Java Forum Stuttgart 2003

03. Juli 2003

pleXX

Ein Framework auf dem Weg zur MDA

Inhalt

- MDA
  - Ursprung
  - Modellgetriebene Entwicklung
  - Methodik
- pleXX
  - Überblick
  - Historische Entwicklung von pleXX
- pleXX vs. MDA
- Weiterentwicklung von pleXX

MDA - Ursprung

- Model Driven Architecture
- Standard der OMG (Object Management Group)
  - CORBA
  - UML
  - MOF
  - CWM
- Ziele:
  - Portabilität
  - Interoperabilität
  - Wiederverwendbarkeit

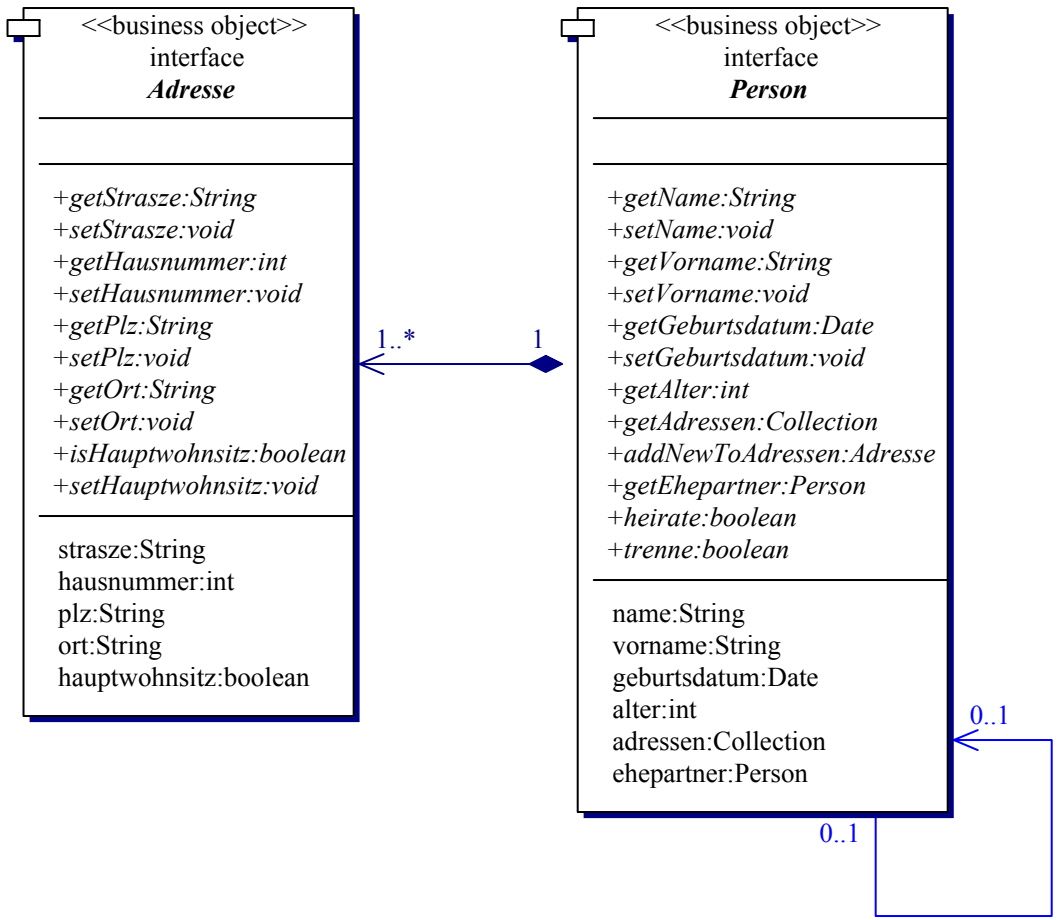
Modellgetriebene Entwicklung

- Ausgangspunkt ist das Modell der Anwendung (UML)
  - Geschäftsobjekte
  - Eigenschaften (Attribute)
  - Beziehungen
  - Verhalten
- Nur fachliche, keine technischen Aspekte

