Workshop Hot Spots der Software-Entwicklung

9. Oktober 2006

Technische Universität München
Institut für Informatik
Software & Systems Engineering
Prof. Dr. Dr. h.c. Manfred Broy
&
ViSEK / Zentrum für Softwarekonzepte München

Report: VSEK/061/D
Version: 1.0
Klassifikation: extern

München, 22. Dezember 2006

Martin Feilkas
Elmar Jürgens
Tilman Seifert

Inhaltsverzeichnis

1 Einleitung

Am Montag, den 9. Oktober 2006, widmete sich ein weiterer Hot Spots Workshop dem Thema „Domänen-spezifische Sprachen und generative Entwicklung“. Dieser wurde in Kooperation der Projekte ViSEK und dem Zentrum für Softwarekonzepte am Lehrstuhl für Software & Systems Engineering der Technischen Universität München organisiert und veranstaltet.
Die Reihe „Hot Spots der Software-Entwicklung - HSE“ wurde im Jahre 2002 im Rahmen des ViSEK-Projekts (Virtuelles Software Engineering Kompetenzzentrum) gegründet. Ziel der Veranstaltungen ist es, eine Plattform zum Erfahrungsaustausch zu aktuellen Themen des Software-Engineering zu schaffen. Etwa 30 Teilnehmerinnen und Teilnehmer aus Forschung und Industrie diskutieren dabei über aktuelle Themen, Herausforderungen und Lösungswege.

Domänen-spezifische Sprachen halten derzeit unter vielen verschiedenen Namen wie Language-Oriented Programming, Generative Programmierung, Software Factories oder einfach Domänen-spezifische Sprachen (DSL) als neue Art der Software-Entwicklung Einzug in die Software-Branche. Die Technologie, die sich hinter diesen Bezeichnungen verbirgt, hat sich zum Ziel gesetzt, heutige Allzweck-Programmiersprachen und deren Modellierungsmethoden hinter sich zu lassen, um in stärkerer Fokussierung auf Anwendungsdomänen einen neuen Weg der Software-Entwicklung zu etablieren. Domänen-spezifische Sprachen treten an, die Software-Entwicklung effizienter und kostengünstiger zu gestalten und letztendlich Systeme mit weniger Fehlern zu produzieren. Mittlerweile drängen immer mehr Werkzeuge sowohl aus der Open-Source-Community als auch aus dem kommerziellen Umfeld auf den Markt, um die Erstellung von Domänen-spezifischen Sprachen zu erleichtern.
DSL-Technologien werden vielfach als Grundlagentechnologie zur Etablierung von Software-Produktlinien verstanden. Gerade Unternehmen, die stark fokussiert Software für eine bestimmte Domäne entwickeln, können sich durch die Einführung derartiger Techniken Skaleneffekte erhoffen, da eine DSL aufgrund der Anpassung an die Domäne eine wesentlich effizientere Art der Beschreibung zulässt, als eine konventionelle Programmiersprache. Dies beschleunigt die Entwicklung, führt dadurch zu geringeren Entwicklungskosten und trägt durch eine strukturierte Wiederverwendung zu besserer Software Qualität bei. Auch die Kommunikation mit Anwendern und Domänenexperten, die in der Software Entwicklung die Grundlage für jedes erfolgreiche Projekt bildet, kann durch den Einsatz von Sprachen, die sich auf Domänen-spezifisches Vokabular stützen, verbessert werden.
DSLs verlangen allerdings Vorleistungen. Die Entwicklung erfordert hoch ausgebildete Fachleute, die sowohl über das nötige Domänen-Wissen als auch über Kenntnisse in der Entwicklung von Programmier- und Modellierungssprachen verfügen.

Die neuen Perspektiven und Fragestellungen, die dieser neue Ansatz mit sich bringt, wurden im Rahmen des Workshops in einem Kreis von Experten aus Wirtschaft und Wissenschaft genau beleuchtet. Dabei wurden Berichte und Erfahrungen über Projekte vorgetragen, in denen generative Technologien oder DSLs zum Einsatz kamen, sowie unterschiedlichste Werkzeuge zur DSL-Entwicklung vorgestellt. Gerade auch die langfristigen Auswirkungen des Einsatzes von DSLs wurden in den Mittelpunkt gestellt und diskutiert, inwieweit sich deren Einsatz in Zukunft auf die Problematiken der Software Wartung auswirken wird.

Der Workshop wurde durch Mittel aus folgenden Projekten unterstützt:

- ViSEK- das virtuelle Kompentenzzentrum für Software-Engineering - wird von acht

renommierten Forschungseinrichtungen geführt und bietet ein Kompetenznetzwerk für den Transfer von Wissen und Erfahrungen zwischen Praxis und Forschung der Software- und Systementwicklung.

- Das Zentrum für Softwarekonzepte (ZfS) am Lehrstuhl für Software & Systems Engineering der Technischen Universität München ist eine Kooperation mit Microsoft Deutschland, die sich zum Ziel gesetzt hat, den Standort Deutschland zu stärken, indem der Transfer von Konzepten und Technologiewissen zwischen Mittelstand und Forschung vorangetrieben wird. Diese Zusammenarbeit konzentriert sich auf den Austausch von konzeptionellem Wissen rund um die Entwicklung, Einführung und den Betrieb von flexiblen Softwaresystemen.

2 Teilnehmerliste

- Josef Adersberger, Qaware
- Thomas Benedek, BMW Car IT GmbH
- Sebastian Benz, BMW Car IT GmbH / TU München
- Prof. Dr. Dr. h.c. Manfred, Broy, Technische Universität München
- Christian Buckl, Technische Universität München
- Martin Feilkas, Technische Universität München
- Peter Fleischer, Flughafen München GmbH
- Jörg Flügge, Realtime Technology AG
- Martin Fritzsche, Technische Universität München
- Josef Fuchsbauer, Qaware
- Dr. Robert Gerstberger, Münchener Rückversicherungs-Gesellschaft
- Frank Hoppe, msg systems ag
- Benjamin Hummel, Technische Universität München
- Lars Jordan, Realtime Technology AG
- Elmar Jürgens, Technische Universität München
- Gunther Lenz, Siemens Corporate Research
- Carsten Lichy-Bittendorf, msg systems ag
- Dr. Karl-Rudolf Moll, Selbstständiger Berater
- Birgit Penzenstadler, Technische Universität München
- Dr. Markus Pizka, itestra GmbH / TU München
- Harald Ranner, Flughafen München GmbH
- Dr. Bernhard Schätz, Technische Universität München
- Ronny Schulz, Realtime Technology AG
- Tilman Seifert, Technische Universität München
- Rainer Singvogel, msg systems ag
- Bernhard Steckenbiller, Flughafen München GmbH
- Siegmar Szlavik, Realtime Technology AG
- Martin Tiede, BMW Car IT GmbH
- Markus Völter, Selbstständiger Berater

