

# MDSD@BIT

23. / 24. März 2010

Stefano Juri / Frank Buchli



## Agenda

- Vorstellung BIT
- Ausgangslage
- Probleme
- Lösung
- Erkenntnisse
- Q&A



# Bundesamt für Informatik und Telekommunikation

## *Aufgabe*

- erbringt **Informatik- und Telekommunikationsleistungen** zur Unterstützung der Geschäftsprozesse in der Bundesverwaltung
- liefert sowohl kundenspezifische Lösungen wie Querschnittlösungen (SAP, Netzwerk, usw.)

## *Eckdaten*

- 1200 Mitarbeiter, davon etwa 350 in der Entwicklung
- 1600 Server
- 32'000 User
- 6,5 Mio Mails/Monat
- 1000 SW-Anwendungen
- Entwicklung hauptsächlich in Java und DotNet. Legacy Anwendungen in Cobol, Natural/Adabas, Oracle Forms



# Ausgangslage

- Das BIT hat seit vielen Jahren die Notwendigkeit für eine **standardisierte Anwendungsarchitektur** erkannt.
- Die Entwicklung der ersten Version hat im Jahr 2000 angefangen.
- Heute haben wir eine ausgereifte Architektur für Java Anwendungen.
- Die Entwicklung für DotNet hat erst im Jahr 2009 gestartet.

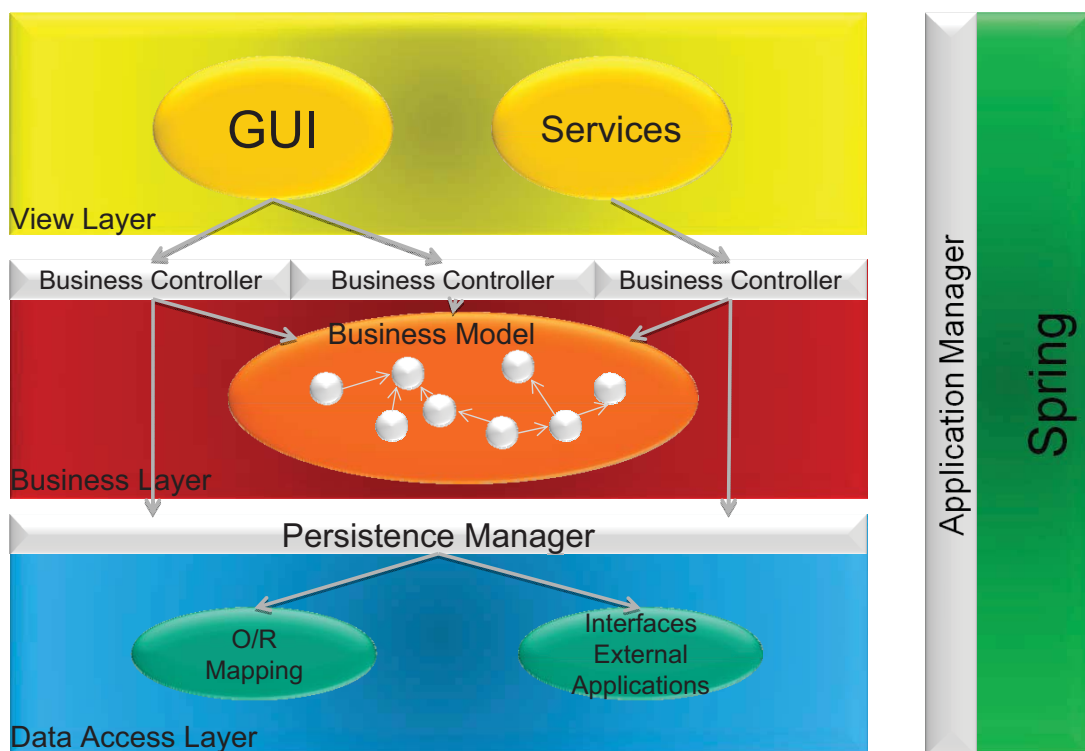


# Standard-Architektur Prinzipien

- Stark Domain Driven
- Das BIT hat ein Framework zur Implementierung der Standardarchitektur entwickelt.
- Er basiert auf marktüblichen Tools: Spring, JSF oder Struts, JPA/Hibernate, EJB, SOAP, usw.
- Motto: „Wann immer möglich nicht selber entwickeln“  
→ Wenn eine Funktion des Framework auf dem Markt verfügbar ist, wird sie im Framework ersetzt.



