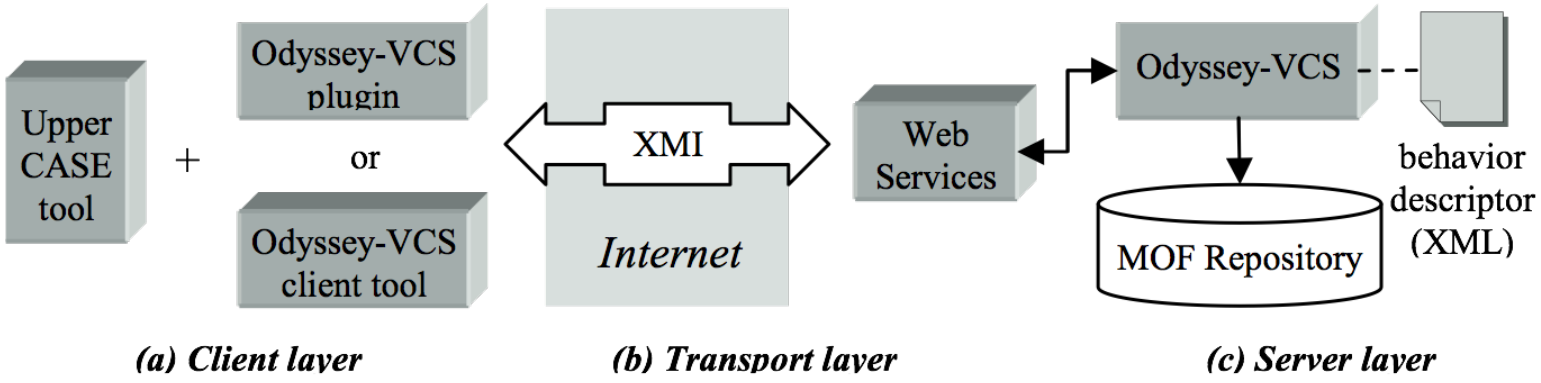
Vergleich von Model-Repositories

Seminar Modellgetriebene Softwareentwicklung
Abschlusspräsentation

Michael Siebauer

Odyssey-VCS

- **Hauptfokus:** universelle Einsetzbarkeit
- Den „klassischen“ Repositories am ähnlichsten
- Textdatei-basiert (XMI)
- Client/Server Architektur
- Eclipse-Plugin vorhanden
- Merging fast vollständing automatisiert



Odyssey-VCS

- Jedes Projekt besteht aus Vielzahl von *Konfigurationseinheiten* **(CI)**
  - müssen manuell identifiziert und konfiguriert werden
  - ordnen jedem Artefakt einen spez. Verhaltensweisen zu
- *Versionierungseinheit* (**UV**)
  - Nur für UV werden Versionsnummern vergeben
- *Vergleichseinheit* (**UC**)
  - Bestimmen „zusammengehörige“ Elemente
  - Konflikterkennung nur innerhalb UC Elementen

Jedes Modell wird in CI logisch geteilt

Jedem CI wird UV und/oder CI Eigenschaft zugewiesen

Odyssey-VCS

```
<typename=„org.omg.uml.foundation.core.UmlClass“>
    <UC>true</UC>
    <UV>true</UV>
</type>
<typename=„org.omg.uml.foundation.core.Attribute“>
    <UC>true</UC>
    <UV>false</UV>
</type>
<typename=„org.omg.uml.foundation.core.Operation“>
    <UC>true</UC>
    <UV>false</UV>
</type>
```

Odyssey-VCS Versionierung

- Homogene Versionierung (innerhalb) eines Modells durch *Baselines*
- Baseline entspricht der Version des gesamten Modells → hängt von Versionen seiner Elemente ab
- Baseline ist ein „zusammengesetztes CI“

Vorteile:

- Durch die homogene Versionierung kann gesamten Modell auch wieder CI eines noch größeren Modells sein
- Baseline ist nichts „besonderes“ sondern auch nur eine CI

Nachteil:

- Gefahr der „Explosion“ von Versionen

Unicase --- EMFstore

- EMFstore eigentlich auf Unicase zugeschnitten
- Unicase Entwicklungsumgebung mit Vielzahl(!) vordefinierter Klassen und Modelle
- Alle Klassen und Modelle sind streng hierarchisch gegliedert -> Alles intern als Graph repräsentiert

Motivation: Nachteil textbasierter VCS:

- CI Ansatz problematisch bei *Inter*-Modell Verknüpfungen
- Große Projekte → viele verschachtelte CI → Komplexer und hoher Merge Aufwand

Nachteile zustandsbasierter VCS:

- Skalieren schlecht
- Änderungen können sich überdecken und unentdeckt bleiben
  - Kleine Modell-Änderungen → große MetaModell-Änderungen
  - Bleiben evtl. unentdeckt wenn nur aktueller Zustand des Modell-Elements betrachtet wird

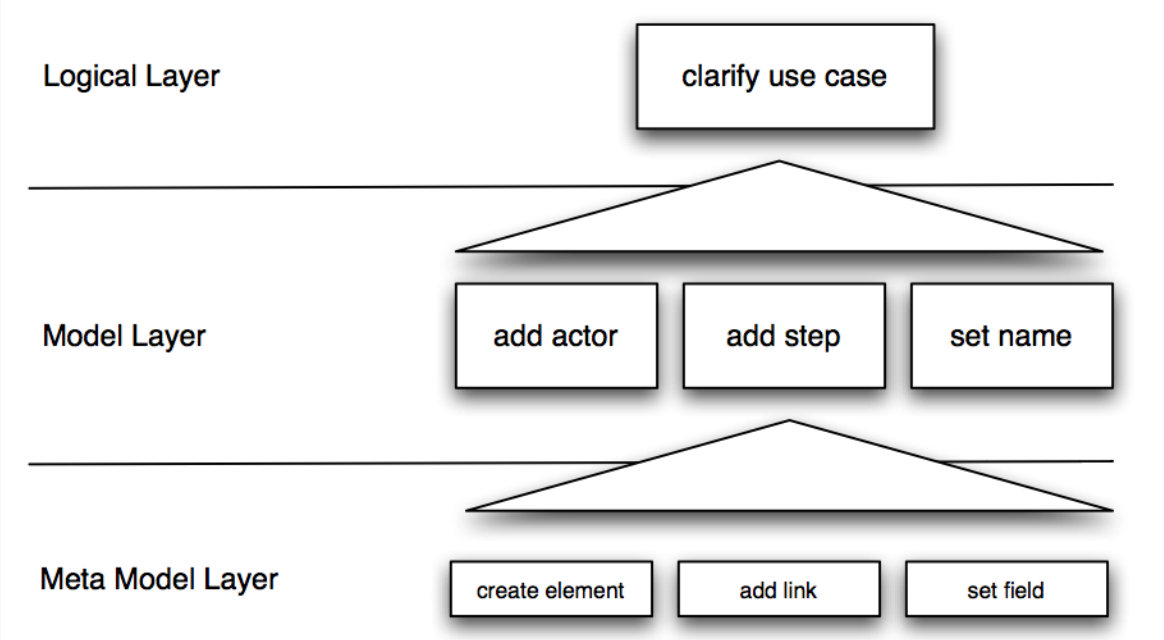
Unicase --- EMFstore

Daher hier: **Operations-basierte Versionierung**

- Speicherung der exakten Änderungsoperationen des Editors!
- Modell Elemente *benachrichtigen* Editor über deren Editierung (transparent vor dem Benutzer)
- Editor speichert (und normiert) alle Änderungsoperationen bis zum nächsten Check-in
- Delta ist geordnete Sequenz von Änderungsoperationen
- Operationen müssen jedoch gewissen Einschränkungen unterliegen (z.B. kein Suche-Ersetze operation)
- Jeder Zustand des Modells kann durch vorwärts oder rückwärts Abspielen der Deltas rekonstruiert werden

Unicase --- EMFstore

- Erfassung der Änderungsoperationen auf 3 Ebenen
- **Logik-Ebene:** welche Änderungen gehören logisch Zusammen und warum → Aufgabe des Nutzers (Log-Messages)
- **Modell-Ebene**: eigentliche Änderung durch den Nutzer → erzeugt Menge von Meta-Modell Änderungen
- **Meta-Modell-Ebene**: atomare Änderungen an konkreten Modell Attributen → vor Anwender meist verborgen