3 Programm

Block 1	
13:00	**Begrüßung am Lehrstuhl für Software & Systems Engineering**
	Prof. Dr. Dr. h.c. Manfred Broy, Technische Universität München
13:05	**Einführung zum Workshop**
	Martin Feilkas, Technische Universität München
13:15	**Wieviel MDA/MDSD braucht bzw. verträgt ein Projekt?**
	- Bericht über den Balanceakt aus einem laufenden Projekt.
	Carsten Lichy-Bittendorf, msg systems ag
14:00	*Kaffee-Pause*
Block 2	
14:15	**Leistungsfähige Generatoren sind mehrstufig und evolutionär.**
	Dr. Markus Pizka, itestra GmbH / TU München
	Werkzeug-gestützte Sprach-Evolution
	Elmar Jürgens, Technische Universität München
15:00	**Software Factories, von der Theorie zur Praxis...**
	Gunther Lenz, Siemens Corporate Research
15:45	*Kaffee-Pause*
Block 3	
16:00	**Erstellung von DSLs mit Eclipse und openArchitectureWare**
	Markus Völter, unabhängiger Berater
16:45	**RGen - Ruby Modelling and Generator Framework**
	Martin Thiede, BMW Car IT GmbH
Diskussion	
17:30	**Abschlussdiskussion**
18:00	**Empfang**

4 Vorträge und Diskussion

In diesem Abschnitt werden die Vortragsfolien und die wichtigsten Diskussionspunkte dargestellt. Der Diskussionsverlauf wird dabei nicht chonologisch wiedergegeben, sondern orientiert sich an inhaltlichen Schwerpunkten. Dabei werden zusammenhängende Diskussionspunkte, die in mehreren Vorträgen zu Diskussionen geführt haben, oftmals zentral an einer Stelle zusammengefasst.

4.1 Martin Feilkas: „Einführung zum Workshops“

Der Gastgeber

- TUM, Lehrstuhl IV, Prof. Dr. Dr. h.c. Manfred Broy
  - Software & Systems Engineering
  - ca. 70 wissenschaftliche Mitarbeiter
- Schwerpunkte der Arbeit
  - Software- und Systementwicklung: Grundlagen, Methoden, Prozesse, Modelle, Beschreibungstechniken und Werkzeuge

TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Lehrstuhl IV: Software & Systems Engineering
2

Veranstaltende Projekte

- VSEK - virtuelles Kompentenzzentrum für Software-Engineering

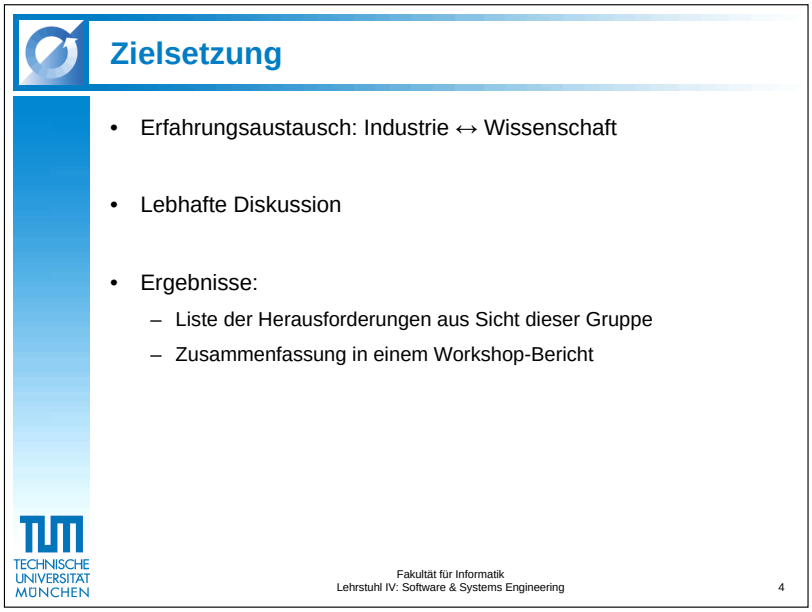
se software-engineering
www.software-kompetenz.de

- Zentrum für Softwarekonzepte

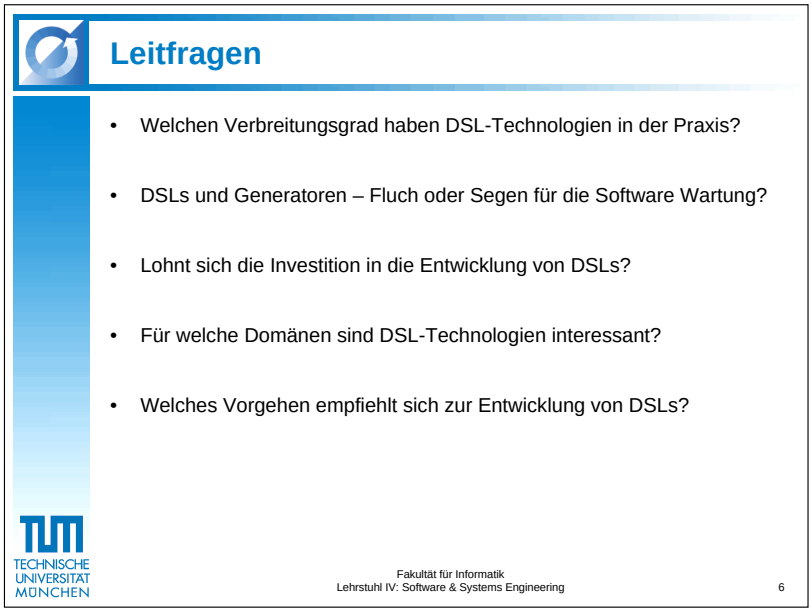
Zentrum für Softwarekonzepte
.NET-Technologien. Coaching. Know-how.
In Zusammenarbeit mit Microsoft Deutschland

TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Lehrstuhl IV: Software & Systems Engineering
3


Zielsetzung
• Erfahrungsaustausch: Industrie ↔ Wissenschaft
• Lebhafte Diskussion
• Ergebnisse:
– Liste der Herausforderungen aus Sicht dieser Gruppe
– Zusammenfassung in einem Workshop-Bericht
TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Informatik
Lehrstuhl IV: Software & Systems Engineering
4


Programm
Block 1
13:00 bis 13:05 Begrüßung durch Prof. Dr. Dr. h.c. Manfred Broy, Lehrstuhl für Software & Systems Engineering, TU München
13:05 bis 13:15 Martin Feilkas (TU München) Einführung zum Workshop
13:15 bis 14:00 Carsten Lichy-Bittendorf (MSG): Wieviel MDA/MDSD braucht bzw. verträgt ein Projekt? - Bericht über den Balanceakt aus einem laufenden Projekt.
Kaffeepause
Block 2
14:15 bis 15:00 Dr. Markus Pizka (itestra GmbH, TU München) Leistungsfähige Generatoren sind mehrstufig und evolutionär. Elmar Jürgens (TU München) Werkzeug-gestützte Sprach-Evolution
15:00 bis 15:45 Gunther Lenz (Siemens Corporate Research) Software Factories, von der Theorie zur Praxis...
Kaffeepause
Block 3
16:00 bis 16:45 Markus Völter (Unabhängiger Berater) Erstellung von DSLs mit Eclipse und openArchitectureWare
16:45 bis 17:30 Martin Thiede (BMW Car IT GmbH) RGen - Ruby Modelling and Generator Framework
Diskussion
17:30 bis 18:00 Abschlussdiskussion
Anschließend Empfang
TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Informatik
Lehrstuhl IV: Software & Systems Engineering
5


Leitfragen
• Welchen Verbreitungsgrad haben DSL-Technologien in der Praxis?
• DSLs und Generatoren – Fluch oder Segen für die Software Wartung?
• Lohnt sich die Investition in die Entwicklung von DSLs?
• Für welche Domänen sind DSL-Technologien interessant?
• Welches Vorgehen empfiehlt sich zur Entwicklung von DSLs?
TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Informatik
Lehrstuhl IV: Software & Systems Engineering
6

4.2 Carsten Lichy-Bittendorf: „Wieviel Modellgetriebenheit braucht bzw. verträgt ein Projekt?“

4.2.1 Abstract

In der Literatur werden sehr viele abstrakte und theoretische Ansätze zum „Modellgetriebenen Vorgehen“ unter vielen Akronymen beschrieben.
Beginnt man jedoch ein konkretes Projekt, stellen sich sehr schnell sehr viele Fragen:

- welche Modelle gibt es als reale Modelle? - welche sind nur virtuell?
- wer definiert den Inhalt der Modelle? - technisches Framework vs. fachlich getriebenen Vorgehen
- wie sieht die Domain Specific Language aus? Gibt es eine Enterprise DML?
- wie funktioniert ein iteratives, inkrementelles Vorgehen (u.U. unvollständige Modelle)?
- welche Modellierungssprachen werden wann eingesetzt?
- und nicht zuletzt wo läßt sich welcher Mehrwert erzeugen und welche Werkzeuge braucht man, um diesen zu nutzen?

Dieser Vortrag stellt die aktuellen Antworten vor, die ein Projekt der msg für sich gefunden hat. Diese werden in einen Rückblick reflektiert und es wird ein Ausblick auf die nächsten Schritte gegeben.

4.2.2 Folien

msg systems ag – Übersicht

.consulting .solutions .partnership

Wer sind wir ?

Als produktbasiertes Lösungs- und Service Haus agieren wir unter den führenden Top 25 der IT-Berater und Systemintegratoren in Deutschland.

Kerngeschäft:

*** Branchenspezifische Gesamtlösungen**
- **Beratung, Anwendungen, Systemintegration**
- **Versicherungen, Finanzdienstleistungen, Automotive, Gesundheitswesen**

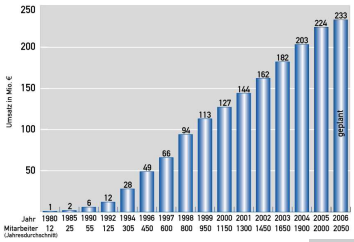
*** Branchenübergreifende Technologien & Services**

Gründung: 1980

Geschäftssitz: Ismaning / München

Vorstand: Hans Zehetmaier
Dr. Peter Brössler
Karl-Martin Klein
Volker Reichenbach

Mitarbeiter: > 2.000



Dipl.-Ing. Carsten Lichy-Bittendorf
Persönliche Vorstellung

.consulting .solutions .partnership

Verantwortete IT-Projekte :

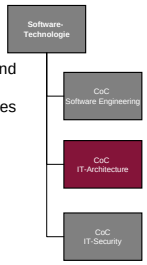
- Anwendungs-Entwicklung / Betreuung
  - Konzeption
  - Implementierung
  - Betrieb
- Entwicklungswerkzeuge
  - Diverse C und Java-IDEs
  - Rational Development Platform
- Architekturen
  - JEE basierende Technologie Prototypen in der Automobil-Industrie
  - JEE Großprojekte in den Bereichen:
    - Banken
    - Versicherungen
    - Öffentlicher Dienst
    - Automobil-Industrie

Carsten Lichy-Bittendorf
Dipl. Ing.
Center of Competence
IT-Architecture

msg systems ag
Geschäftstelle Hannover
Karl-Wiechert-Allee 20
30625 Hannover
Telefon 0511/35333-0
Telefax 0511/35333-113
Mobil 0172/5433736
lichyc@msg.de

msg – Crossbereich Software Technologie
- Geschäftsbereichsübergreifend
- Software Technologie Experten, Trainer und Coaches
- Owner von msg.PROFI (**Pro**zessrahmen **F**ür **I**ndividualentwicklung)

