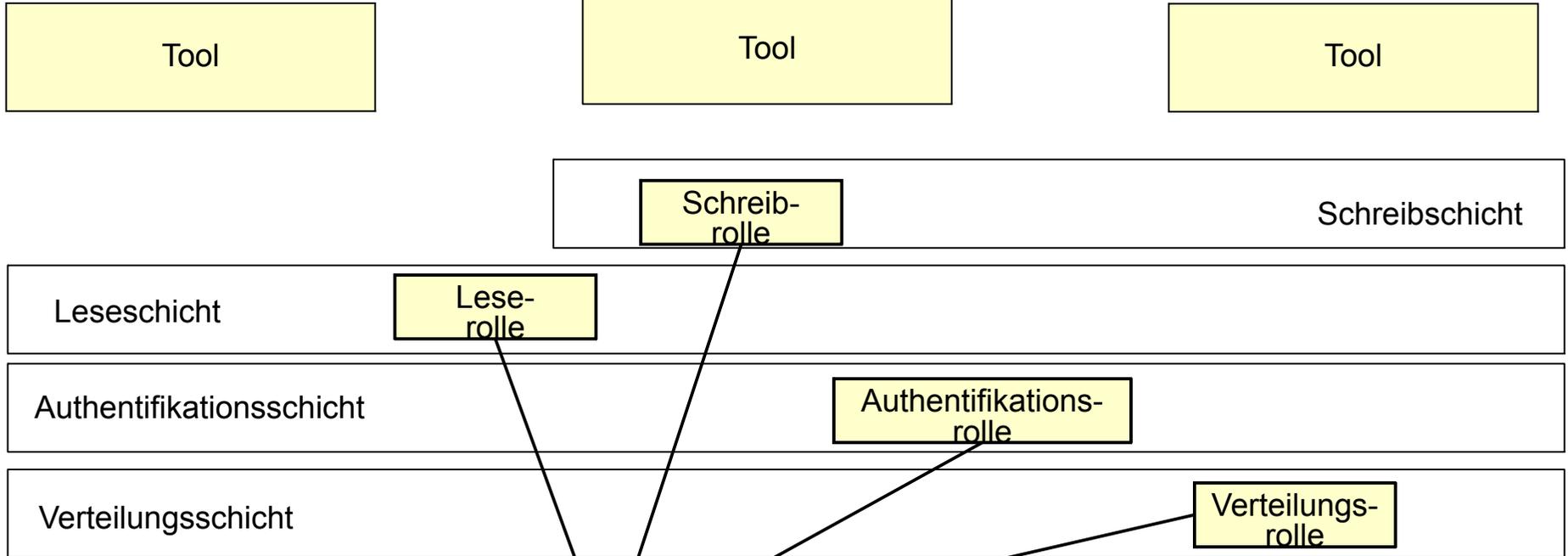
30

- ▶ 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)
 - im RO-Pattern regeln Dekorator-Ketten über Schichten den Zugriff auf Materialien im Repository
- ▶ Viele Schichten, vielen Rollen-Klassen können generiert sein
- ▶ **Achtung:** Schichtreihenfolge kann unterschiedlich sein, je nach Repository



Material als Rollenobjekt-Muster

31



Werkzeug-Effekte auf Materialien und Ihre Prägung für Rollen

32

▶ CRUD-Schnittstelle von Materialien in Informationssystemen bietet die Funktionalitäten:

- Create
- Read
- Update (Modification)
- Delete

CRUD wird im Folgenden leicht erweitert:

▶ Erweiterungs-Rolle (**E**xtend)

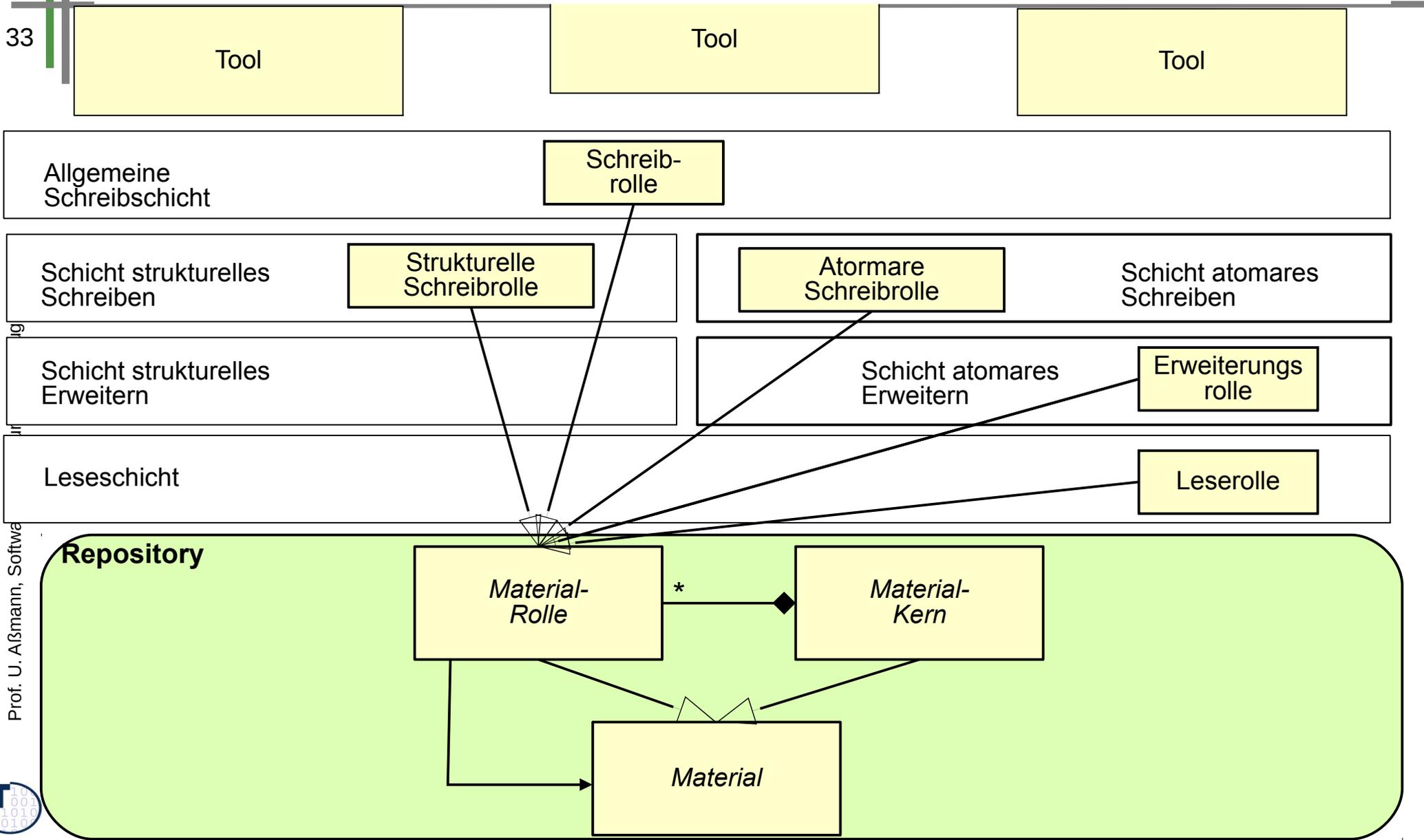
- Erweitert den Zustand des Objektes, zerstört ihn aber nicht
- Atomare Erweiterung
 - Erweitert den Zustand der atomaren Attribute des Objektes
- Strukturelle Erweiterung
 - Erweitert die strukturellen Attribute des Objektes, d.h. Des Objektnetzes, ohne die atomaren Attribute zu modifizieren