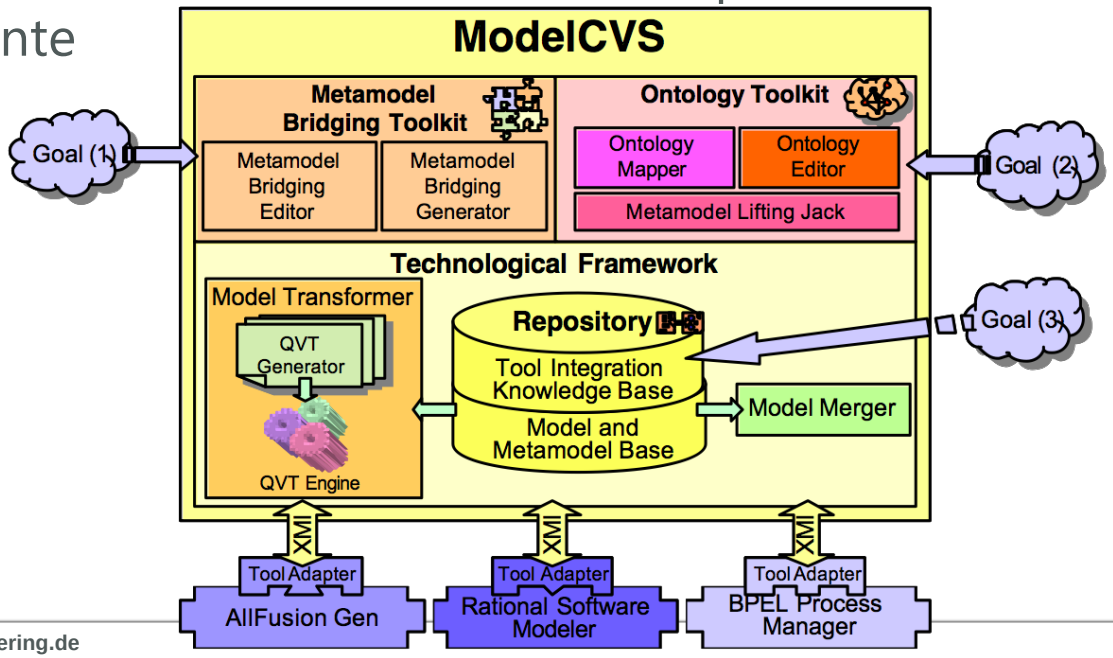
Unicase --- EMFstore Versionierung

- **Produkt-Versionierung**
  - Nur eine **globale** Version des Modells
  - ALLE Elemente des Modells haben dieselbe Version!
  - Nur eine gültige Konfiguration pro Modell Version
  - Änderungssätze können direkt alle ModellElemente betreffen → Änderungspaket
  - Aber: „Varianten“ eines ModellElements schlecht realisierbar

- Alle Änderungspakete werden in Baumstruktur gespeichert.
- Änderung von Quell-Version zu Ziel-Version entspricht Traversierung des Baumes → Abspielen aller Änderungssequenzen auf dem Weg
- Merging-semiautomatisch: Nutzer muss entscheiden welche Modell-Ebenen Änderungen verworfen werden müssen (nicht trivial!) → EMFstore erzeugt automatisch gültige MetaModell-Änderungssequenzen

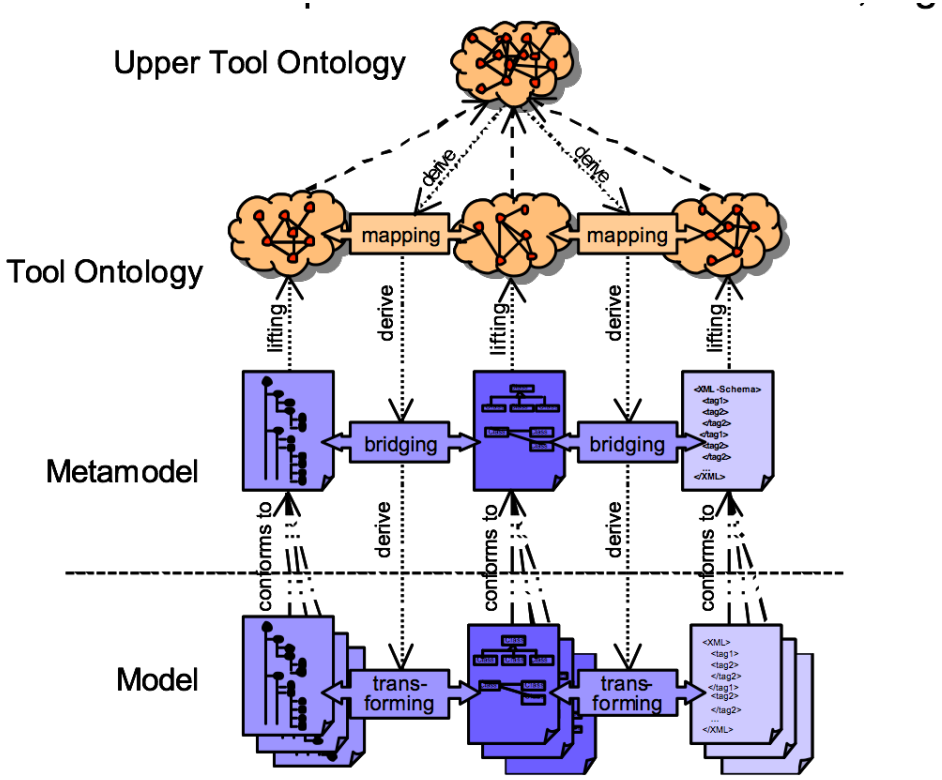
Model-CVS

- Kein eigentliches Tool → eher Idee einer universellen Architektur
- **Hauptfokus**: Interoperabilität *„Modell-basierte Werkzeug Integration“*
- Modelle werden nicht nur versioniert sondern in jedes andere Format transformiert
- **Versuch vorhandenes Wissen aus Gebiet der Wissens-repräsentation auf Modelle zu adaptieren (Schema-Matching, Ontologie, …)**
- Versionierung leider nur in Ansatzweise definiert (intern CVS oder subversion als backbone + nicht näher spezifizierte *ModelMerger* Komponente