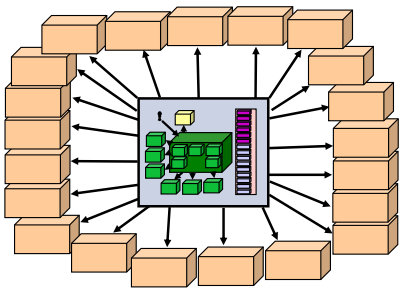
CoC IT-Architecture
- Integrationsarchitekturen
- Softwarearchitektur



Der Projektkontext

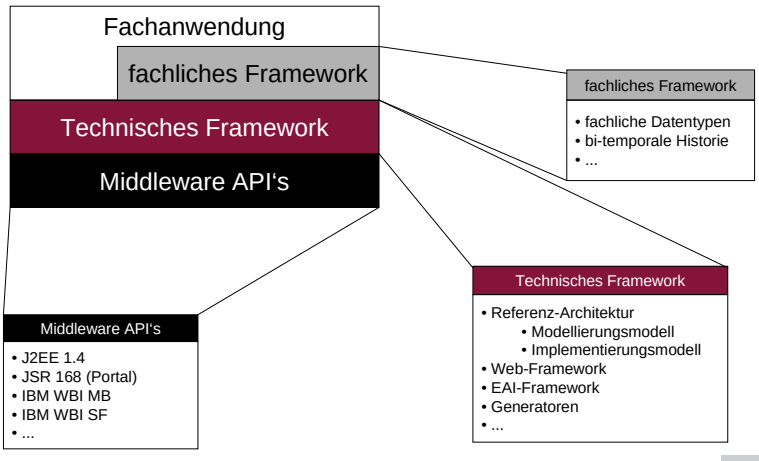
msg systems
.consulting .solutions .partnership

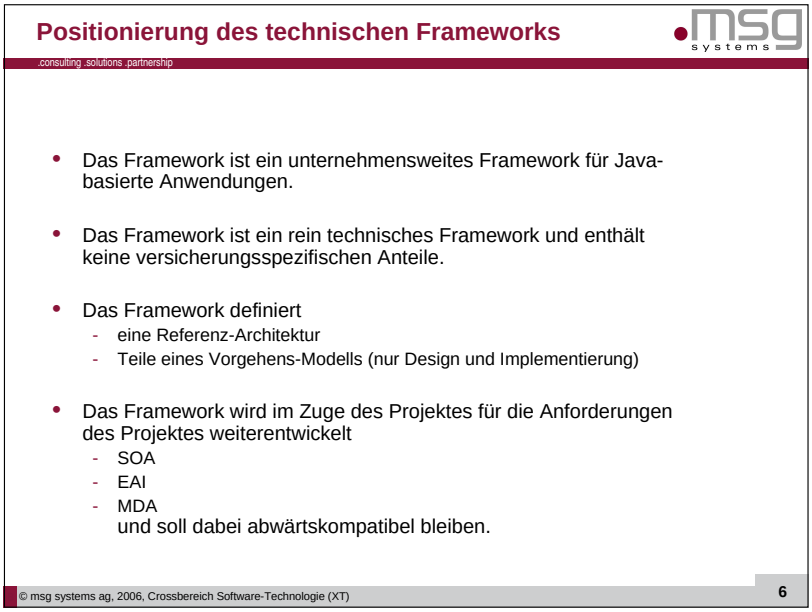
- In dem Projekt realisiert die msg ein versicherungstechnisches Kernsystem in einer SOA auf Basis von JEE-Technologie.
- Parallel werden von der msg diverse kleinere Querschnittssysteme in derselben Architektur und Technologie ersetzt.
- Der Zugriff auf weitere Systeme findet über idealisierte Fassaden statt. Die dahinterliegenden Altsysteme werden mittels EAI-Technologie angeschlossen.
- Diese „Neue Welt“ kennt ca. 30 Subsysteme, die ca. 130 Softwarekomponenten enthalten (Planungsstand), die über Prozesse orchestriert werden.

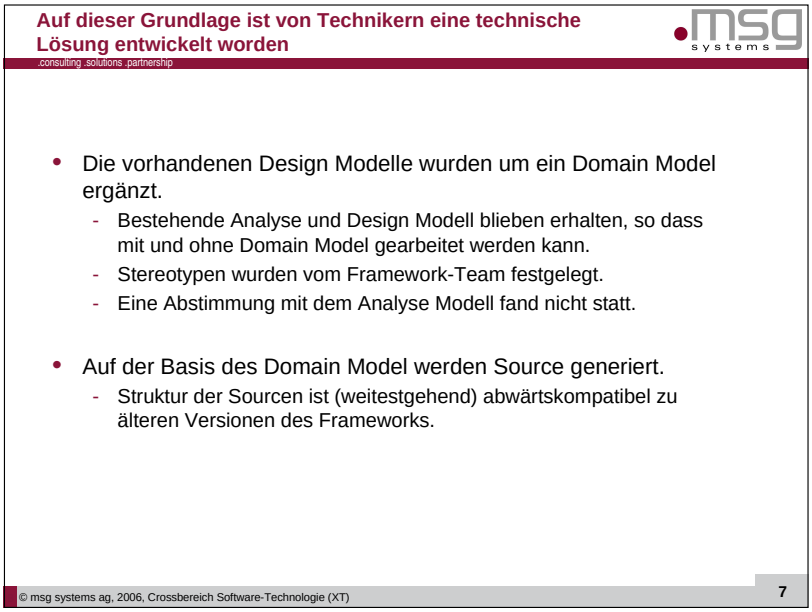


Technische Hierarchie

msg systems
.consulting .solutions .partnership




Positionierung des technischen Frameworks
msg systems
.consulting .solutions .partnership
Das Framework ist ein unternehmensweites Framework für Java-basierte Anwendungen.
Das Framework ist ein rein technisches Framework und enthält keine versicherungsspezifischen Anteile.
Das Framework definiert
- eine Referenz-Architektur
- Teile eines Vorgehens-Modells (nur Design und Implementierung)
Das Framework wird im Zuge des Projektes für die Anforderungen des Projektes weiterentwickelt
- SOA
- EAI
- MDA
und soll dabei abwärtskompatibel bleiben.
© msg systems ag, 2006, Crossbereich Software-Technologie (XT)
6