▶ Modifikation (**U**ppdate)

- Atomare Modifikation
 - Ändert den Zustand der atomaren Attribute des Objektes
- Strukturelle Modifikation

Verfeinerte Schichten-Architektur: Weitere Effekte auf Materialien und Ihre Prägung für Rollen

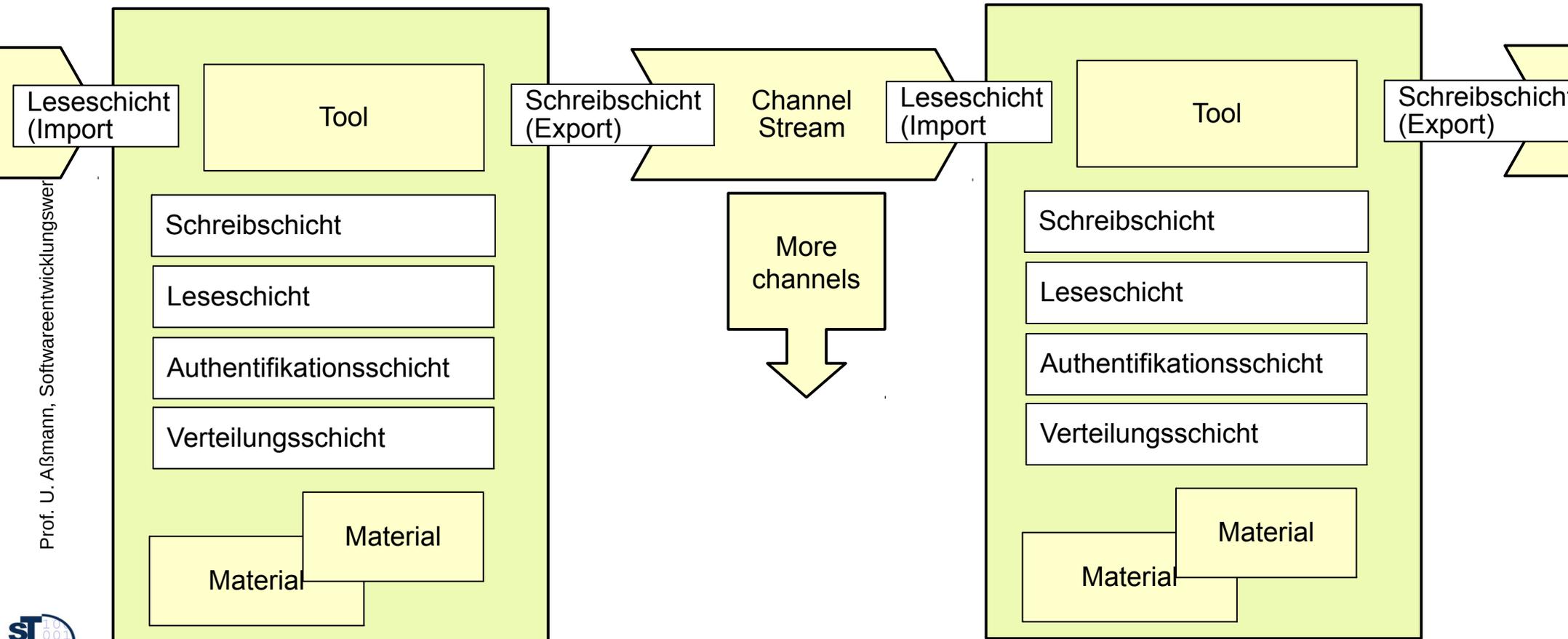
33



TAM für strombasierte Werkzeug-Architekturen

