Model Driven Architecture

02. Dezember 2003

Juri Urbainczyk

iteratec

Übersicht

- MDA – Warum und wozu?
- MDA – Theorie
- MDA – Lessons Learned
- MDA – Vor- und Nachteile
- MDA – Fazit

iteratec

MDA – Warum und wozu?

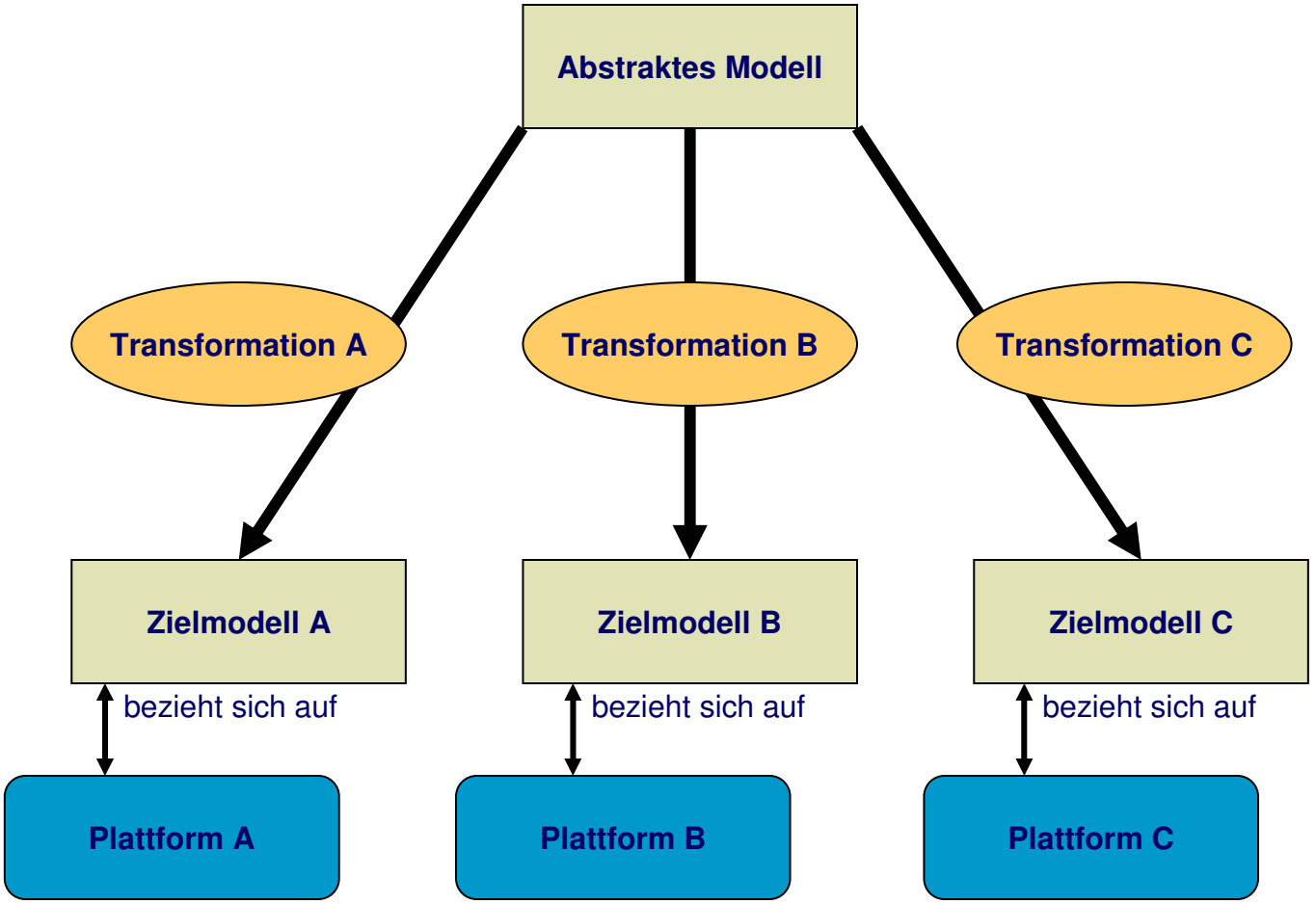
iteratec

Grundgedanke von MDA

- Ein Softwaresystem wird durch *aufeinander aufbauende Modelle* mit *abnehmendem Abstraktionsgrad* beschrieben und entwickelt.
- Anspruch: Nur das erste Modell wird manuell erstellt, alle weiteren Modelle und der Code werden *generiert*.
- MDA baut auf austauschbaren „Plattformen“ auf.