# Standardarchitektur





# Standardarchitektur: Stärken und Schwächen



- Ähnliche Architektur über alle Projekte
- Zeitgewinn bei der Festlegung der Architektur
- Gute Isolation der Schichten
- Leichtere Austauschbarkeit der Basis-Frameworks



- Immer noch viel handwerkliche Arbeit
- Qualität ist stark von der Disziplin des Entwicklers abhängig
- Produktivität immer noch nicht sehr hoch

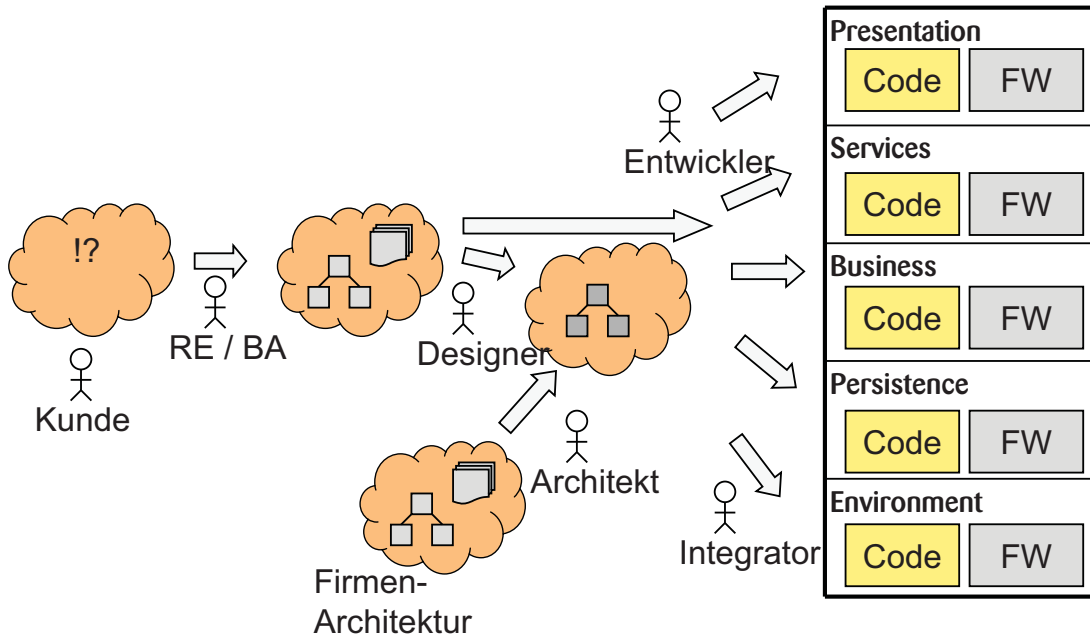


# MDSD: Zielsetzung

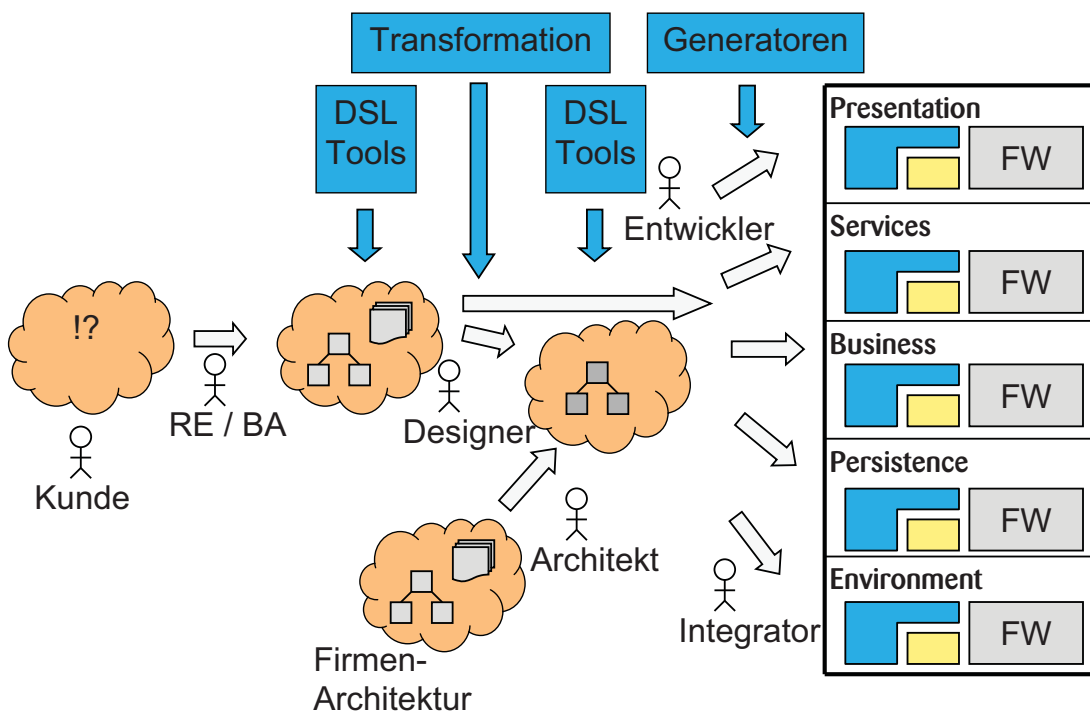
- **Produktivität erhöhen & bessere Qualität**
  - Weniger handcodierte Programmzeilen
  - Bessere Überlegung in der Designphase
  - Code näher an der Design-Dokumentation
  - Automatisierung von repetitiven und/oder fehleranfälligen Programmstellen
  - Unterstützung des Testing