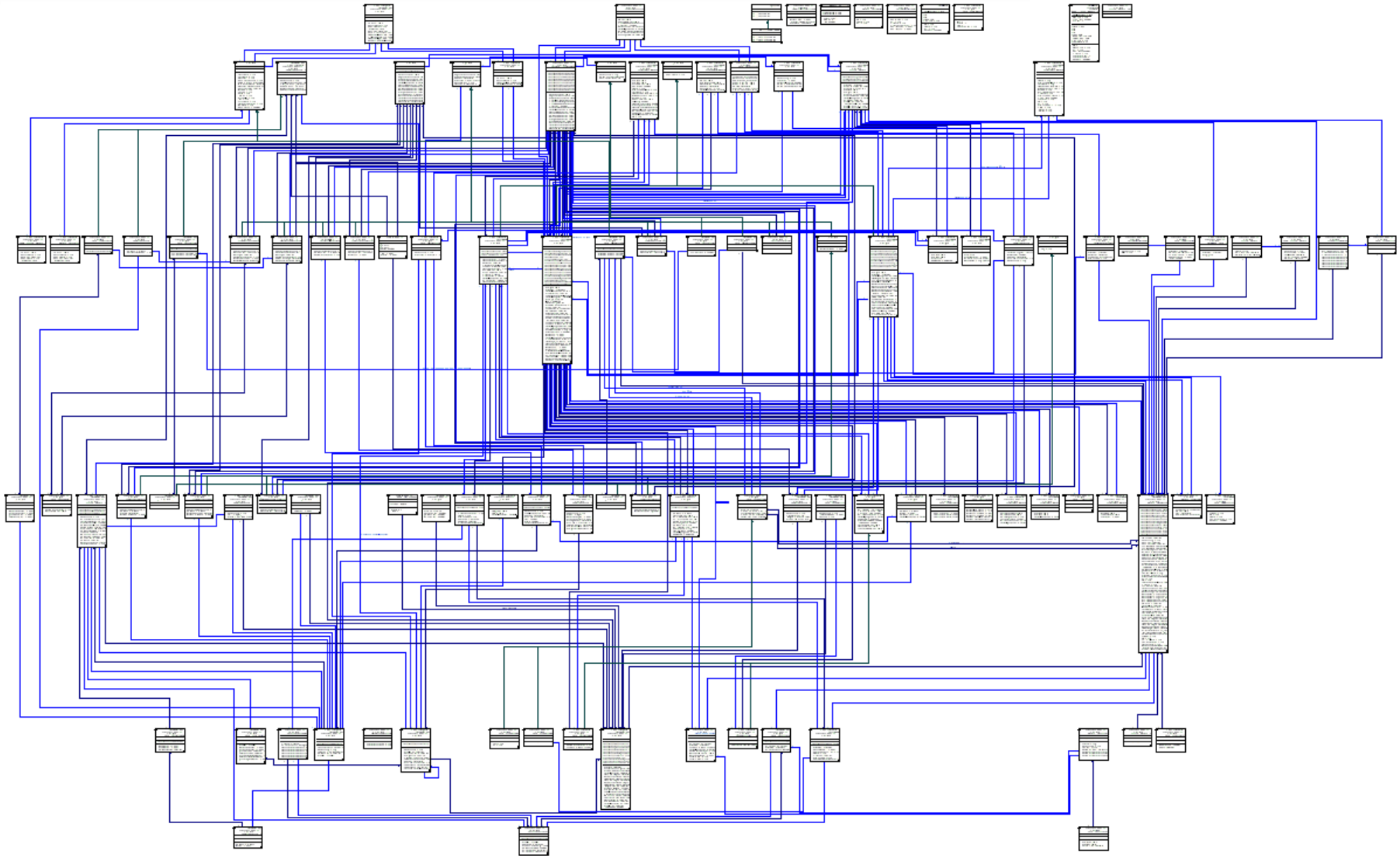
- Problem: Überführung bzw. Verwendung des fachlichen Modells in der technischen Anwendung

- Lösung:
  - Anreichern durch Metainformationen
  - Generativer Ansatz

Beispielmodell (1)