iteratec

Grundgedanke von MDA



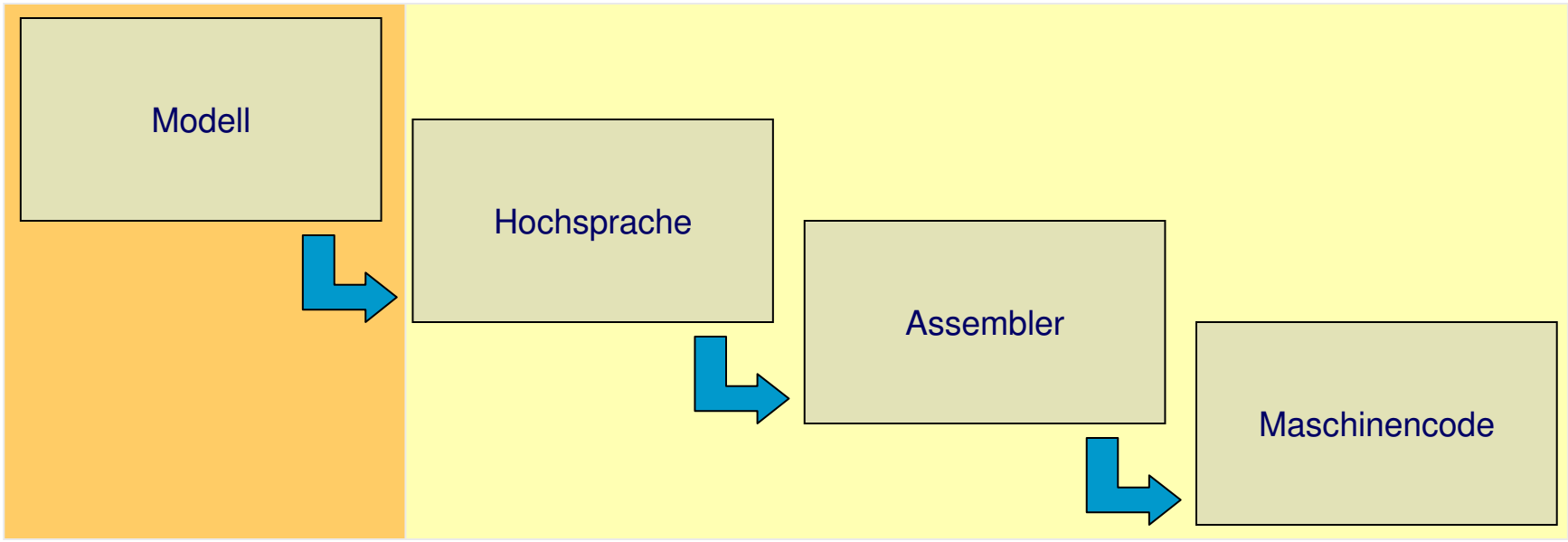
Warum MDA: Schutz vor Änderungen

- Änderungen der Technologie verursachen Probleme.
- Investitionen müssen vor Änderungen geschützt werden.
- Technologieabhängige Bereiche werden zusammengefasst und so weit wie möglich generiert.
- Bei Änderung der Technologie wird nur der Generator angepaßt.
- Abstrakte Teile des Systems können wiederverwendet werden.

iteratec

Warum MDA: Erhöhte Produktivität

- Durch MDA wird die Softwareentwicklung auf eine weitere „abstraktere“ Ebene überführt (neues Paradigma):



- MDA macht die Komplexität moderner Softwareprojekte beherrschbar.

Was ist MDA?

- Model Driven Architecture (MDA) ist ein Standard der Object Management Group (OMG).
- MDA stellt *Modelle* und *Generatoren* in den Vordergrund.
  - Modelle: von Design- zu Programmierungs-Artefakten.
- Ansätze für Modelltransformationen werden vorgegeben.
- MDA liefert Metamodelle für Standardplattformen (EJB, CORBA).
- MDA stützt sich auf verschiedene OMG-Standards:
  - Für alle Modelle wird **UML** verwendet.
  - MDA nutzt die Erweiterbarkeit der UML (*Profiles*, *Stereotypes* und *Tagged Values)*.
  - **MOF** zur Beschreibung von Metamodellen.
  - **XMI** zum Austausch von Metadaten.

Was ist MDA nicht?

- MDA ist keine neue Architektur!
  - Die Architektur hängt u.a. von den verwendeten Plattformen ab.
- MDA ist kein neuer Softwareentwicklungsprozess!
  - MDA kann mit verschiedenen Prozessen kombiniert werden.
  - MDA definiert nicht, wie Anforderungen und Architektur gewonnen werden!
  - MDA macht keine Aussagen über Qualität!
- MDA ist keine neue Modellierungstechnik!
  - MDA stützt sich auf vorhandene Modellierungsstandards.

