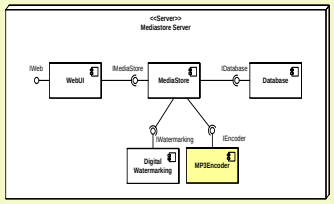
Experiences with Model Driven Software Development
Creating the Palladio Tool Chain

Eclipse Application Developer Day
7. July, 2009

Dr.-Ing. Steffen Becker
sbecker@fzi.de
Abteilungsleiter Software Engineering
FZI Forschungszentrum Informatik

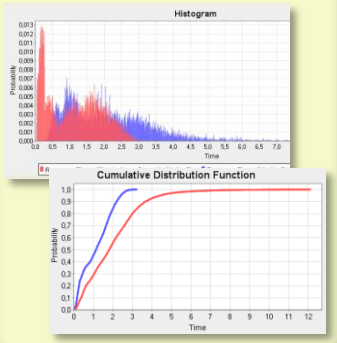
Problemstellung

Definition einer Modellierungssprache



- Modellierung von komponentenbasierten Software-Architekturen
- Annotation mit Qualitätsattributen
- Statik, Dynamik und Allokation abbildbar
- Semantisch reicher als UML

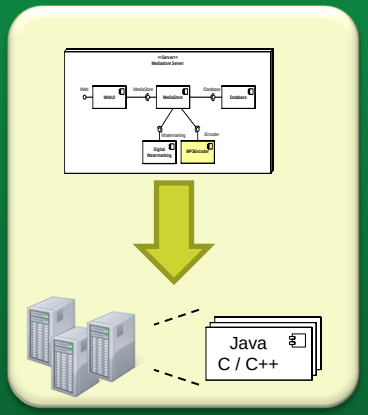
Modellanalysen



- Performance-Vorhersagen
- Zuverlässigkeitsvorhersagen
- Kostenschätzungen
- Wartbarkeitsanalysen

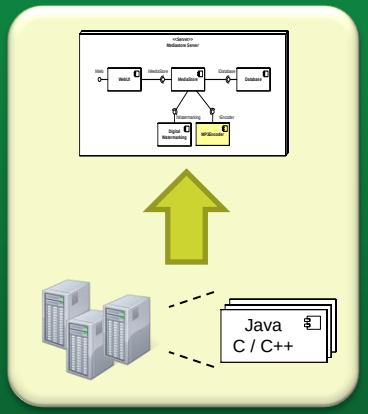
Problemstellung II

Codegenerierung



- Code-Skelette aus Modell zur Validierung
- Prototypen aus Modell

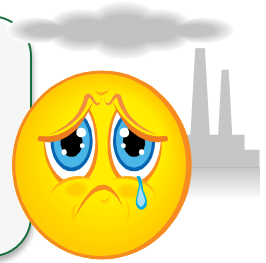
Modellgewinnung



- Erstellung von Modellen aus Altanwendungen
- Gewinnung von Annotationen aus laufenden Systemen

Erster Versuch
Klassischer Ansatz in .NET

1.
- OO-Entwurf des Modells
- Realisierung in C#



2.
- Editor implementieren
- GUI Entwurf

3.
- COM Komponenten Recovery
- Parser und Analyser Entwurf für C# Code

Hohe Änderungsrate am OO-Modell

- Implementierung schnell veraltet
- Serialisierung fehleranfällig trotz Mustereinsatz
- Tests oft nicht schnell genug anpassbar

GUI-Editor Realisierung

- Extrem fehleranfällig trotz MVC oder PAC Muster und ausgiebigen Tests
- Schlechtes Rahmenwerk

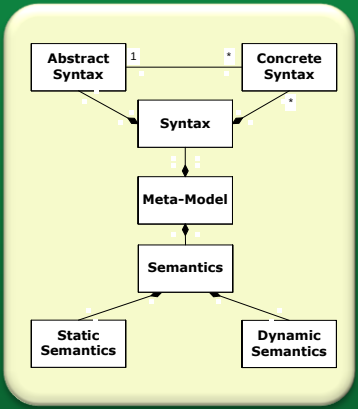
Zweiter Versuch
Eclipse und MDSD

1.
   - EMF Meta-Modell erzeugen
   - Realisierung durch EMF

2.
   - Editor mittels GMF mappen
   - Editor generieren und adaptieren

3.
   - Transformationen
   - Realisiert mittels oAW

4.
   - Reverse Engineering mittels JDT
   - Modellaufbau durch Visitor-Muster



Bestandteile eines Metamodells

- Abstrakte Syntax
- Mind. eine konkrete Syntax (s. Editoren)
- Statische Semantik
- Dynamische Semantik