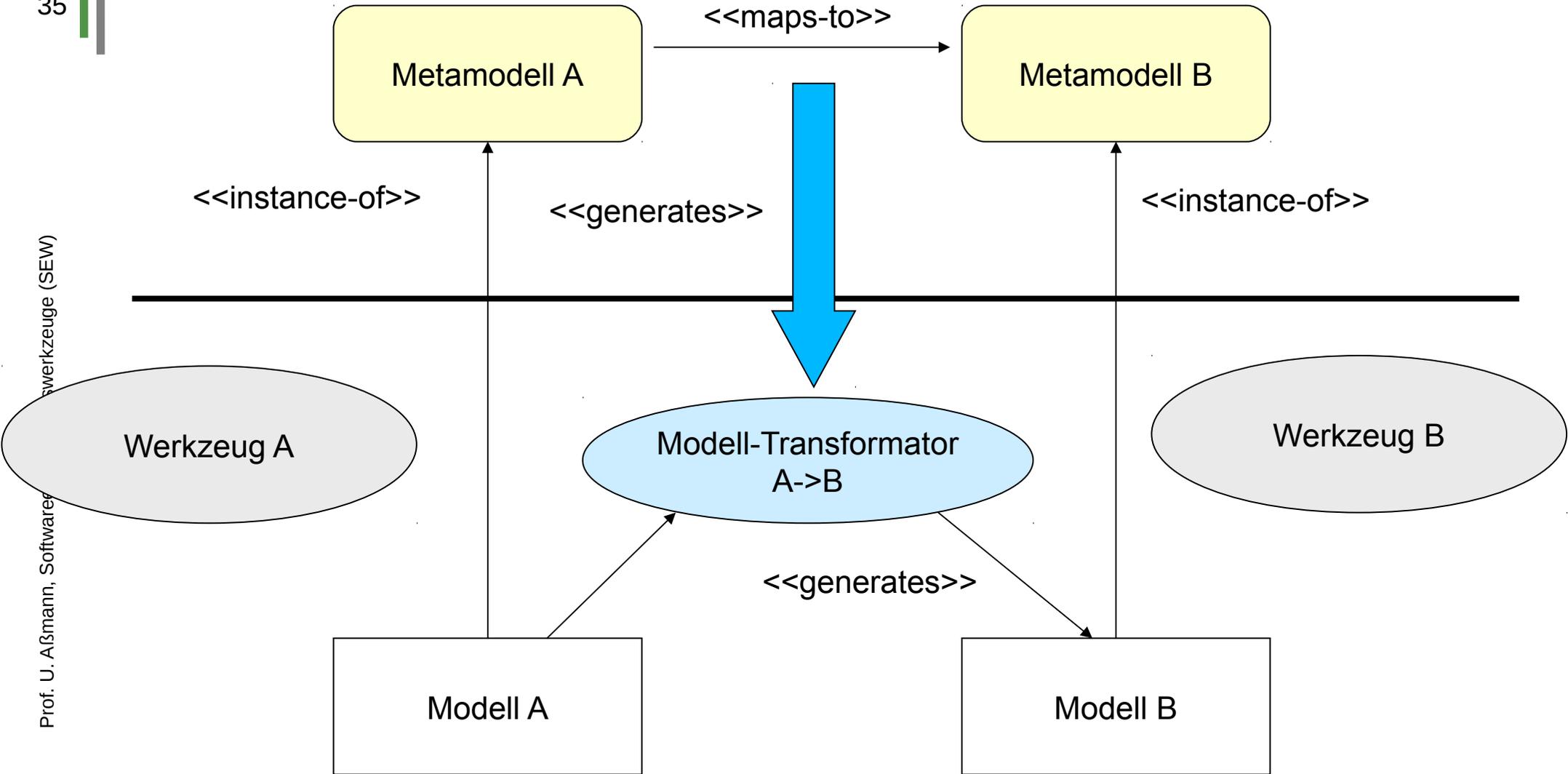
34

- ▶ Ist ein Werkzeug in eine strombasierte Architektur eingebunden, besitzt es separate Lese- und Schreibschichten bzgl. der Input bzw. Output-Channels.



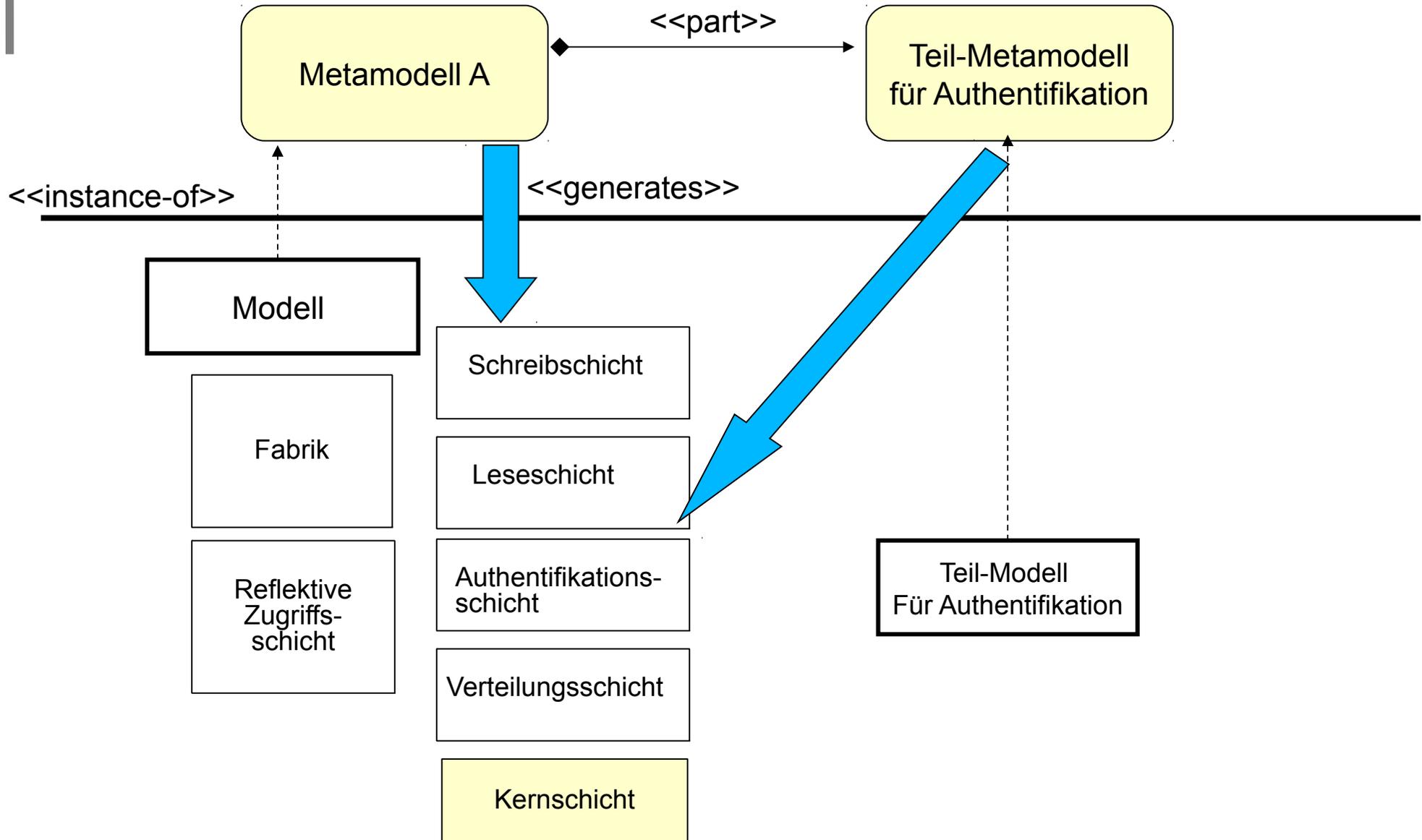
Generation of Model Transformer from Metamodel Mapping