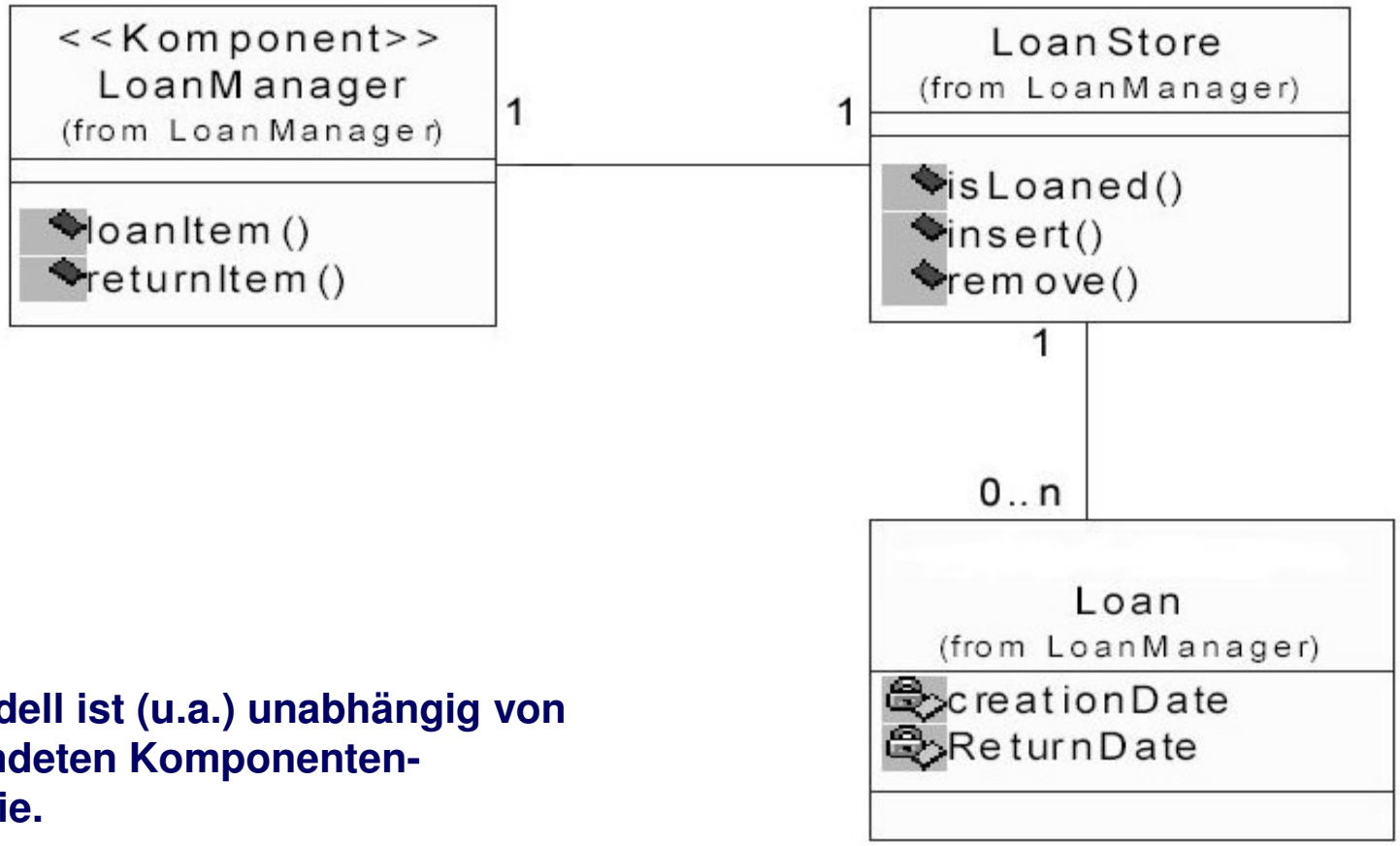
MDA – Theorie

iteratec

Die Modelle

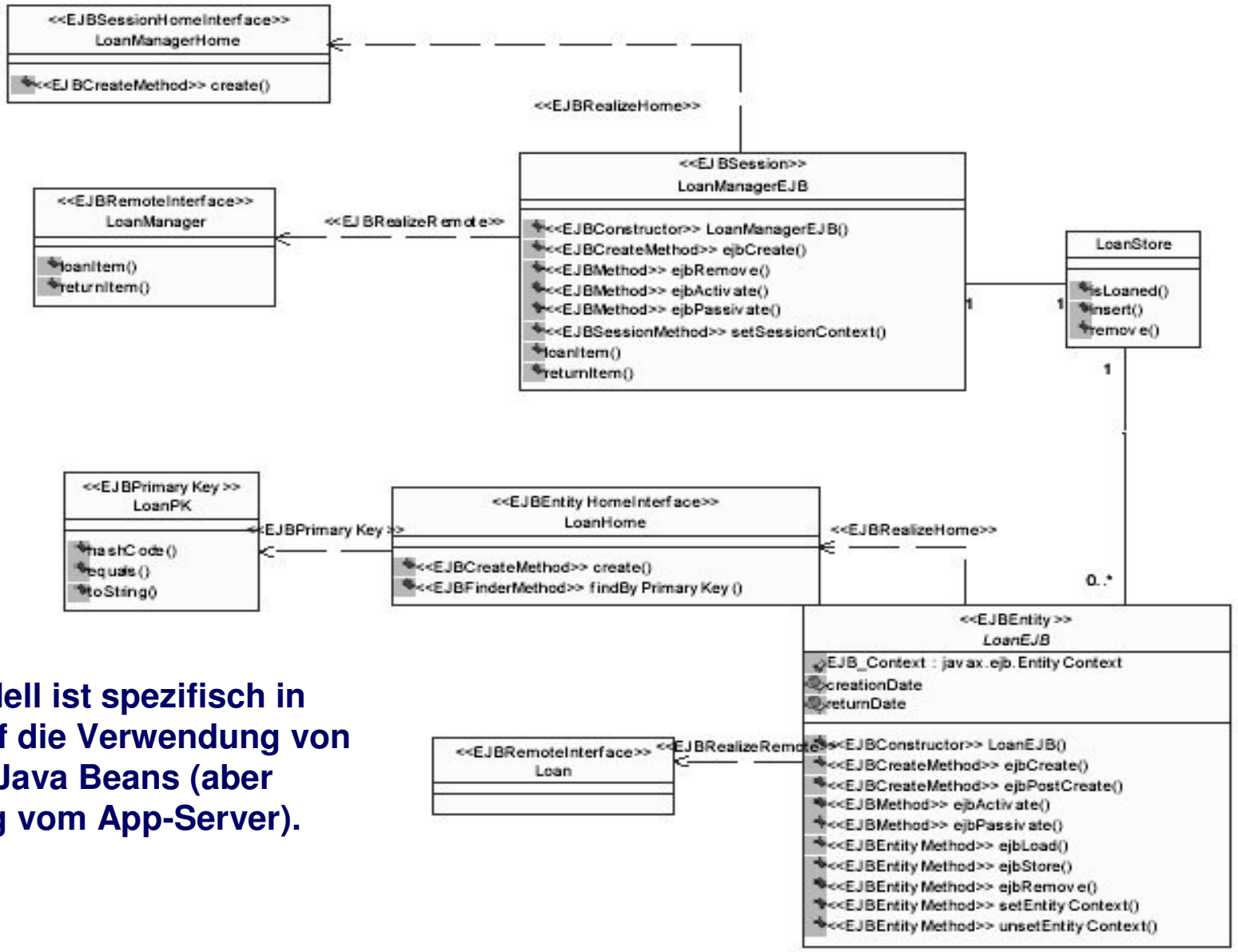
- CIM - *Computation Independent Model*
  - Ein „Domain Model“, beschreibt die Umgebung des Systems.
  - Enthält die Anforderungen an das System.
  - Spielt bei MDA untergeordnete Rolle.
- PIM - *Platform Independent Model*
  - Verfeinerung des CIM.
  - Abstrahiert von Details spezifischer Plattformen.
  - Enthält technische Details (z.B. Navigierbarkeit, Key Attribute).
- PSM - *Platform Specific Model*
  - Enthält alle für die Plattform notwendigen Informationen.
  - Adaptiert an mindestens eine bestimmte Technologie oder ein Produkt (Programmiersprache, Middleware, Datenbank, App-Server, ...)
  - Wird durch eine *Transformation* aus dem PIM gewonnen.