Umsetzung

- Abstrakte Syntax mittels RSA UML2 Modell
- Ecore: Transformation mittels EMF Importer
- Statische Semantik mit OCL und EMF Extension
- Dynamik mittels In-Model Dokumentation

Erfahrungen
EMF Metamodellierung

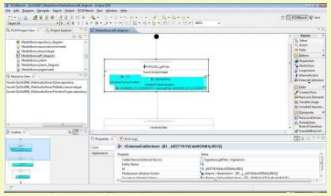
Agilität	Sehr hoch
Komplexität	Über 120 Metaklassen in 12 Paketen
Flexibilität	Erweiterbarer Modellcode
Qualität	Modell und Code gut versteh- und wartbar

Schritt 2
Generierung von GMF Editoren

GMF: Werkzeug für graphische Syntax

- Domain Model
- Tooling Model
- Graph Model
- Mapping Model

Umsetzung



- 4 GMF Editoren erzeugt
- Editor für Verhaltensmodell höchste Komplexität (~100 Modellelemente)
- Initiales Mapping und Feedback schnell erreichbar

Palladio Component Model

EDITOR DEMONSTRATION

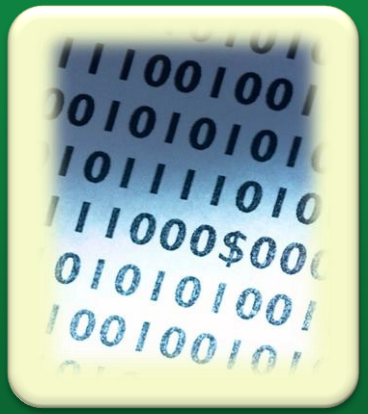
Erfahrungen GMF Editoren

Agilität	Prototypen schnell erzeugbar Nicht alle gewünschten Mappings modellierbar
Komplexität	Generat komplex, Mappingmodelle komplexer für realistische Editoren
Flexibilität	80:20 Regel: Anpassungen sind aufwändig aber möglich
Qualität	Editor qualitativ hochwertig

Vergleich Kosten/Nutzen der Editoren

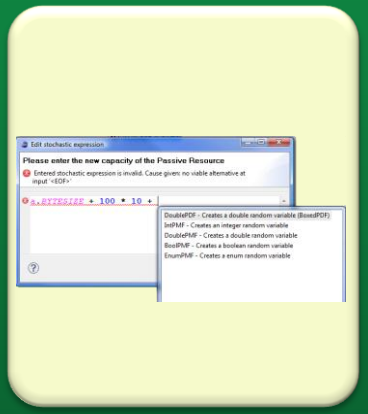
Kriterium	1. Versuch (manuell)	2. Versuch (MDSD)
Frameworks	C#, .NET, Netron („buggy“)	Java, Eclipse, EMF, GMF, GEF
Undo / Redo	Manuell	Generiert
Event-Handling	Manuell	Generiert
Entwicklungsaufwand	Hoch	Mittel
Aufwand nach Modelländerungen	Hoch	Mittel
API	Nach Notwendigkeit	Umfassend
Fehlerwahrscheinlichkeit	Hoch	Gering (Systematische Fehler)
Lernaufwand	Hoch	Mittel
Zeitbedarf	~ 4 Personen-Jahre	½ Personen-Jahre

2. Schritt
Textuelle Syntax



Anforderung

- Sprache für QoS Annotationen (Stochastic Expressions)
- Beispiel: `a.BYTESIZE * 100 + 10`
- Sprache metamodelliert
- Lexer und Parser mittels ANTLR



Umsetzung

- Manuelle Umsetzung
- Nutzung der Eclipse RCP-API
- MDSD-basierte Lösungen zu der Zeit (2007) untauglich (xText, TCS, ..)
- Features: Syntax Highlighting, Code Completion, etc.