Auf dieser Grundlage ist von Technikern eine technische Lösung entwickelt worden
msg systems
.consulting .solutions .partnership
Die vorhandenen Design Modelle wurden um ein Domain Model ergänzt.
- Bestehende Analyse und Design Modell blieben erhalten, so dass mit und ohne Domain Model gearbeitet werden kann.
- Stereotypen wurden vom Framework-Team festgelegt.
- Eine Abstimmung mit dem Analyse Modell fand nicht statt.
Auf der Basis des Domain Model werden Source generiert.
- Struktur der Sourcen ist (weitestgehend) abwärtskompatibel zu älteren Versionen des Frameworks.
© msg systems ag, 2006, Crossbereich Software-Technologie (XT)
7

Aus diesem Vorgehen ergaben sich eine Reihe von Problemen

.consulting .solutions .partnership

- Übergang von Analyse ins Design gestaltete sich schwierig.
- Eine Trennung von Basis-Generatoren und fachlichen Erweiterungen war nicht möglich.
- Iteratives, inkrementelles und mehrfaches Generieren funktionierte nicht reibungslos.
- Aufwand für die Modellierung stieg stärker als erwartet, während der Aufwand für die Implementierung weniger stark sank als erwartet.

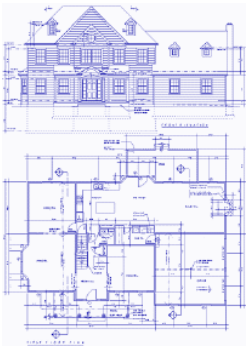
Hieraus folgte, dass

- die erwartete Effizienzsteigerung nicht spürbar wurde und
- die Aufwände für das Framework gestiegen sind, sodass
- im Projekt Zweifel am Vorgehen standen sind.

Unsere Motivation an MDA festzuhalten

.consulting .solutions .partnership

- Fachliche Komplexität für Einzelne nicht beherrschbar
  - Komplexes Versicherungskernsystem
  - Anbindung bzw. Erneuerung vieler Neben- und Querschnittssysteme
- Technische Komplexität für Einzelne nicht beherrschbar
  - J2EE 1.4
  - Portal
  - Workflow Engine
  - MQ Message Broker
  - Umsysteme: Cobol, Smalltalk, C, ABAP, Assembler
  - + Tooling



➔ Die Größe des Projektes erfordert eine ingenieurmäßige Planung.
  - Entwurf von Lösungsmustern für wiederkehrende Anforderungen
  - Berücksichtigung von Abhängigkeiten
  - Planung der Tragfähigkeit

➔ Die Größe des Projektes erfordert eine „industrielle Fertigung“
  - Automatische Serienfertigung der Technik durch Generatoren.
  - Qualitätsmanagement durch den Einsatz identischer Lösungen für identische Probleme.
  - Fokussierung der fachlichen Implementierung auf das lokale fachliche Problem.

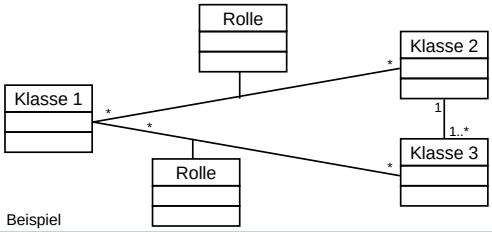
Zusammengefasst:

Wir haben keine andere Chance, als mit MDA zu erfolgreich zu sein!

Übergang von Analyse ins Design gestaltete sich schwierig – Ursachen

.consulting .solutions .partnership

- Sprache des Domain Model war technisch geprägt
  - z.B. <<domainklasse>>, <<domainklasseextern>>, <<produktmodellklasse>> oder <<journal>>
- Analyse Modell war zu unspezifisch
  - Beziehungen waren zu allgemein modelliert
  - Keine Fachlichen Muster vorhanden

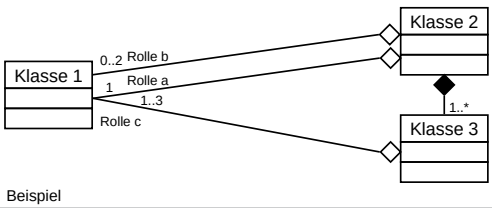


Beispiel

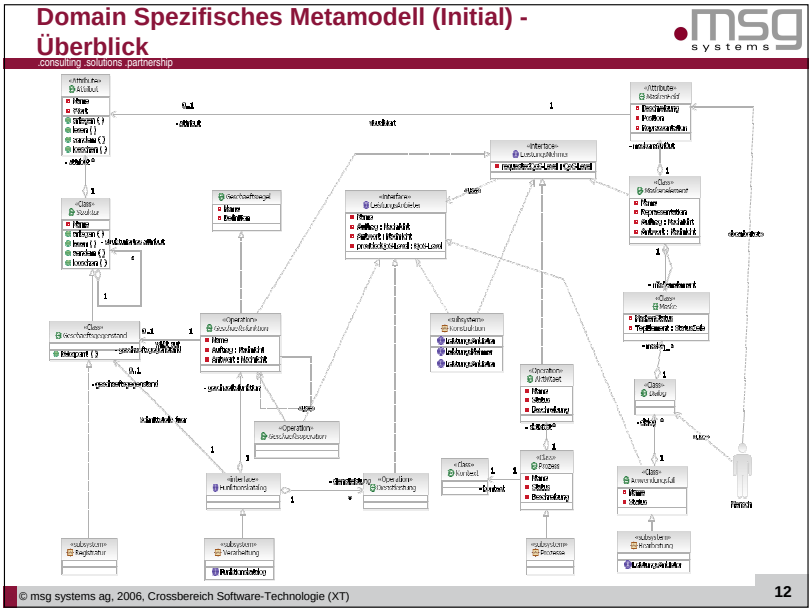
Übergang von Analyse ins Design gestaltete sich schwierig – Maßnahmen