35

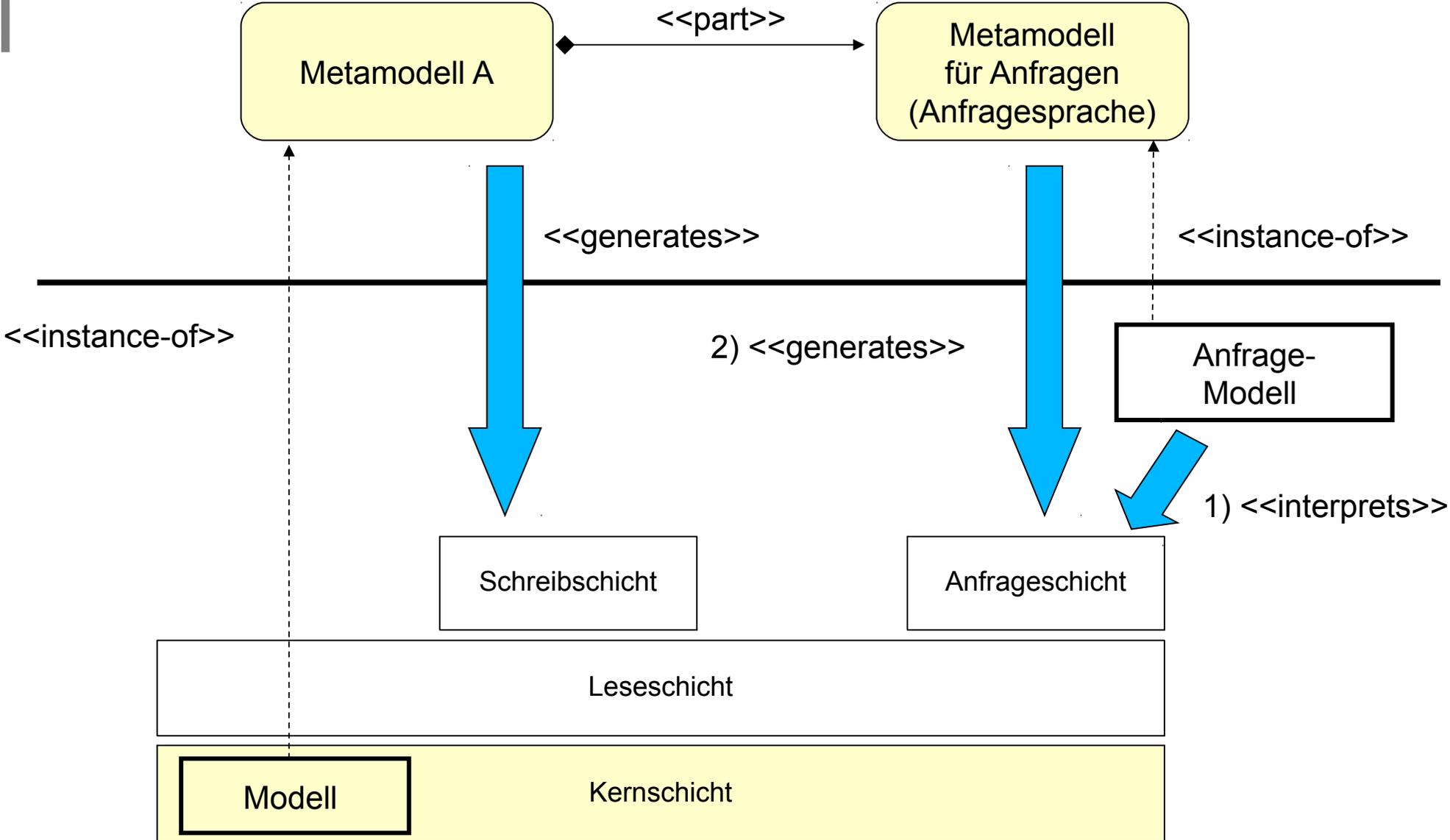


Generation of Access Layers from Metamodels

36

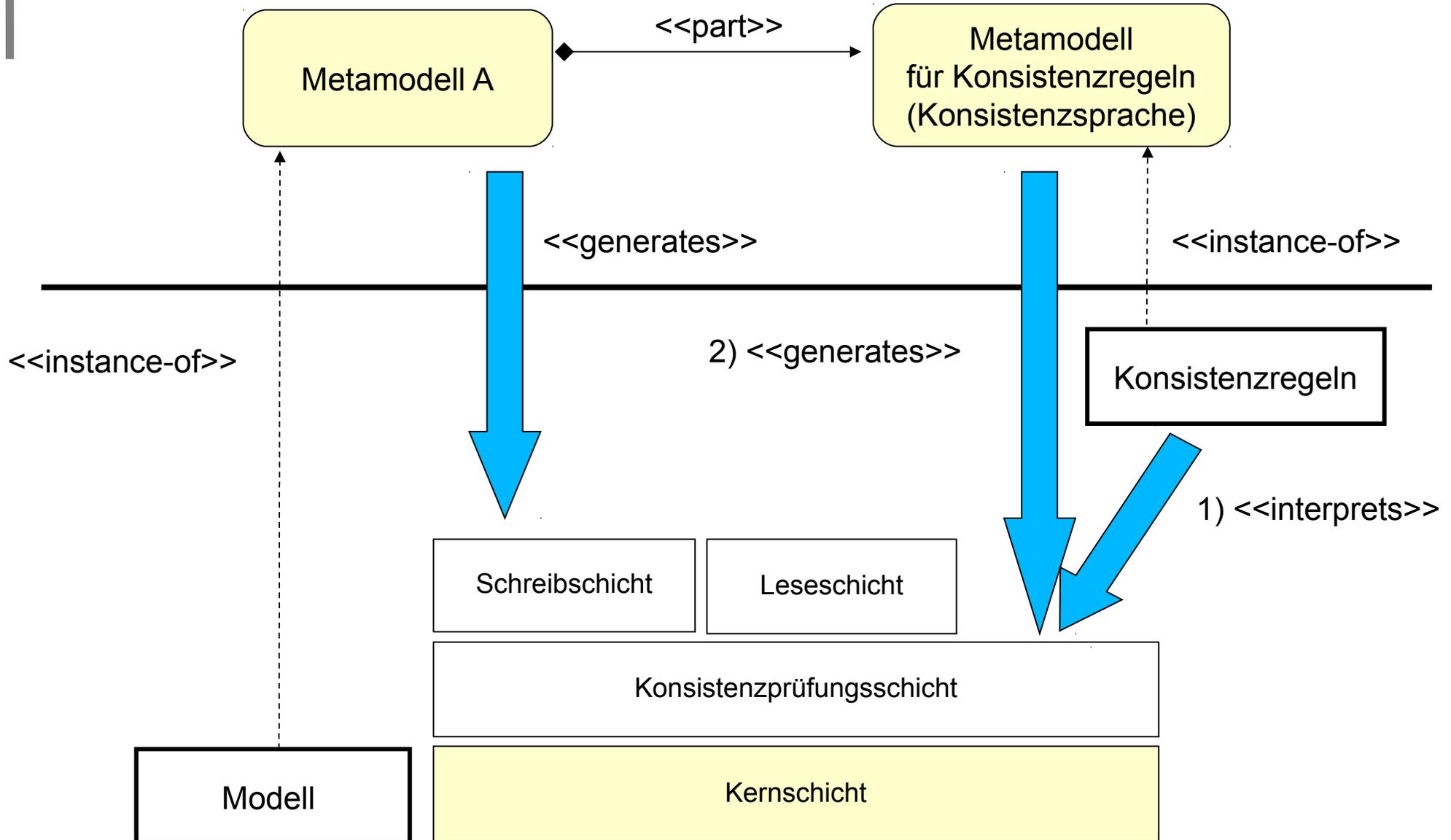


Generation of Query Layer from Metamodel



Generation of Administrative Layer

38



14.4 Beispiele für Datenablagen

39

Datenablagen können für verschieden Szenarien eingesetzt werden:

Temporär für ein Werkzeug

Persistent für ein Werkzeug

Persistent für mehrere Werkzeuge

Persistent für ganze Firma

Historische Beispiele für Repositories

40

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

41

Bezeichnung	Kurzcharakteristik