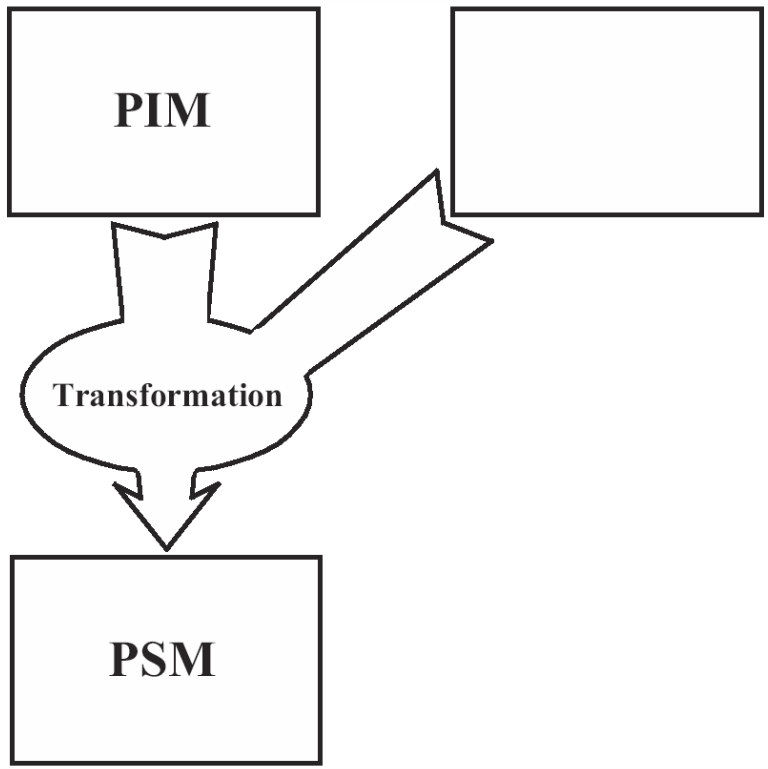


Beispielmodell (2)



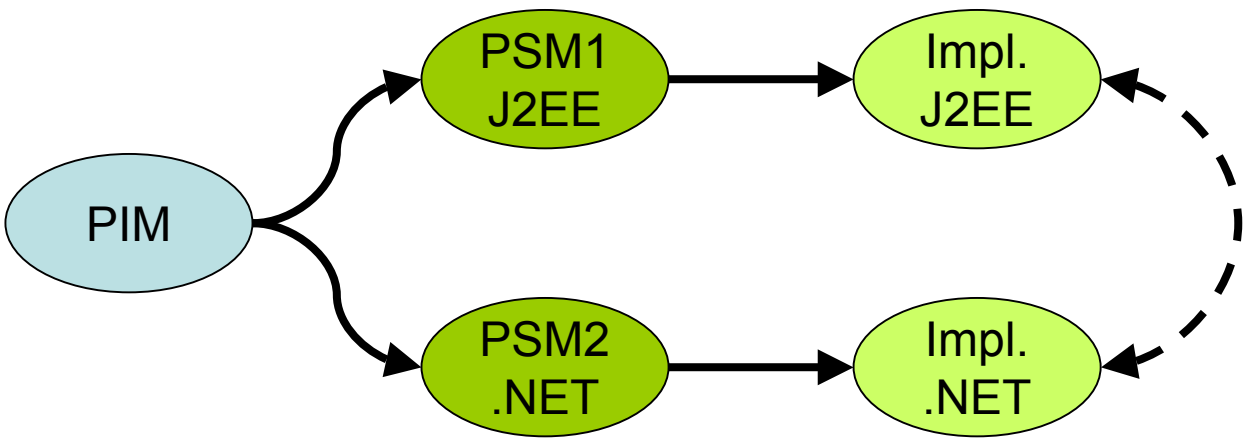
MDA - Methodik

- Computation Independent Model (CIM)
  - Modell ohne Bezug zur Struktur oder Verarbeitungsweise des Systems
- Platform Independent Model (PIM)
  - Modell ohne Bezug zu einer konkreten Plattform
- Platform Specific Model (PSM)
  - Spezifisches Modell für eine konkrete Plattform
- Überführung der Modelle durch Anreicherung und Transformation
  - Manuell
  - Semi-Automatisch
  - Automatisch