Beispiel - PIM



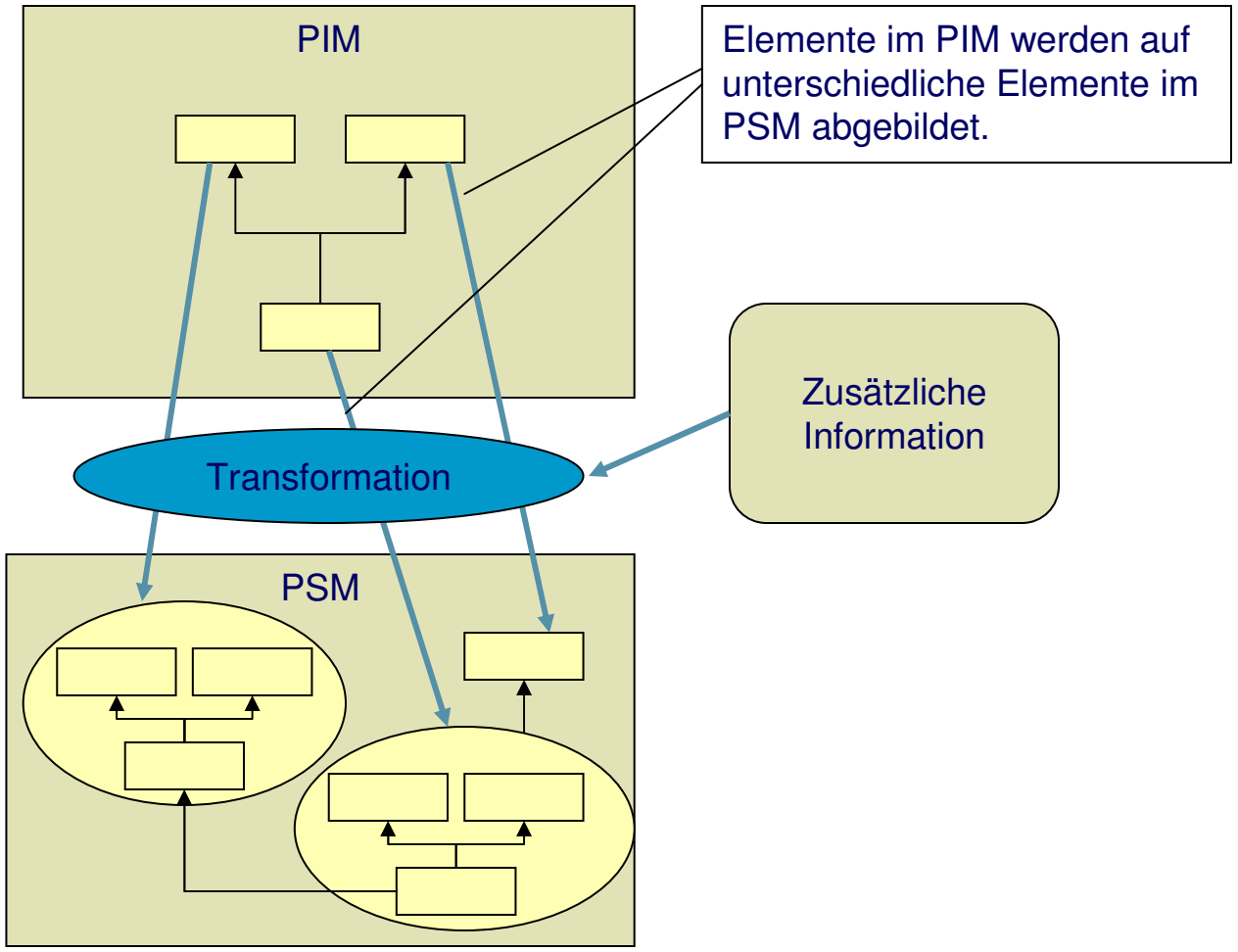
Dieses Modell ist (u.a.) unabhängig von der verwendeten Komponenten-Technologie.