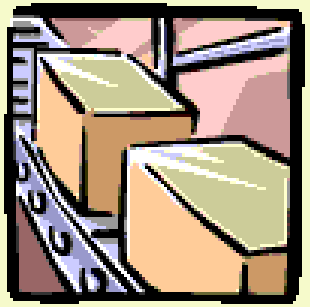
Palladio Component Model

STOCHASTIC EXPRESSIONS DEMONSTRATION

openArchitectureWare (M2T-Engine)

- Bietet Modell-Konzistenzprüfungen
- Template-basiertes Model-2-Text
- Workflow-Engine

Umsetzung



- Transformation Modell → Simulationscode, Modell → EJBs, Modell → Prototypen
- Modulare Transformation dank Aspektkonzept

Palladio Component Model

MODELLTRANSFORMATIONEN DEMONSTRATION

Erfahrungen Transformationen

Agilität	Hoch, Templates leicht änderbar
Komplexität	Mittel, > 30 Template Dateien
Flexibilität	Notfalls oAW erweiterbar, da Quelloffen
Qualität	Templates nicht ganz leicht wartbar, Fehler jedoch leicht auffindbar

Ausblick Transformationen

QVT (M2M-Engine)

- 2 Sprachen: QVT-Relational, QVT-Operational Mapping
- Werkzeuge werden gerade nutzbar
- Erwartung: Höhere Produktivität durch spezialisierte Transformationssprachen

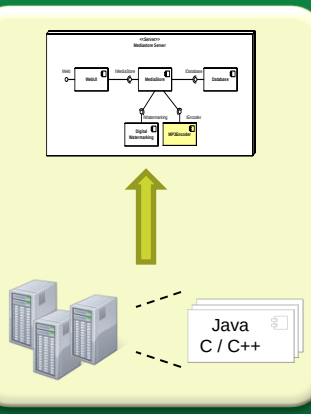
Einsatzgebiete

- Transformationsmodularisierung
- Einbinden von Plattformspezifika
- Modellmanipulationen (In-Place-Transformationen)
- Aktuell: Sammlung erster Erfahrungen mit Werkzeugreife und Wartbarkeit des Codes

4. Schritt
Modellgewinnung

Java-2-PCM

- Ausnutzen des G-AST von JDT
- Zugriff auf die Java-Code Struktur
- Einfache Ansteuerung der JDT-API



Einsatzgebiete

- Generierung der PCM-EMF-Instanz
- Erstellung des Modells dank EMF-generierter API
- Abstraktionsberechnung in Java
- Gute Unterstützung, „EMF-Builder“ vermisst

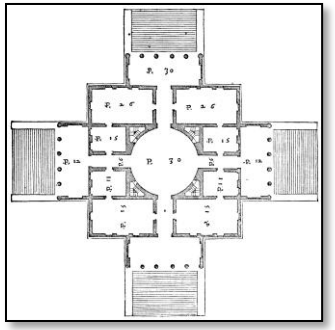
“Gefühlte” Qualität

Agilität	Der Einsatz von MDSD hat Palladio erst ermöglicht (Forschung als “Moving Target”)
Produktivität	Hohe Produktivität durch einzelne erzielbar
Reife	Stabilität variiert (z.B. EMF: gut, GMF: Potential)
Wartbarkeit	Unklar, einige Probleme (z.B. Metamodellevolution) identifiziert
Qualität	Für Forschungsprototyp hoch (z.B. durch integrierte Dokumentation & Tests)

Fazit

- MDSD bereits heute Praxistauglich
- Agile und Qualitative Entwicklung möglich
- Verbesserungspotential in Werkzeugen erkannt
- Entwicklung von DSLs (Graphisch & Textuell)
- Weitere Fallstudien in der Praxis nötig
- Ausbildung (z.B. am KIT)

Kontakt



Palladio Component Model

http://www.palladio-approach.net

Dr.-Ing. Steffen Becker
Abteilungsleiter Software Engineering
Ansprechpartner Palladio Component Model

FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
D-76131 Karlsruhe
Tel.: +49-721-9654-612
Fax: +49-721-9654-613
http://www.fzi.de/se