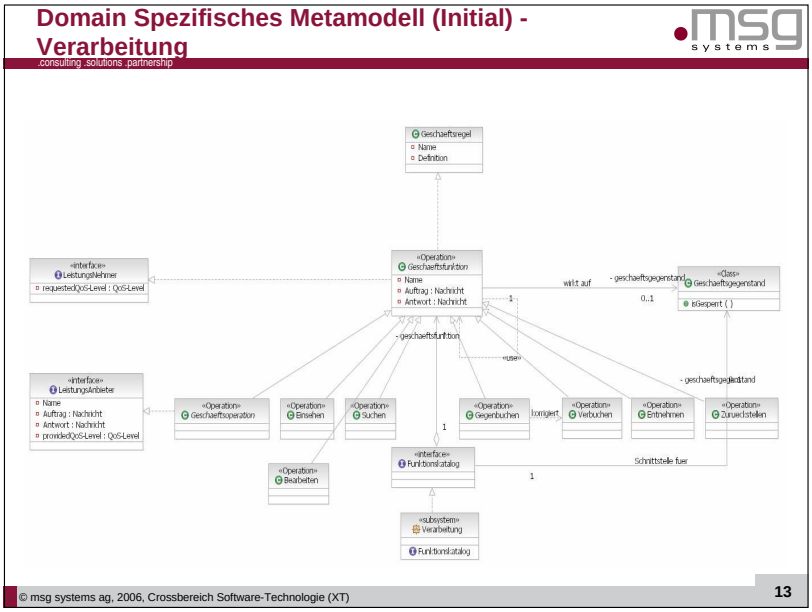
.consulting .solutions .partnership

- Schärfung der Analyse Modelle
  - Detaillierte Modellierung
  - Systematische Identifikation von fachlichen Mustern
- Einführung einer fachlich motivierten Domain Spezific Language
  - Entwicklung eines allgemeinen Meta-Modells
  - Kontinuierliche Erweiterung um Bezeichnungen fachlicher Muster



Beispiel

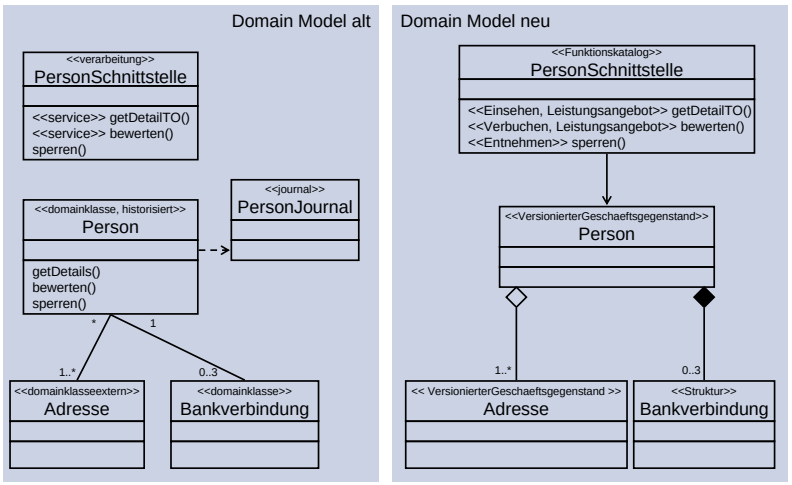

Domain Spezifisches Metamodell (Initial) - Überblick
.consulting .solutions .partnership
msg systems
© msg systems ag, 2006, Crossbereich Software-Technologie (XT)
12


Domain Spezifisches Metamodell (Initial) - Verarbeitung
.consulting .solutions .partnership
msg systems
Geschäftsregel
Name
Definition
«Operation»
Geschäftsfunktion
Name
Auftrag : Nachricht
Antwort : Nachricht
«interface»
LeistungsNehmer
requestedQoS-Level : QoS-Level
«interface»
LeistungsAnbieter
Name
Auftrag : Nachricht
Antwort : Nachricht
providedQoS-Level : QoS-Level
«Class»
Geschäftsgegenstand
isExpert ( )
wirkt auf
- geschäftsgegenstand
0..1
- geschäftsfunktion
«use»
- geschäftsgegenstand
«Operation»
Geschäftsoperation
«Operation»
Einsehen
«Operation»
Suchen
«Operation»
Gegenbuchen
«Operation»
Verbuchen
«Operation»
Entnehmen
«Operation»
Zurückstellen
«Operation»
Bearbeiten
korrigiert
«interface»
Funktionskatalog
Schnittstelle fuer
«subsystem»
Verarbeitung
Funktionskatalog
© msg systems ag, 2006, Crossbereich Software-Technologie (XT)
13

Unterschied in der Modellierung am Beispiel

msg systems

.consulting .solutions .partnership



Eine Trennung von Basis-Generatoren und fachlichen Erweiterungen war nicht möglich – Ursachen

msg systems

.consulting .solutions .partnership

- Unzureichende Erfahrung im Einsatz von Generatoren.
  - Konzeptionelle Fehler infolge „Learning by Doing“, die in Folge Termindrucks nicht korrigiert wurden.
- Unzureichender Funktionsumfang des Generators.
  - Keine Model to Model Transformationen möglich.
  - Keine „ARound“- Funktionalität vorhanden.
- Ungeeignetes Design des Source-Codes.
  - Durch Aufrechterhaltung der Abwärts Kompatibilität konnte keine ausreichende Trennung von Generat und User-Code erfolgen.

Eine Trennung von Basis-Generatoren und projektspezifischen Erweiterungen war nicht möglich – Maßnahmen