Beispiel - PSM



Dieses Modell ist spezifisch in Hinblick auf die Verwendung von Enterprise Java Beans (aber unabhängig vom App-Server).

Transformation PIM zum PSM

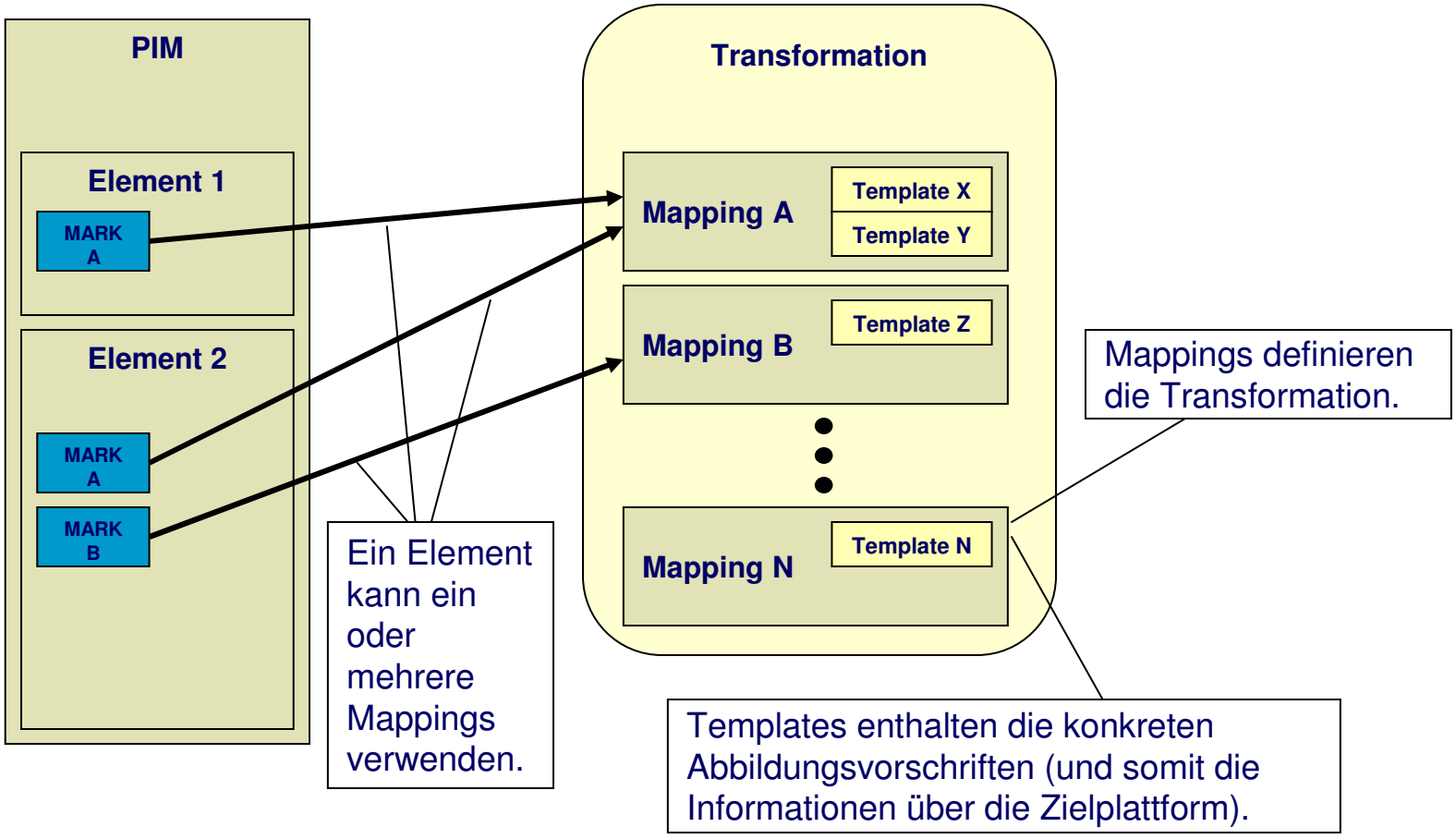


Transformation durch Annotation

- Eine *Transformation* wird mit *Marks (Annotationen)* gesteuert.
  - Ein oder mehrere *Marks* werden einem Element des PIM hinzugefügt.
  - Marks bestimmen, welche Elemente wie transformiert verwenden.
  - Marks sind nicht Teil des PIM (da Plattform-spezifisch).
- Eine *Transformation* besteht aus einem oder mehreren *Mappings*
- Mappings verwenden *Templates*:
  - Enthalten die konkreten Abbildungsvorschriften.
  - Repräsentieren z.B. Design Patterns.
  - Enthalten Informationen über die Zielplattform.
  - Templates werden von Architekten erstellt.

iteratec

Der Zusammenhang von Mappings, Marks und Templates



MDA – Lessons Learned

iteratec

MDA-Werkzeuge: Was leisten sie heute?