# Software Engineering: Ablauf

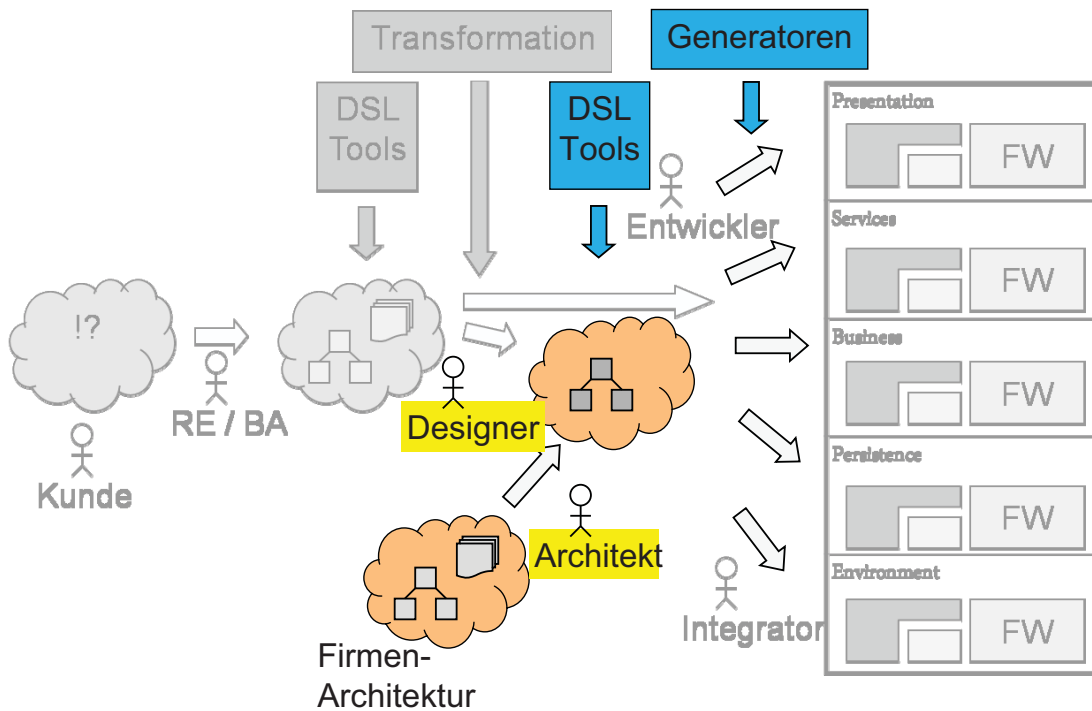


# Software Engineering: Ablauf mit MDSD





# MDSD: Ansatzpunkte im BIT



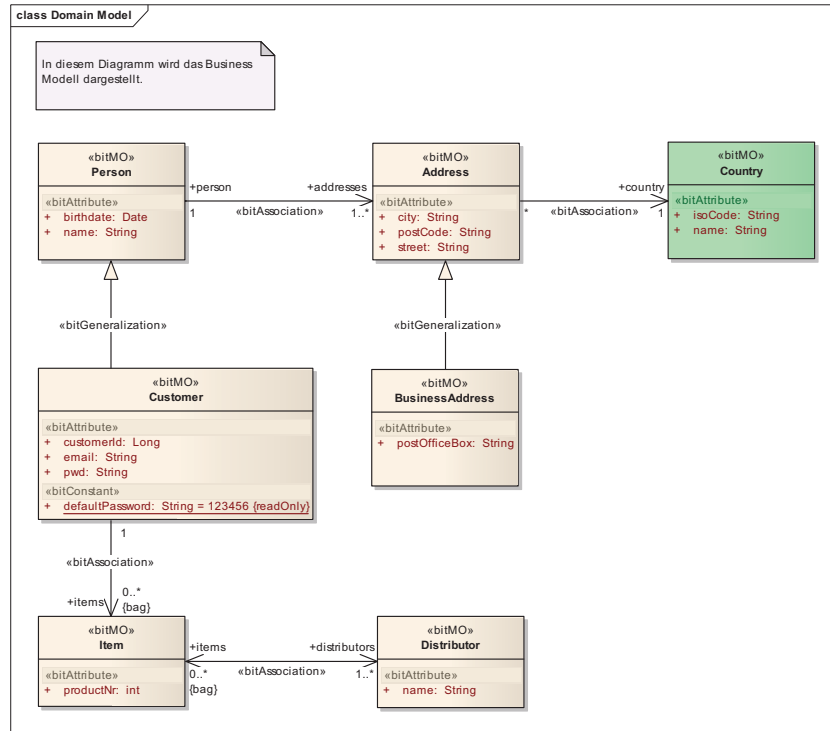
## MDSD im BIT: Wie machen wir es?

- bitDSL
  - DSL als UML2 Profil
- SparxSystems  
Enterprise Architect
- Generatoren
  - oAW 4.3.1