WebSphere Repository Database von IBM	Administration-Tools zur Installation, Überwachung und Pflege 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 Pflege, Konsolidierung und Harmonisierung integrierter Informationen über gültige Stammdatenattribute

Beispiele für metadatengesteuerte Repositories

42

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

43

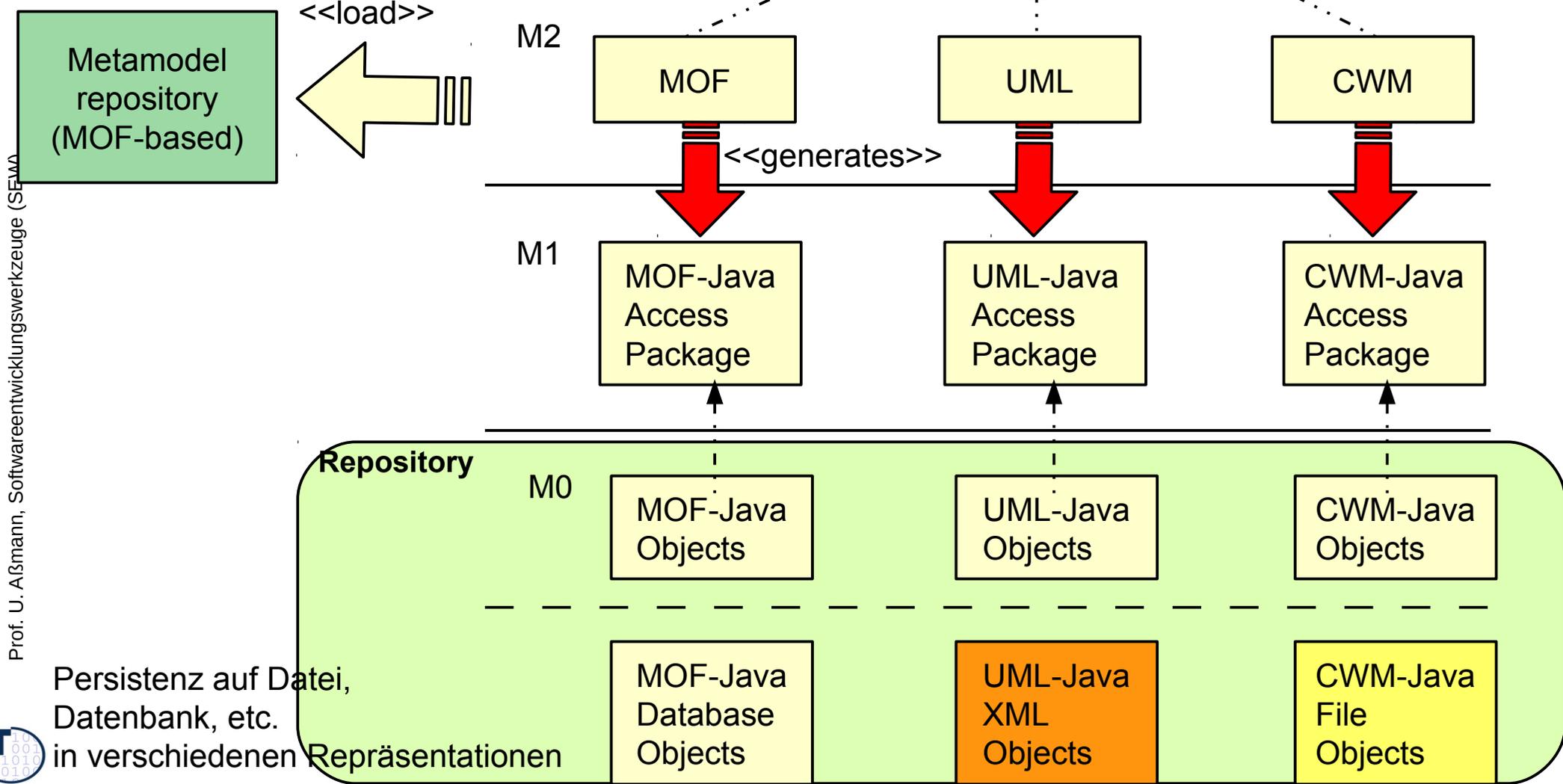
Bezeichnung	Kurzcharakteristik
Microsoft Sharepoint	Webbasiertes, filesystem-basiertes Repository
WebDAV	Webprotokoll zum verteilten Datenmanagement
Subversion/CVS/git	Versionsverwaltungssysteme auf File-Basis; Verwaltung von Versionen aller Files und Verzeichnisse; über spezielle Clients vom Browser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