msg systems

.consulting .solutions .partnership

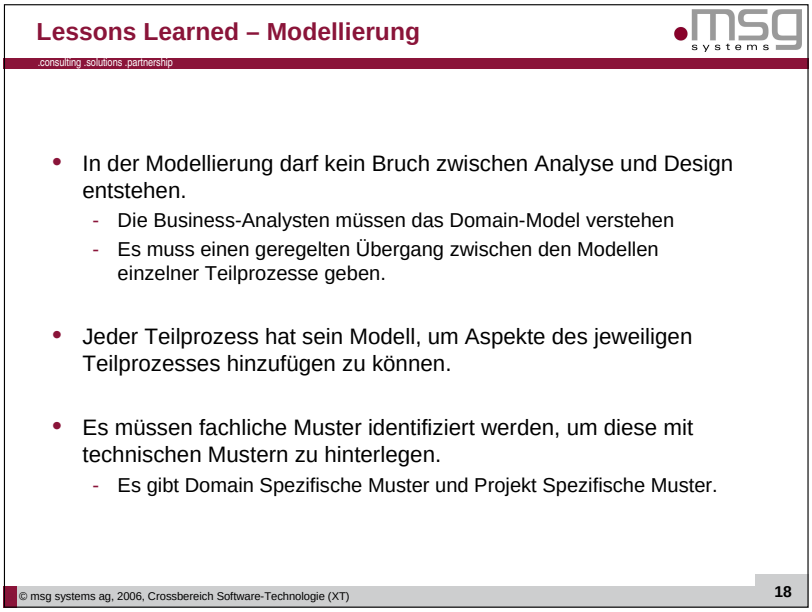
- Umstieg auf eine neue Version der Generators (oAW)
- Berücksichtigung des Aspektes Generierbarkeit im Design
  - Massiver Einsatz des PoJO-Patterns zur Trennung von technischem und fachlichem Code.
  - Durchgängiger Einsatz von Business-Interfaces und BusinessDelegates um technische Aspekte weitgehend aus der fachlichen Implementierung zu entfernen.
- Systematisches Reengineering der Generator-Templates
  - Trennung von Basis und Erweiterungen in verschiedene Templates.
  - Berücksichtigung der Anforderungen hinsichtlich iteratives, inkrementelles und mehrfaches Generieren.

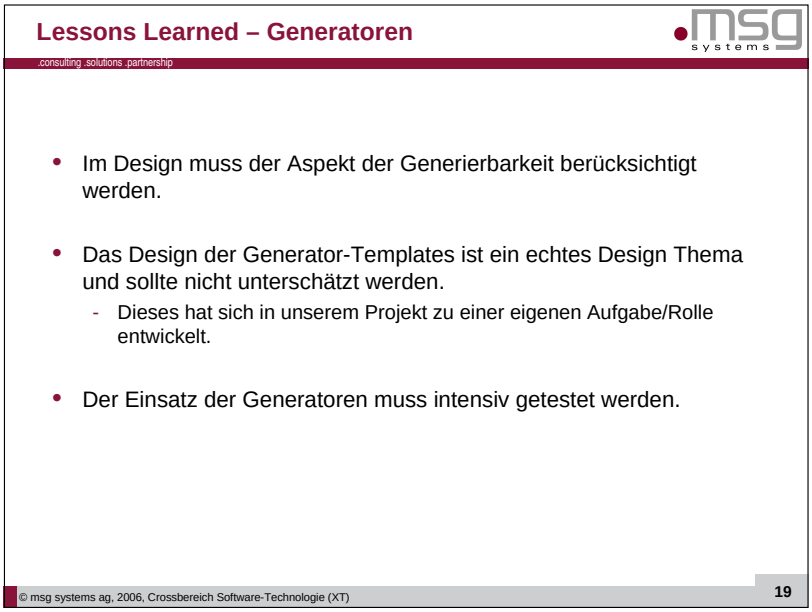
Zweifel am Vorgehen im Projekt - Maßnahmen

msg systems

.consulting .solutions .partnership

- Aufsetzen eines Methodischen Prototypen
  - Verprobung der eingeleiteten Maßnahmen
    - Agiler Verbesserungsprozess
  - End-to-End Betrachtung der Teilprozesse (Analyse, Design, Implementierung) im Projekt und insbesondere der Übergänge
    - Identifikation bislang verborgender Anforderungen
    - Schärfung der Arbeitsanweisungen
  - Validierung von Aufwandskenngrößen


Lessons Learned – Modellierung
msg systems
.consulting .solutions .partnership
In der Modellierung darf kein Bruch zwischen Analyse und Design entstehen.
Die Business-Analysten müssen das Domain-Model verstehen
Es muss einen geregelten Übergang zwischen den Modellen einzelner Teilprozesse geben.
Jeder Teilprozess hat sein Modell, um Aspekte des jeweiligen Teilprozesses hinzufügen zu können.
Es müssen fachliche Muster identifiziert werden, um diese mit technischen Mustern zu hinterlegen.
Es gibt Domain Spezifische Muster und Projekt Spezifische Muster.
© msg systems ag, 2006, Crossbereich Software-Technologie (XT)
18


Lessons Learned – Generatoren
msg systems
.consulting .solutions .partnership
Im Design muss der Aspekt der Generierbarkeit berücksichtigt werden.
Das Design der Generator-Templates ist ein echtes Design Thema und sollte nicht unterschätzt werden.
Dieses hat sich in unserem Projekt zu einer eigenen Aufgabe/Rolle entwickelt.
Der Einsatz der Generatoren muss intensiv getestet werden.
© msg systems ag, 2006, Crossbereich Software-Technologie (XT)
19

Lessons Learned – Wieviel Modellgetriebenheit braucht bzw. verträgt ein Projekt?

msg systems

.consulting .solutions .partnership

- Große Projekte müssen ingenieurmäßig geplant werden, für sie ist MDA unerlässlich!
- Derzeit gibt es noch keine „out-of-the-box“ Domain Spezifischen MDA Toolkits.
  - Es gibt einen erheblichen Implementierungsaufwand.
  - Domain Spezifische Anpassungen sind heute i.d.R. wiederverwendbar.
  - Kleine bis mittelgroße Projekte können sich z.Z. keine vollausgebaute MDA leisten, wenn für das Projekt aufgebaut werden muss.
- Jedes Projekt muss (leider) sein eigenes Maß finden.
- Die msg arbeitet zur Zeit eine Basis für MDA in das eigene Vorgehensmodell msg.PROFI ein.

Vielen Dank für Ihre Aufmerksamkeit !

msg systems

.consulting .solutions .partnership

4.2.3 Diskussion

- *Meta-Modell Änderungen / Modell-Migration*
  Derzeit gibt es in noch keiner DSL-Entwicklungsumgebung eine Unterstützung bei Änderungen des Meta-Modells. Dies hat zur Folge, dass man bereits existierende Modelle manuell an das veränderte Meta-Modell anpassen muss.