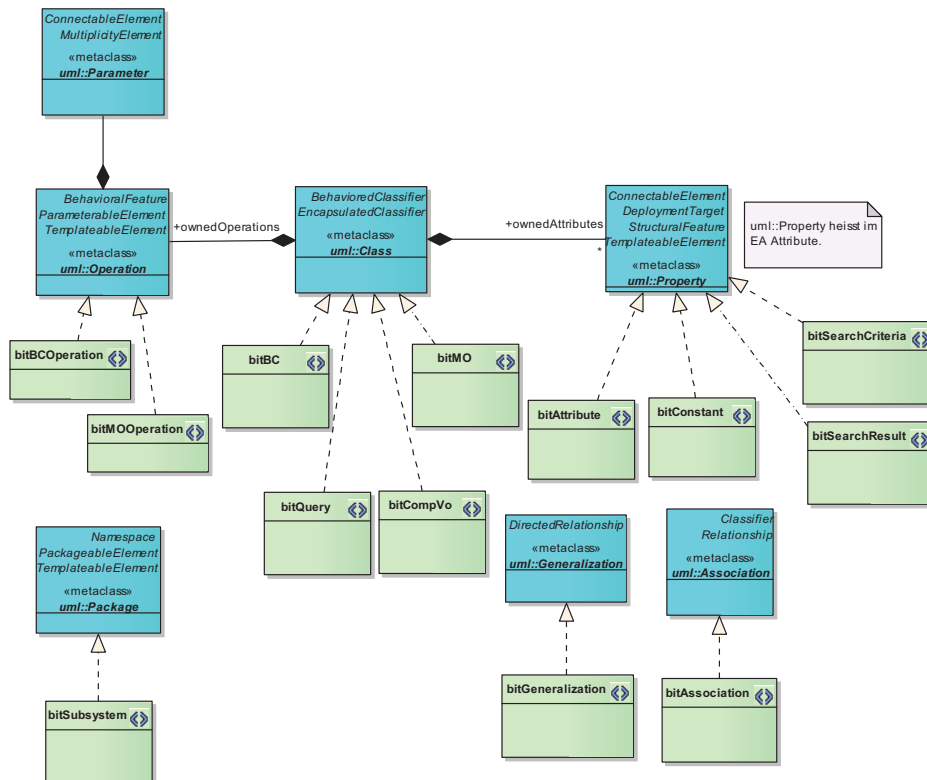




# Beispiel Model



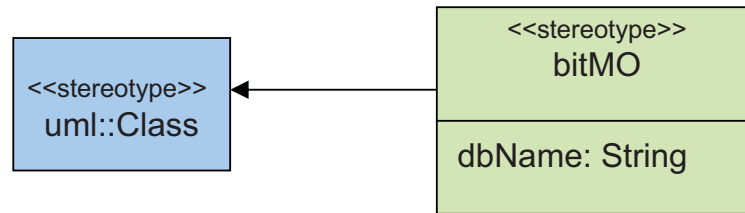
# bitDSL



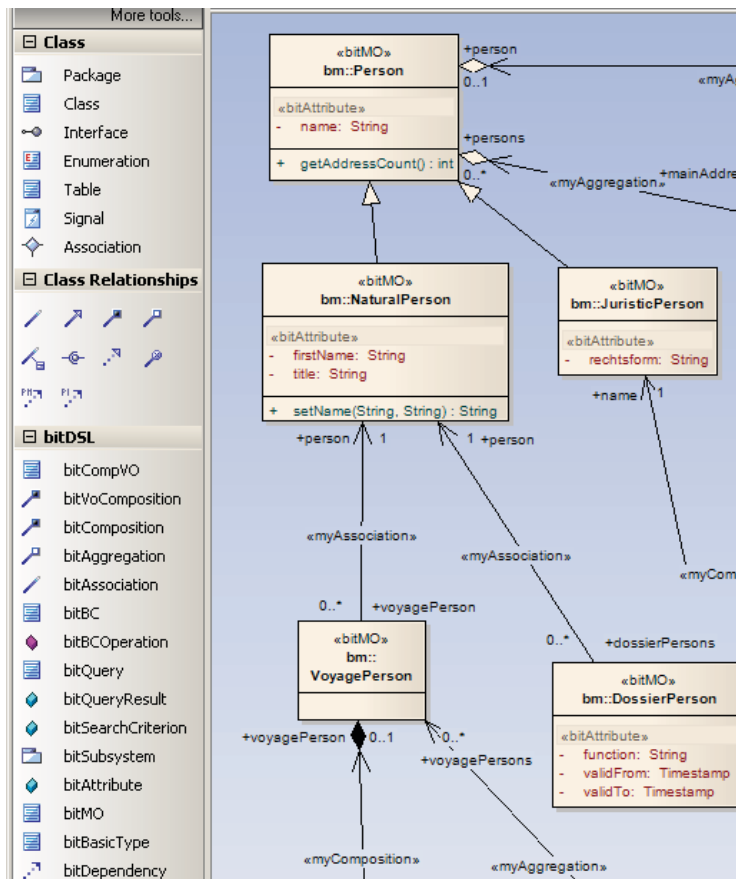
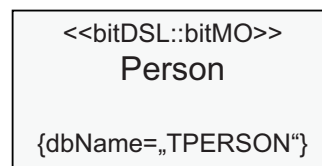


# UML Profile als DSL

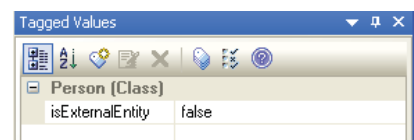
- Definition



- Verwendung



## Enterprise Architect





## Was generieren wir?

- Repetitiver Code
- Fehleranfälliger Code
- „Pareto-Prinzip“
- Granularität: Files
  
- Beispiele
  - Business POJOs samt Constraints
  - O/R-Mapping
  - Services
  - Projektstruktur
  - Tests
  - Konfigurationen



## Was generieren wir (noch) nicht?

- Geplant
  - Support Historisierung
  - Status-Klassen
  - View Scaffolding
  
- Reserviert für manuelle Implementierung
  - User Interface (Masken)
  - Anbindung an Fremdsysteme
  - Algorithmen



## Erkenntnisse



- Gute Akzeptanz bei den Entwicklern
- Bessere Designqualität und bessere Diskussionen darüber
- Qualität des generierten Codes sehr gut
- Projektstrukturierung



## Erkenntnisse



- Höhere Abstraktionsfähigkeit vom Designer verlangt
- Unterliegende Architektur muss vom Designer verstanden werden.
- Qualität der Standardarchitektur ist sehr wichtig, da Änderungen schwieriger werden.



# Erkenntnisse



- UML ist komplex -> Kapselung (Bsp: Model-Transformation)
- Überzeugungs- und Begleitungsaufwand ist unumgänglich



# Fragen und Antworten