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

- ▶ Ein metamodelgesteuertes, persistentes Repository für die Netbeans SEU <http://www.netbeans.org>, Metasprache MOF
 - Kann firmenweit genutzt werden
 - 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

45



Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)

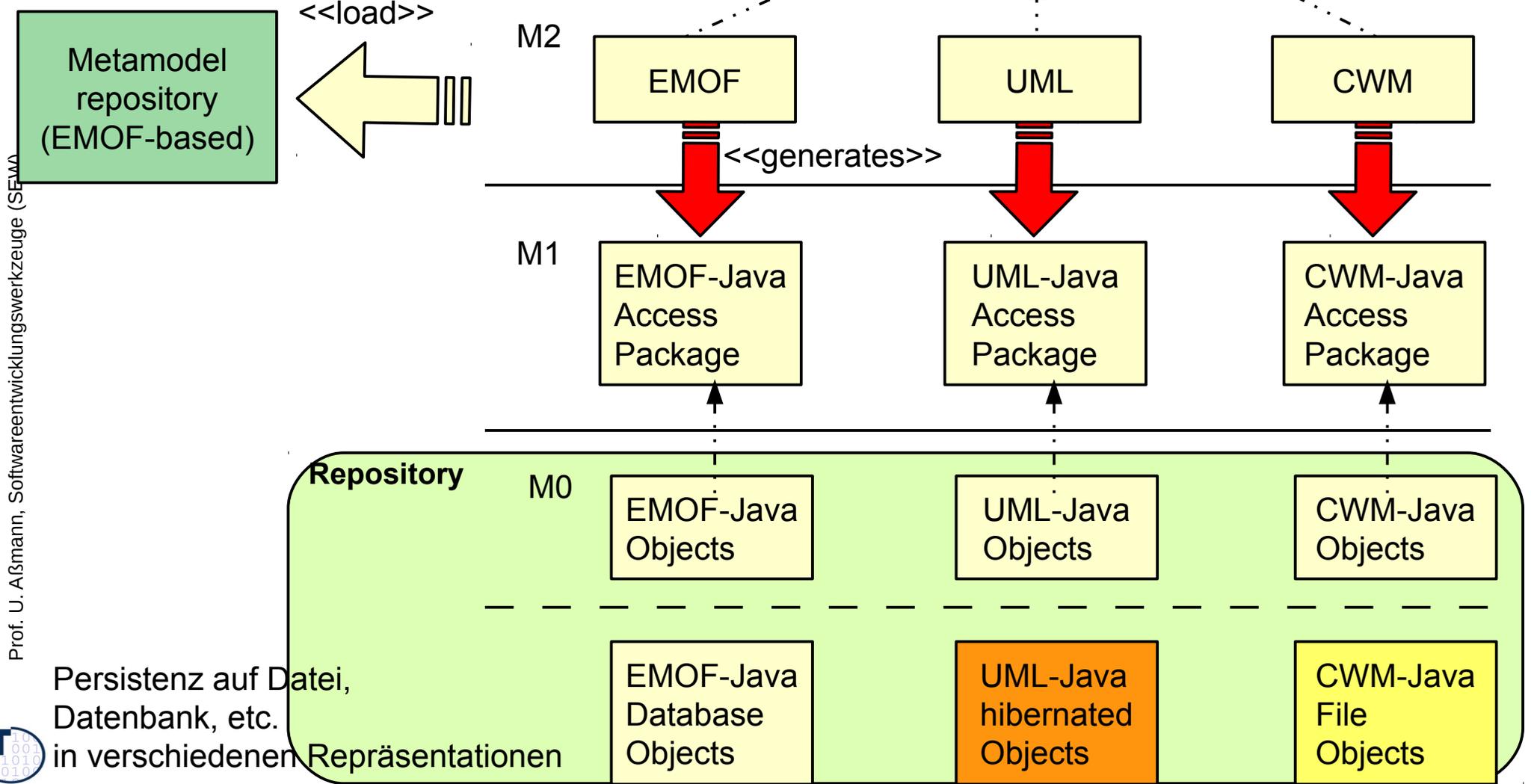
Persistenz auf Datei,
Datenbank, etc.

in verschiedenen Repräsentationen



Eclipse EMF (basierend auf EMOF)

46



Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)

Persistenz auf Datei,
Datenbank, etc.
in verschiedenen Repräsentationen



Model Management Funktionen Tools im ModelBus - Verteiltes Repository für Modelle

47

Tool	Service	Parameter	Multiplizität	Type
UML- Repository	findClass	ClassName Class	in [1..1] out [1..1]	primitiveType (String) specif cModelType (Foundation::Core::Class)
	findPackage	PackageName Package	in [1..1] out [1..1]	primitiveType (String) specif cModelType (Model_Management::Package)
UML to EJB	transform	sourceModel	in [1..*]	specif cModelType (Model_Management::Package)
		targetModel	out [1..*]	specif cModelType (EJB:: EjbComponent)
Code Generation	generateSingle Component	EJBComponent	in [1..1]	specif cModelType (EJB:: EjbComponent)
	generate Components	EJBComponent	in [1..*]	specif cModelType (EJB:: EjbComponent)