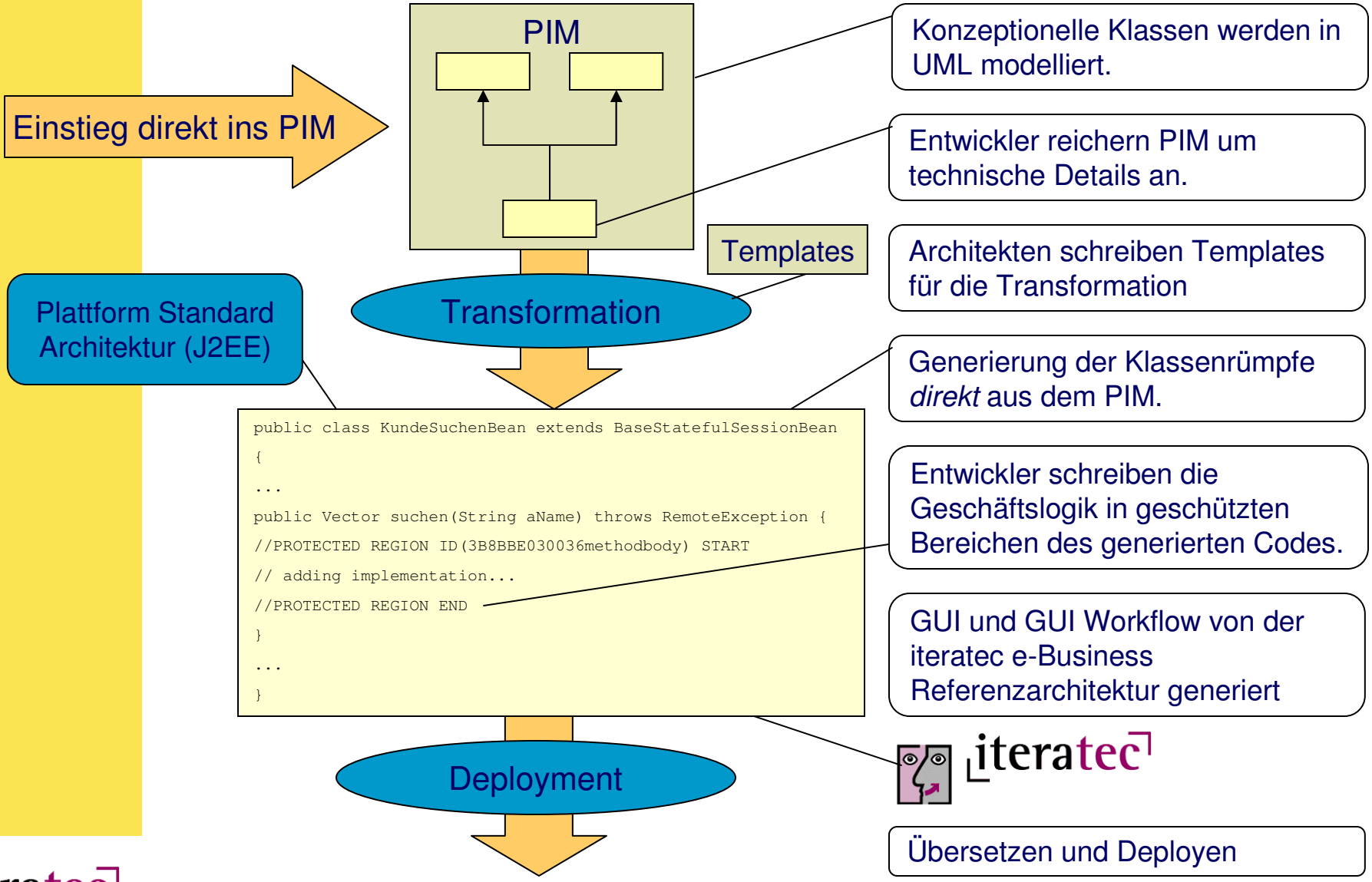
- Viele Werkzeuge behaupten, MDA umzusetzen.
- Angebliche MDA-Fähigkeiten entpuppen sich meist bei genauer Betrachtung als Seifenblasen.
- De facto gibt es kein Werkzeug, dass die komplette *Theorie* der MDA umsetzt.
- Effektive Tool-Unterstützung erfolgt lediglich in Teilbereichen.
- Werkzeuge sind insgesamt noch nicht ausgereift.
- Schwerpunkt liegt auf der Generierung.

iteratec

MDA-Werkzeuge: Was leisten sie nicht?

- Sie ersetzen nicht die Notwendigkeit zu programmieren.
  - Bislang keine Realisierung der Programmierung im Modell
- Sie entbinden den Entwickler nicht von der Kenntnis der Architektur.
  - Welche Architektur-Komponenten existieren?
  - Welche Modell-Komponenten kann ich auf welche Architektur-Komponenten abbilden?
- Sie ersetzen keinen Software-Architekten im Projekt!
- Auf Ebene der Generierung ist eine Unabhängigkeit nicht gegeben, da es keinen Standard für Generator-Sprachen gibt.
- Werkzeuge verwenden Dialekte von XMI.

Was ist heute Realität?



Das Projekt LDZ: Leistungsdaten Zug

- Auftraggeber: ÖBB, GB Netz, Trassenmanagement
- Ziel: Schaffung der Grundlage zur leistungsbezogenen Abrechnung der Infrastrukturbenutzung mit den Eisenbahnverkehrsunternehmen
- Entwicklungsumgebung:
  - ArcStyler 2.6, RationalRose 2001, ClearCase, JBuilder, Ant, Bea AS 4.5, Tomcat, Oracle 8.1.7, Toad
- Prüfung der bestehenden Funktionalität gegen nicht-funktionale Anforderungen (insbesondere Performance).
- Redesign eines Teil des Modells aufgrund geänderter fachlicher Anforderungen.

iteratec

Das Projekt LDZ: Was hat‘s gebracht?