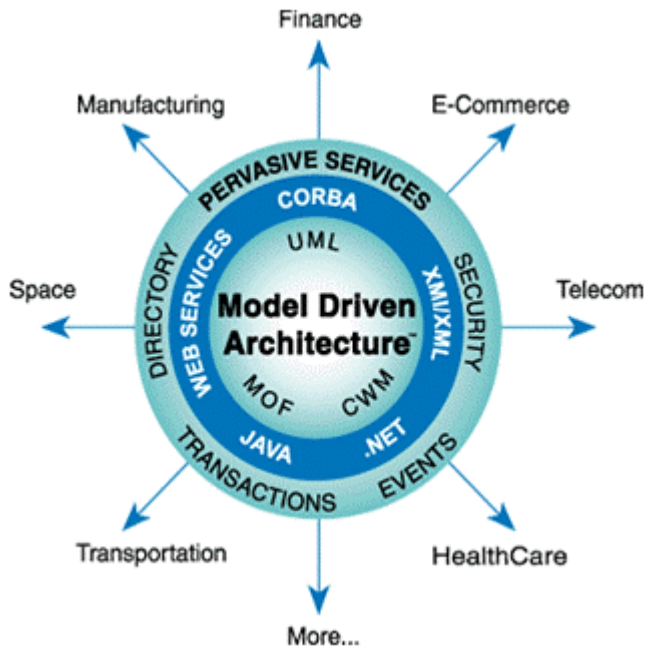
MDA - Einsatz

- Mehrere PSMs pro PIM
- Dadurch
  - Portabilität
  - Interoperabilität
  - Wiederverwendbarkeit



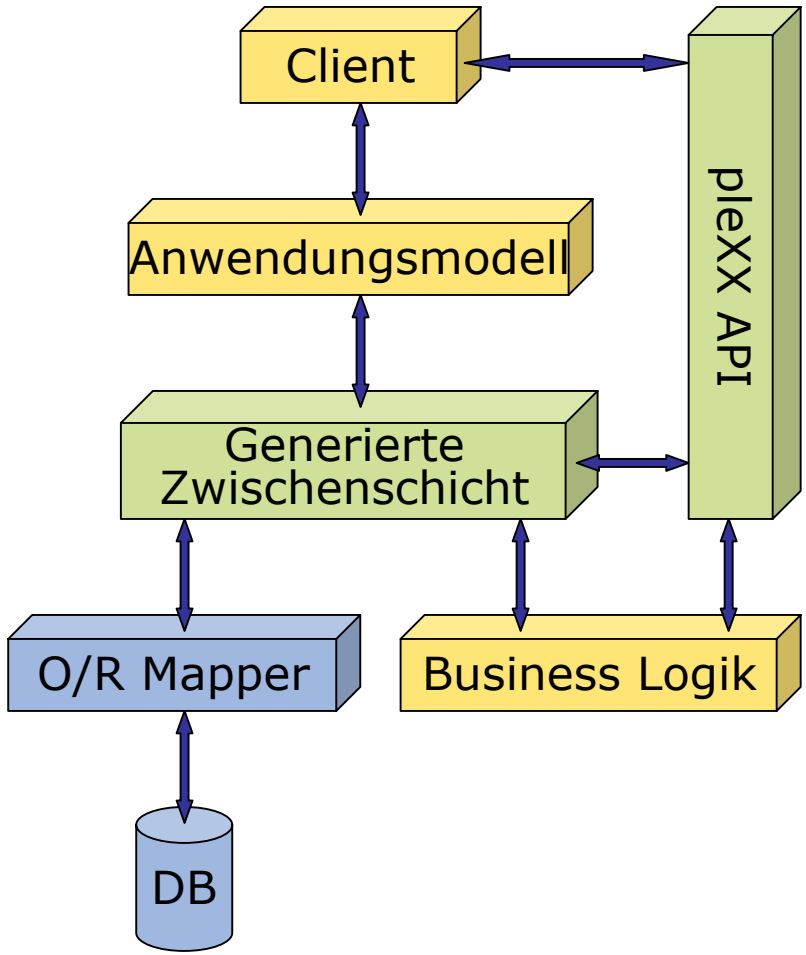
MDA - Vision

- Modell als Kern
- Transformation in reale Software
  - Unterstützt durch standardisierte „Profiles“
- Interoperabilität durch standardisierte „Pervasive Services“
- Wiederverwendung durch standardisierte „Domain Facilities“