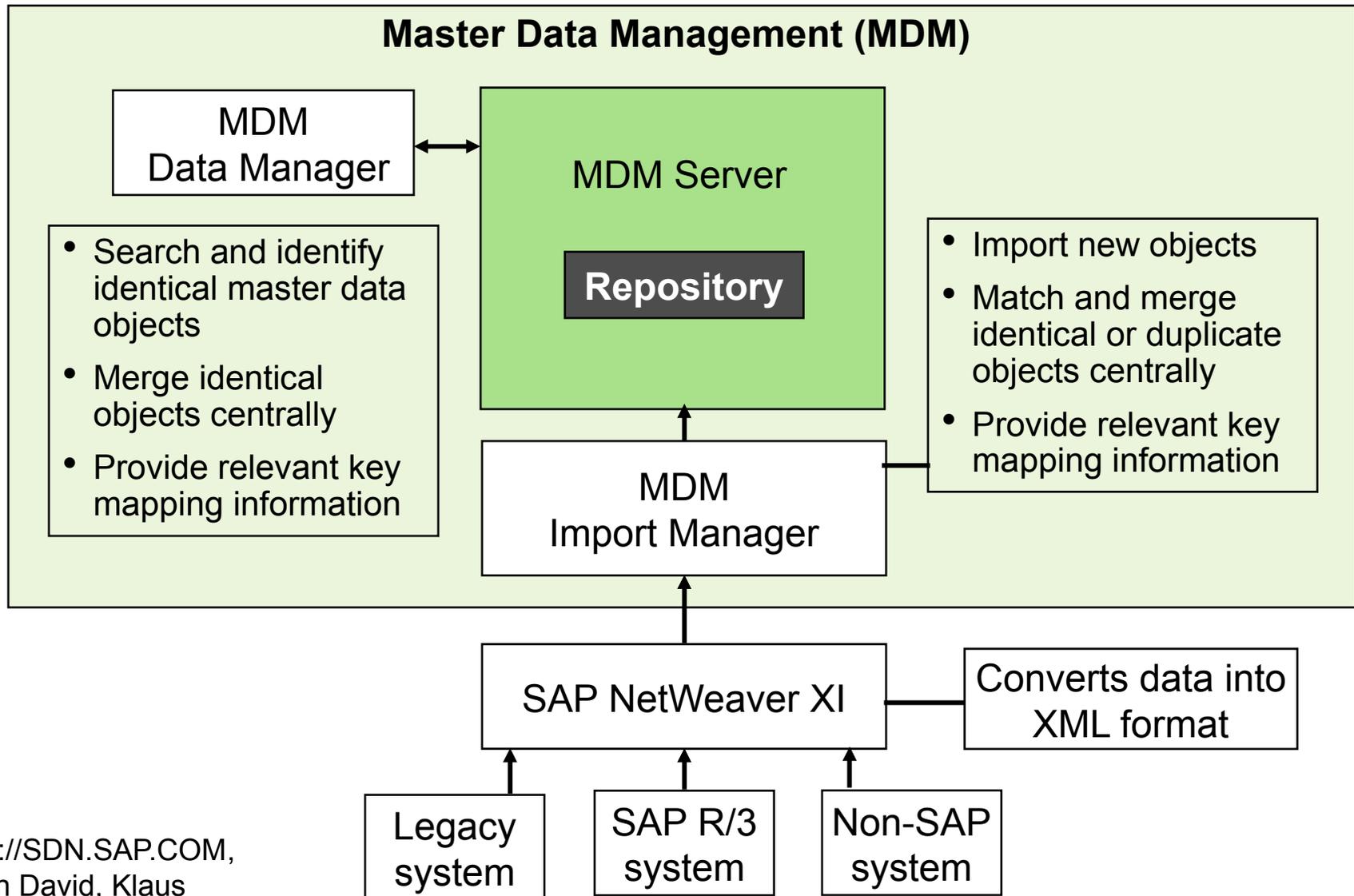
14.4.2 Master Data Management (MDM)