- Die Einarbeitung in eine derart komplexe Umgebung braucht Zeit, lohnt sich aber.
- Der Schwerpunkt der Entwicklung verlagert sich auf die Modellierung.
- Die Implementierung beschränkt sich nahezu auf die fachlichen Algorithmen.
- Die Aufgaben eines Architekten verändern sich:
  - Keine Entwicklung von Framework-Komponenten.
  - Fundierte Kenntnisse der Abblidungsvorschriften des Generators.
  - Reviews auf Modelle.
- Entwicklungsbudget gegenüber den Angeboten um insgesamt 35% unterschritten (ÖBB).

MDA – Vor- und Nachteile

iteratec

Nachteile I

- Keine standardisierte „Mapping Language“.
  - Transformation der Modelle daher nicht portabel.
  - Weitere Abhängigkeiten von Herstellern.
- Wie wird QS auf die Modelle durchgeführt?
  - „Pseudo“-Qualität durch Modelle?
- Generierte Artefakte schlecht von Menschen zu verstehen.
  - Manuelle Nachbearbeitung und Tuning schwierig.
- UML 2.0 ist noch nicht verabschiedet.
- Weiterentwicklung der Templates und Generatoren führt zu Folgeaufwand.
- Verträglichkeit mit Softwareentwicklungsprozessen unklar.

Nachteile II

- Entfremdung der Entwickler vom Code:
  - Erzeugter Code basiert auf Annahmen der Architekten bei der Entwicklung der Mappings (dem Entwickler unbekannt).
  - Code wird von den Entwicklern in vielen Fällen nicht mehr verstanden (Vertrauensverlust).
- Das Kreative am Programmieren geht verloren.
  - Ein Modellierer har weniger technischen Spielraum.
  - Kreativität verschiebt sich von der Programmierung in das Modell.
- Architektur nur implizit in der Transformation (Generierung) vorhanden.
  - Wird noch über Architektur nachgedacht?
  - Bei Standardarchitekturen akzeptabel.
- Gefahr, dass nichtfunktionale Anforderungen vernachlässigt werden.

Vorteile I

- Wenn Templates vorhanden sind:
  - Erhöhte Produktivität.
  - Erste Ergebnisse werden schnell erreicht.
  - Integration mehrerer Plattformen erleichtert
- Koexistenz verschiedener Plattformen wird explizit unterstützt.
- System ist resistent gegen Änderungen der Technologie.
  - PIM kann weiterhin unverändert benutzt werden, wenn die Plattform geändert wird.
  - Umstellung auf eine andere Plattform wird einfacher.
  - Man bleibt technologisch flexibel.
  - Investitionen in Fachmodelle leben länger (Investitionsschutz).
- Validierung und Ausführung auf Modellebene (Vision).

iteratec

Vorteile II

- Die Softwarearchitektur wird durch die automatische Transformation durchgängig realisiert.
  - Theoretisch automatische Einhaltung von Programmierstandards.
  - Theoretisch Konsistenz von Modell und Implementierung.
- Viele Probleme werden nur einmal gelöst.
  - Lösung im Modell oder
  - Lösung in den Templates.
- Vermeidung von Fehlern auf Code-Ebene (z.B. „Cut and Paste“).