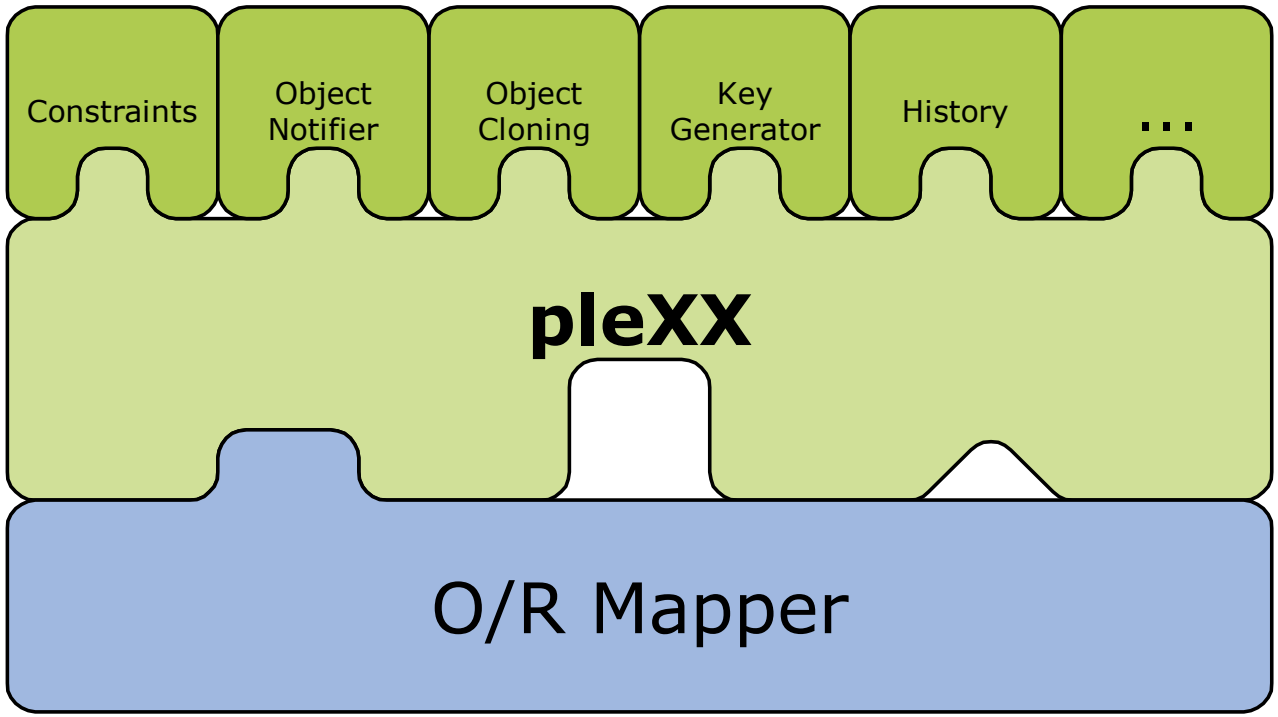


pleXX (1) – Überblick

- Allgemeines Framework für die Modellierung und Abbildung persistenter Business Objekte
- Richtlinien zur Modellierung
  ➔ logische Trennung von Client und Business Logik
- Einheitliche Programmierschnittstelle
- Mehrwertdienste im Framework
- Abstraktionsschicht zwischen der Java-Anwendung und dem eigentlichen Persistenzmechanismus
- Codegenerator für die Basis-implementierung und Anbindung an den Persistenzmechanismus



pleXX (2) – Features



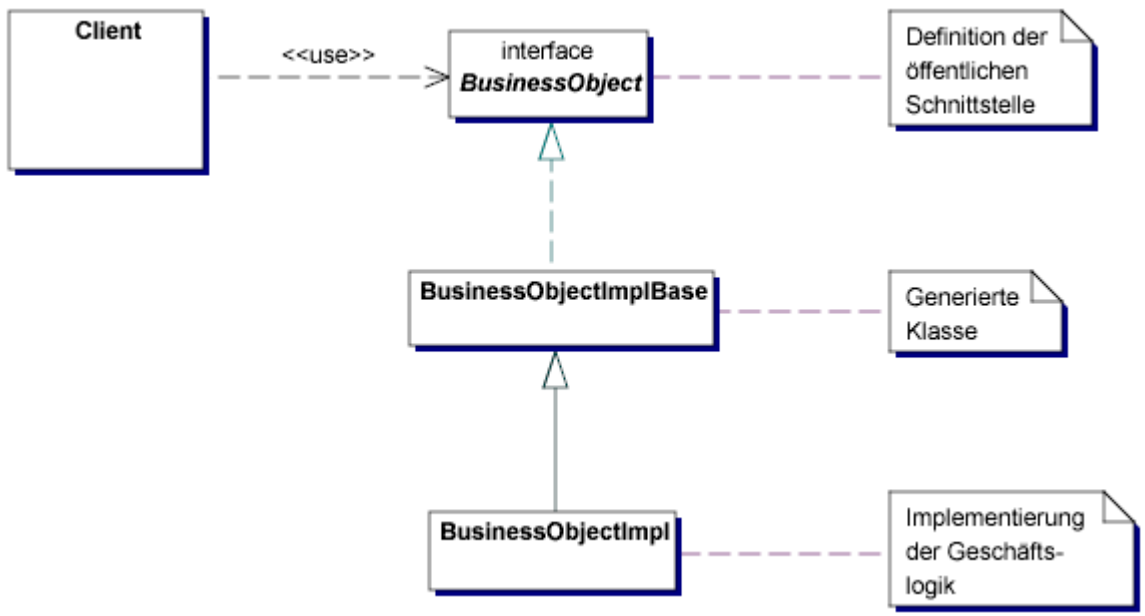
- Austauschbarer O/R Mapper
- Constraints
- Change Notifier
- Selbstbeschreibende Objekte
- Clonen von Objektgraphen
- KeyGenerator
- Bidirektionale Beziehungen [zur Laufzeit]
- Historisierte Objekte