48

- ▶ 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

50

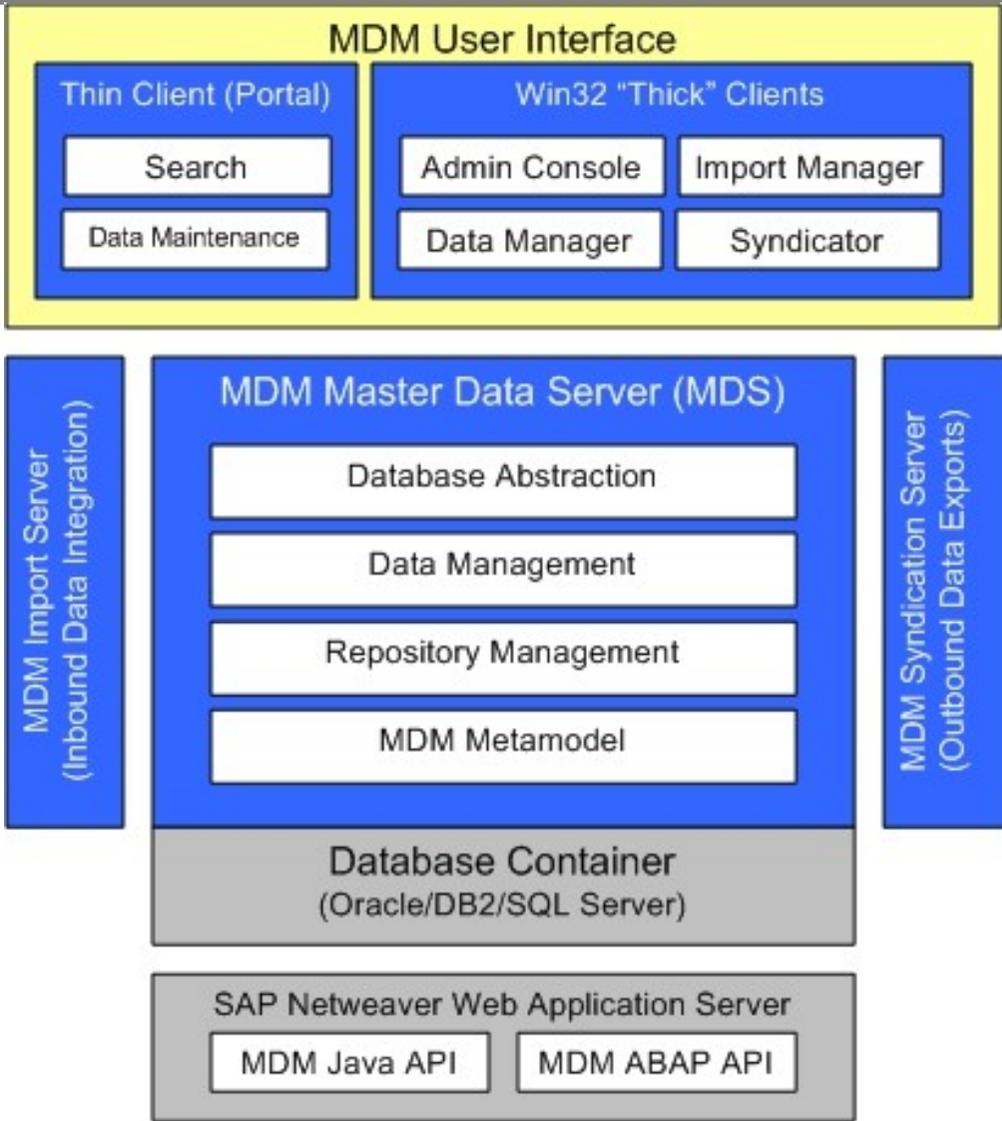


Quelle:

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

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





14.5 Extension of Repositories by Metamodel Extension

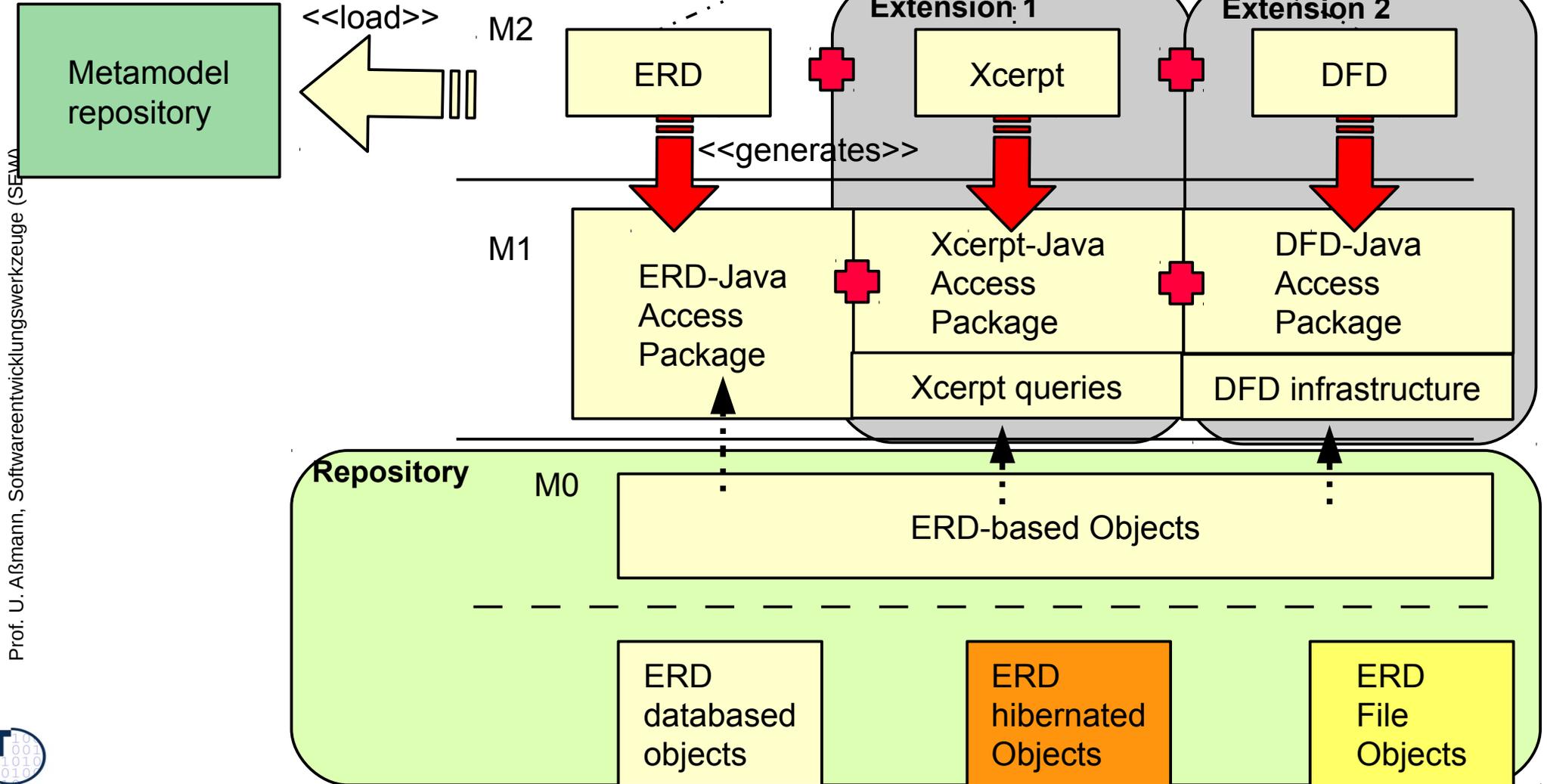


52



Extension of Metamodel-Driven Repositories

53



Warum sind metamodelgesteuerte Repositorien wichtig für Werkzeugnutzung und -bau?

Erweiterung der typisierten Zugriffsschnittstellen und Codepakete durch Erweiterung bzw. Komposition von Metamodellen

- ▶ In a metamodel-driven repository, loading of new metamodels extends the access interfaces
 - load the new metamodel
 - generate new typed access interfaces (and code) for access, query, consistency, etc
 - load new code by dynamic class loading

Für eine domänenspezifische Sprache schafft man ein Repository als eine Erweiterung des Repositories einer GPL

Für eine SEU einer Methode schafft man ein Repository durch Komposition der Repositorien der Basistechniken