iteratec

Konzeptionelle Vorarbeiten lösen viele der Probleme

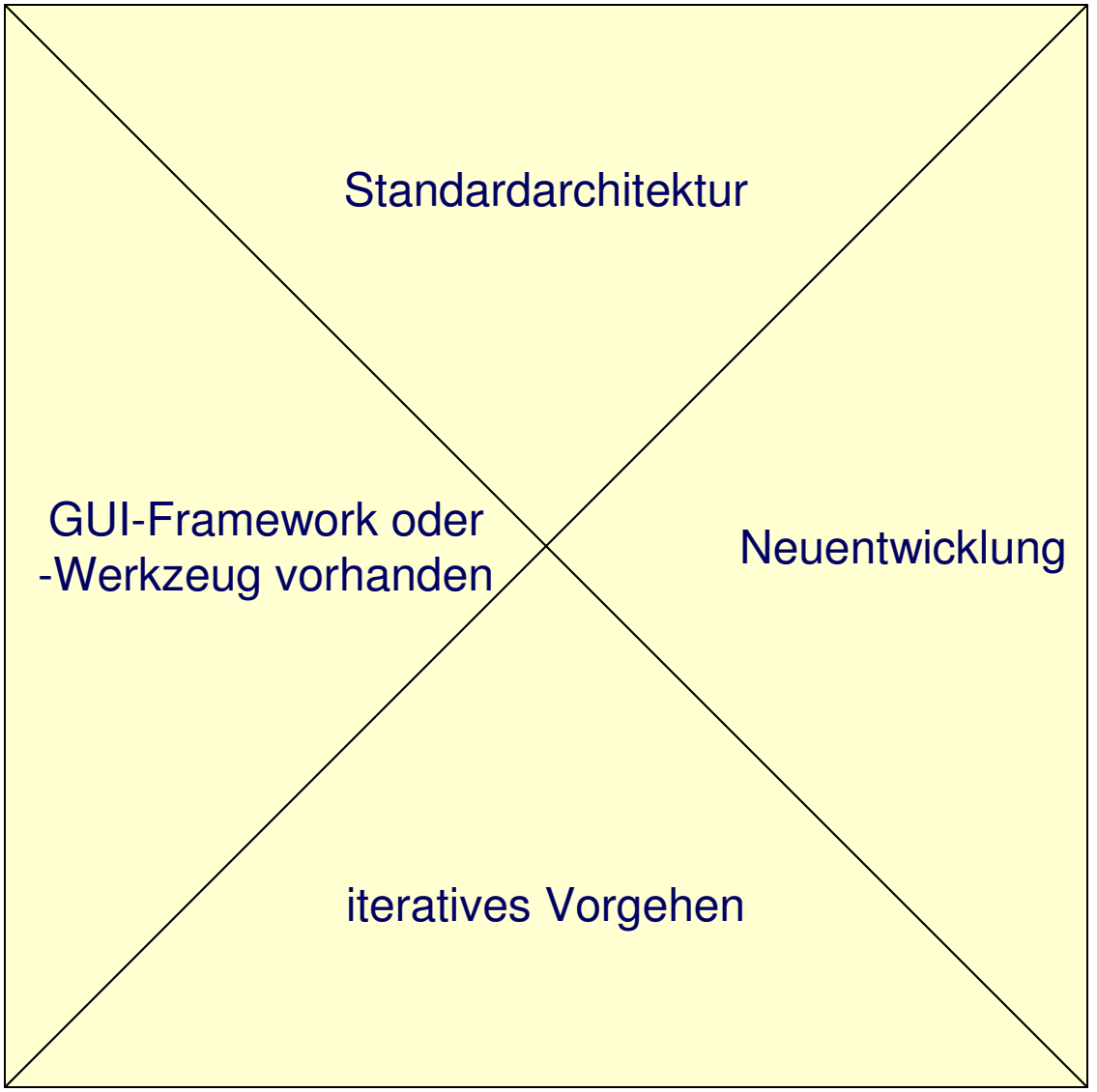
- Entfremdung der Entwickler
  - Templates in den normalen Entwicklungsprozess einbeziehen.
- Softwarearchitektur
  - Spezialist f. Generator notwendig.
  - Aktivität explizit in das Vorgehen aufnehmen (Teilprojekt Architektur).
- Softwareprozess
  - Abbildung von MDA auf Prozess (o. umgekehrt) vornehmen.
- Anforderungen
  - Vorgehen bzgl. nichtfunktionaler Anforderungen erstellen.
- Validierung von Modellen
  - Konzept zur QS von Modellen erstellen, in Vorgehen aufnehmen.
- UML 2.0
  - Welche Teilmenge der UML reicht für die spezifische Aufgabe aus?

iteratec

MDA - Fazit

iteratec

Wann kann man MDA einsetzen?



Was erhöht die Wirksamkeit der MDA?

- Es gibt viele Projekte mit gleicher technischer Basis oder ein Projekt mit vielen Releases.
- Es steht eine Änderung der technischen Plattform während der Projektlaufzeit oder kurz danach an.
- Das Projekt soll kurzfristig auf Erweiterungen der Fachlichkeit reagieren können.
- Der Anwender kann mit Modellen umgehen und bringt ein Grundverständnis der UML mit.
- Das Projekt besteht aus vielen Teilteams, die teilweise auf unterschiedlichen Ebenen arbeiten.

iteratec

MDA und CASE - Abgrenzung

- CASE:
  - Metamodell und Transformation sind kaum änderbar.
  - Einige CASE-Tools enthalten feste Laufzeitsysteme, die durch den Entwickler nicht modifiziert werden können.
  - Modelle sind nicht austauschbar, daher starke Bindung an Hersteller.
  - Tools erzwingen Wasserfall-Vorgehen.
  - Werkzeuge sind nicht offen und nicht erweiterbar.
  - Zielarchitektur auf HOST ausgerichtet, schlecht änderbar.
- MDA
  - Baut auf etablierten Standards (UML) und OO auf.
  - Entwickler sind mit Modellen vertraut.
  - Nutzt existierende Middleware- und Komponententechnologien.
  - Die Zielsysteme sind austauschbar.
  - Schwerpunkt der Werkzeuge liegt auf der Generierung.

iteratec

Testen

- Testen ist auch im MDA Kontext notwendig!
- Fokus des Testens verschiebt sich vom Code auf die *Modelle* und die *Transformationen*.
  - Wenn Modell und Generator korrekt sind, muss der generierte Code nicht mehr getestet werden.
- Testen des Modells:
  - Fachliche Korrektheit und Vollständigkeit
  - Technische Korrektheit
  - Technische Qualität
- Testen der Transformation:
  - Muss automatische testbar sein, um Änderungen zu erlauben.
  - Einfache Verifikationsmodelle notwendig.
  - *Vor* Start der eigentlichen Entwicklung getestet und erprobt.

iteratec

MDA – Fazit

- Einsatz der MDA *in geeigneten Projekten* sinnvoll.
- Konzeptionelle Vorarbeiten sind notwendig.
- Bei einer effizienten Tool-Unterstützung hilft MDA Kosten sparen, auch dann, wenn man die Plattformunabhängigkeit nicht nutzt.
- Aufwände für die Erstellung von Frameworks entfallen.
  - Nicht jedoch die Erstellung von Basisfunktionalitäten
  - Nicht jedoch die Entscheidung für eine Zielarchitektur
- MDA wirkt auch, wenn man nur Teile anwendet, die aber konsequent.

iteratec

Vielen Dank für Ihre Aufmerksamkeit.

iteratec