pleXX (3) – Am Anfang war...

- Das Problem:
  - Projekt steht an
  - Unklar, welcher O/R Mapper verwendet werden kann/soll
- Die Idee:
  - pleXX als „persistence Layer, eXXcellent"
  - Persistenz nicht selbst implementieren
  - Eigene Schnittstelle zur Transaktionssteuerung
- Die Hürde:
  - Unabhängigkeit der Anwendungsimplementierung vom O/R Mapper?

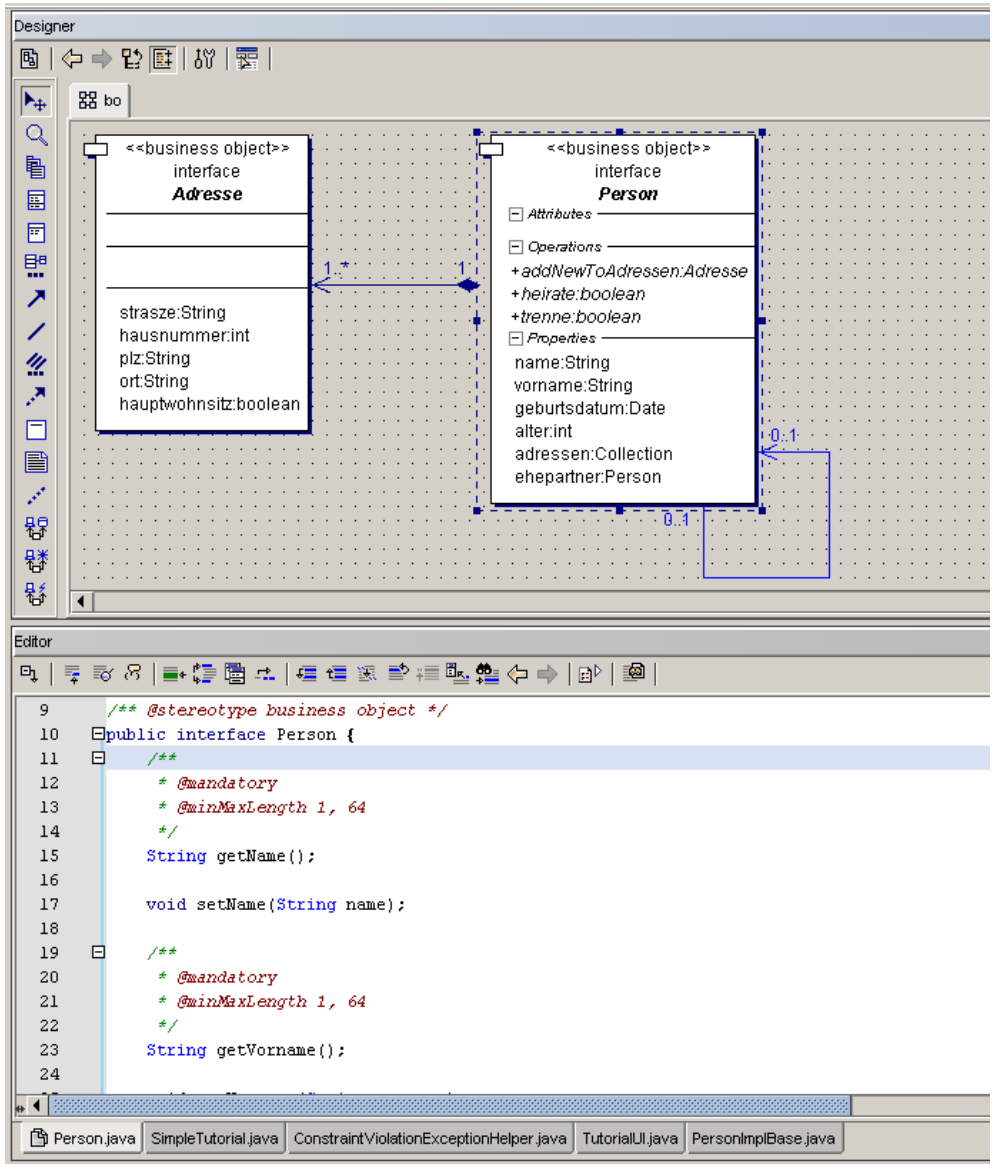
pleXX (4) – Mehrschichtarchitektur

- Logische Trennung der einzelnen Schichten durch
  - Einsatz von Interfaces
  - Generativen Ansatz
  - „Generation Gap“ Pattern



pleXX (5) – Together

- Die nächsten Fragen:
  - Wo ist das Modell?
  - Wie wird der Generator implementiert?
- Die nächste Lösung:
  - Together
- Die Vorteile:
  - Langjährige Erfahrung
  - Single-Source Prinzip
  - Umfangreiche API
- Das Resultat:
  - Rasche Entwicklung des Generators
  - Schnittstelle = Modell



pleXX (6) – Features!

- Constraints zur Sicherstellung der Korrektheit der Daten
- Constraint-Typen:
  - Attribut-Constraints zur Prüfung einzelner Felder
  - Relation-Constraints zur Prüfung von Objektbeziehungen
- Zeitpunkt der Prüfung:
  - Sofort beim Setzen des Wertes bzw. Hinzufügen oder Entfernen einer Objektrelation
  - Jederzeit manuell getriggert
  - Spätestens beim Abschluss der Transaktion