Model-CVS

- **Idee:** Durch ModelLifting (syntaktisches) MetaModel in höhere (semantische) Abstraktionsebene zu „liften“ (Ontologie bilden)
- Mappings zwischen Ontologien finden und daraus Brücken zwischen MetaModellen ableiten
- Aus Brücken wird abstrakte Transformationssprache (QVT) generiert



ModelCVS - Fortsetzung

- Mittels (community) Wissensbasis vorhandenes Wissen sammeln und wiederverwenden
- Heuristiken helfen Mappings zu finden

Integrations Muster:

- Translation: semantisch äquivalente Konzepte werden übersetzt
- Alignment: keine Übersetzung, eher Relation zu einander (Bsp. Löschen des Einen, führt zu Löschung des in Relations Stehenden)
- Modularisierung: Aufspaltung der Modelle in (logische) Teilmodelle → leichter Integrierbar

Konfliktdetektion:

- mögliche Konfliktszenarios a priori während Lifting berechnet
- Detektion dann beim check-in
- Unterscheidung: Semantischer vs. Syntaktischer Konflikt
  - Syntaktisch: Graphenstruktur des (M)Model
  - Semantisch: Bedeutung von (syntaktischen) Modell Elementen

Vergleich

- Internes Datenformat
- Anwender Unterstützung
- Arbeitsplatz Integration
- Deltas
- Konfliktbehandlung
- Versionierung
- Verteiltes Arbeiten
- Rechte Kontrolle
- Sonstiges

Vergleich

- **Datenformat:**
  - Odyssey: Text-basiert (XMI)
  - Unicase: Graphen-basiert
  - ModelCVS: Text-basiert

- **Anwenderunterstützung:**
  - Odyssey: ▯ Nur Versionierung
  - Unicase: ▯Allround IDE
  - ModelCVS: ▯ nur (theoretisches) Framework

- **Arbeitsplatz intergration:**
  - Odyssey: ▯ Eclipse Plugin oder Standalone Client
  - Unicase: ▯ Eclipse
  - ModelCVS: ▯ universell

Vergleich

- **Deltas / Granularität**
  - Odyssey: eher grob (abhängig von UV, UC Modellierung)
  - Unicase: sehr fein (operationsbasiert)
  - ModelCVS: vermutlich sehr grob (zeilenvergleich)

- **Konfliktbehandlung:**
  - Odyssey: Detektion u.U. schlecht, Merging einfach
  - Unicase: Detektion garantiert, Merging knifflig
  - ModelCVS: Detektion gut aber aufwendig, Merging nicht entwickelt

- **Versionierung:**
  - Odyssey: Zustandsbasiert
  - Unicase: Operationsbasiert
  - ModelCVS: Zustandsbasiert

Vergleich

- **Verteiltes Arbeiten:**
  - Odyssey: ✓ möglich
  - Unicase: ✗ ausschließlich Unicase
  - ModelCVS: ✓ Ja, nach Integration

- **Rechte Kontrolle:**
  - Odyssey: ✗ Nein
  - Unicase: ✓ Ja
  - ModelCVS: ✗ (vermutlich) nicht

- **Sonstige Anmerkungen:**
  - Odyssey: Modelle müssen immer im- und exportiert werden
  - Unicase: Ausschließlich auf Unicase zugeschnitten → Probleme beim auschecken
  - ModelCVS: Overkill

Fazit

Anwendungsbereiche:

- **Für kleine überschaubare Projekte → Odyssey**
  - Schnell und leicht einsetzbar
  - Einmalige Konfiguration aufwendig
- **Für große Projekte → Unicase**
  - Unterstützt Modellierung für viele Bereiche
  - Keine extra Konfiguration nötig
  - Man wird Werkzeug abhängig → Import fremder Modelle unmöglich?
- **Für Forschungsprojekte → ModelCVS**
  - Schier begrenzte universelle Einsetzbarkeit
  - Mit (überschaubarem) Aufwand Transformation von Modellen in jegliche Formate
  - Nur Theoretisches Framework im Moment
  - Gute zukunftsprognose

Persönliches Fazit

Danke für die Aufmerksamkeit