- Constraint-Klassen werden mit Attributen bzw. Relationen assoziiert (Together Properties)
- Durchführung der Prüfung an der richtigen Stelle zum richtigen Zeitpunkt durch pleXX Code Generator und Runtime

pleXX (7) – Mehr Features!

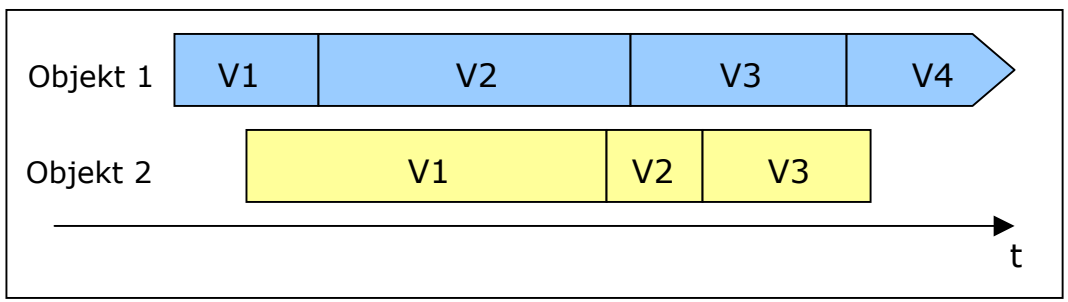
- Änderungsbenachrichtigung: ObjectChangeNotifier
- Objektbeschreibung: Describable
- Kopieren von Objektgraphen: ClonableObject
- Bidirektionale Beziehungen

```
/**
 * @stereotype business object
 * @description Satzung
 * @objectDescription "Gültig ab: " + getValidFrom()
 */
public interface Satzung extends ObjectChangeNotifier, CloneableObject, Describable, Historizable {
    /**
     * @associates <{de.iiru.agv.basistypen.Muellverbandsgemeinschaft}>
     * @supplierCardinality 0..1
     * @constraint de.iiru.agv.satzung.constraints.StandardMuellverbandsgemeinschaftConstraint
     * @description Standard Müllverbandsgemeinschaft
     */
    Muellverbandsgemeinschaft getStandardMuellverbandsgemeinschaft();

...
}
```

pleXX (8) – History

- Gültigkeit und Zustand von Objekten begrenzt von bestimmten Zeitintervallen
- Beispiel: Die Telefonnummer einer Person war vom 01.01.2001 bis zum 30.06.2002 „555-32 34 36“ und seit dem 01.07.2002 ist sie „555-23 43 63“
- Der Zustand eines Objekts ist also abhängig vom Zeitpunkt, den man betrachtet
- Nomenklatur: Objekt, Version



pleXX (9) – Neuer Generator

- Starke Abhängigkeit von Together (API)
- Schlechte Performance des Generators (7 Minuten für 94 BOs)
- Komplette Neuimplementierung
  - Direktes Quelltextparsing
  - Eigenes internes Metamodell
- Ergebnis: 15 Sekunden für 94 BOs

pleXX vs. MDA (1) – Konformität

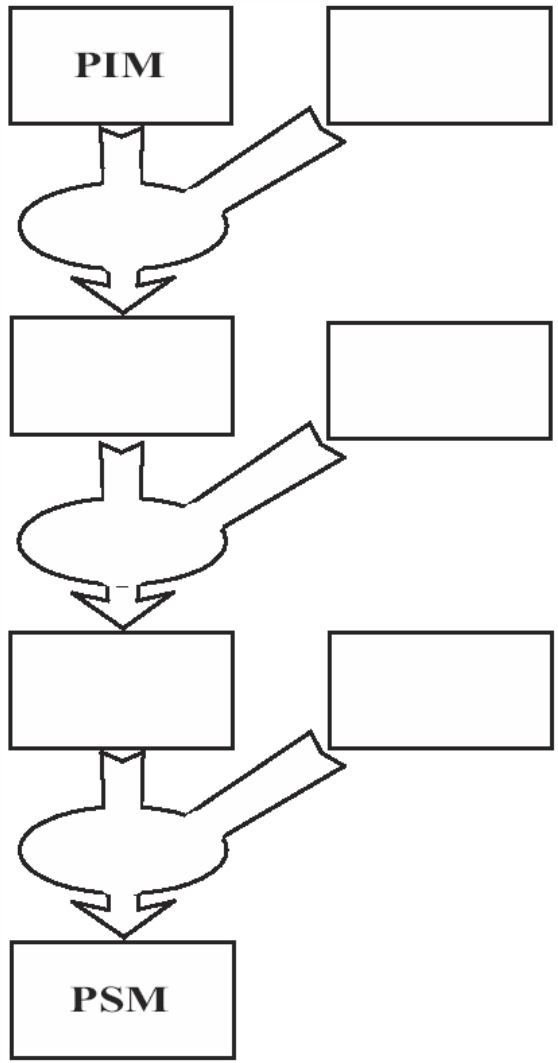
- Ist pleXX MDA?
- Ja!
  - UML-Modell aus Ausgangspunkt
  - Anreicherung durch Meta-Informationen
  - Generativer Ansatz
- Nein!
  - Modell ist kein „Modell" sondern Quelltext
  - Festlegung auf Java und O/R Mapper
  - Benötigt Laufzeitumgebung

pleXX vs. MDA (2) - Plattform

- Der Streitpunkt: Was ist eine Plattform?
- Abhängig von
  - Standpunkt
  - Abstraktionsgrad
- Verschiedene Plattformen:
  - Java vs. C#
  - J2EE vs. .NET
  - Entity Beans vs. O/R Mapper
  - intelliBO vs. TopLink

pleXX vs. MDA (3) - Variation

- Mehrfache Anwendung des MDA Patterns
- Transformationen von Plattform zu Plattform



pleXX vs. MDA (4) – Fazit

- Die Grundidee ist identisch
  - Modellieren der fachlichen Aspekte
  - Generieren der technischen Aspekte

- Pragmatischerer Ansatz:
  - Fachmodell hat Bezug zur Programmiersprache (Java)
  - Generierung nur in eine Programmiersprache (Java)
  - Vorgabe: Datenhaltung mit O/R Mappern
  - Business Logik wird implementiert, nicht generiert

Nächste Schritte

- Ein weiterer Abstraktionsgrad
  - „Richtige“ Modellierung
  - Export in XMI
  - Transformation in Interfaces
  - Weiterverarbeitung durch bestehenden Generator
- Constraints in OCL (Object Constraint Language)

- Reaktion auf Technologiewechsel
  - „The next best thing“ nach O/R Mapper?
  - Eine neue Programmiersprache?

Vielen Dank für die Aufmerksamkeit

- Weitere Informationsquellen:

- MDA
  - http://www.omg.org/mda
  - http://www.openmda.de

- pleXX
  - JavaSpektrum 01/2003, 04/2003
  - http://www.exxcellent.de
  - a.demelt@exxcellent.de

- ...oder jetzt im Anschluss!