- *Standardisierung von Domänen-spezifischen Sprachen*
  Durch den Einsatz von DSL-Technologien läuft man Gefahr, dass jede Firma ihre eigenen Modellierungstechniken für ihre Domänen definiert. Beispielsweise Brachen-einheitliche Standardisierung wäre hierbei sicherlich wünschenswert, lässt sich aber meist nur sehr schwer realisieren. Durch Standardisierung würde die Austauschbarkeit von Modellen erleichtert werden und Skaleneffekte bei der Entwicklung von Entwicklungsumgebungen für branchenweite Sprachen entstehen. Derartige Sprachen hätten auch den Vorteil, dass langfristig Branchenkundige mit diesen Sprachen vertraut sind und dadurch wiederholte Einarbeitungszeit in unternehmensspezifische Sprachen erspart bleiben würde.
  Standardisierungsbemühungen sind meist sehr zeit- und kostenintensiv, da ein Konsenz zwischen vielen Parteien gefunden werden muss. Dabei ist speziell im Kontext der Domänen-spezifischen Sprachen zu befürchten, dass Standardisierung den Scope einer DSL vergrößert und dadurch ihren Abstraktionslevel und ihre Prägnanz senkt.

- *Quantifizierung des Nutzens von DSL-Technologie*
  Der wirkliche Nutzen des Einsatzes von DSL-Technologie lässt sich derzeit in der Praxis noch nicht zuverlässig quantifizieren, da diese Technologien noch einen zu geringen Verbreitungsgrad besitzen und nicht auf nennenswerte Erfahrungswerte zurückgegriffen werden kann. Aus bisherigen Projekten werden allerdings oftmals gerade Vorteile aufgrund der Verwendung von Fach- anstatt Technologie-spezifischer Terminologie berichtet, welche die Kommunikation zwischen Domänen- und Technologieexperten erleichtert.

4.3 Dr. Markus Pizka, Elmar Jürgens: „Evolutionäre Sprachentwicklung“

4.3.1 Abstract

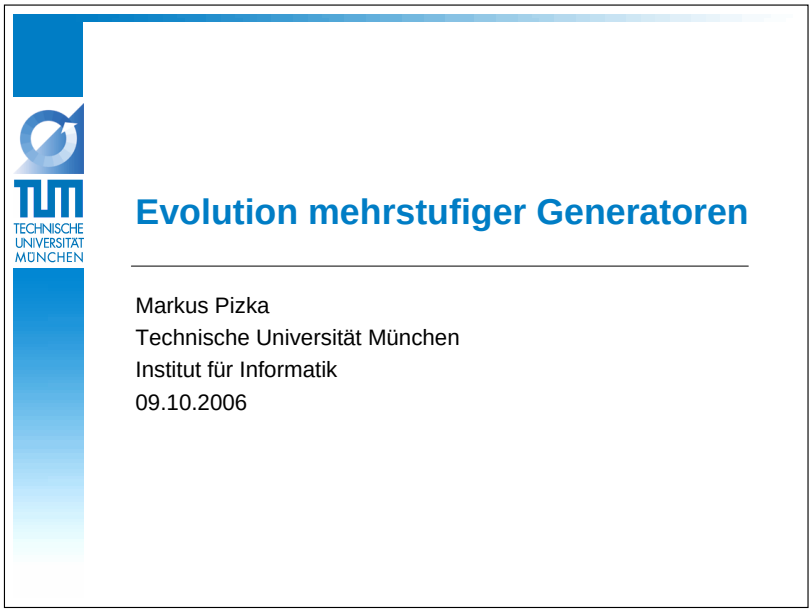
Durch ihren hohen Spezialisierungsgrad versprechen Domänen-spezifische Sprachen höheren Produktivitätsgrad und kürzere Entwicklungszeiten als Allzweckprogrammiersprachen. Da viele Domänen allerdings kontinuierlichem Wandel unterliegen, müssen die zugehörigen DSLs unweigerlich kontinuierlich angepasst werden, um ihren Wert zu bewahren.

Die kontinuierliche Evvolution einer DSL kann selbst jedoch sehr aufwändig und teuer sein, da ihr Compiler und alle bestehenden Wörter (z.B. Instanzen, Programme) entsprechend den Änderungen an der DSL Spezifikation angepasst werden müssen. Diese Wartungskosten gefährden die erwartete Ersparnis der Entwicklungskosten und beschränken daher den Erfolg von DSLs in der Praxis.

Dieser Vortrag präsentiert ein Konzept und ein Werkzeug für die evolutionäre Entwicklung Domänen-spezifischer Sprachen, das zum Ziel hat, Wartungsaufwand in der evolutionären Sprachentwicklung maßgeblich zu senken.

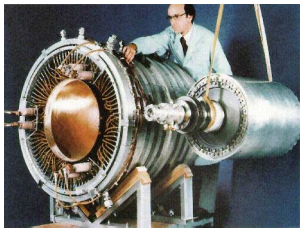
4.3.2 Folien

Dr. Markus Pizka: „Evolution mehrstufiger Generatoren“



Überblick

- Grundlagen der Programm-Generierung
- Multi-Level Generatoren
- Programm Co-Evolution



Programm Generierung

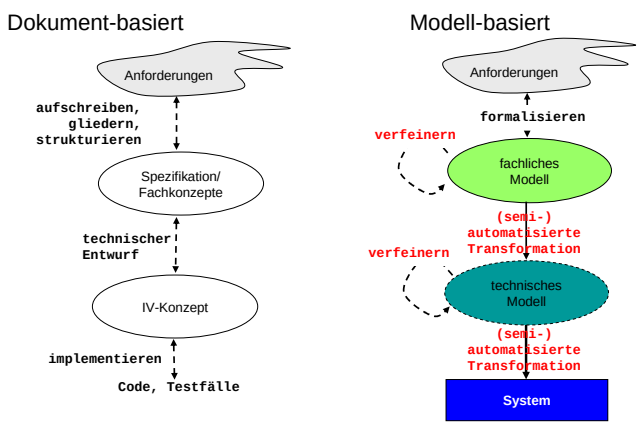
- Beschleunigung der Software-Entwicklung
  ⇨Erhöhung der Wiederverwendung von Komponenten
- Bewahren / Erhöhen der Qualität von Produkten
  McIlroy 68: “*No user of a particular member of a family should pay a penalty, in unwanted generality …*”
  ⇨Komponenten an Nutzungskontext anpassen
- Wartungskosten reduzieren
  ⇨Umfang reduzieren
  ⇨Konsistenz sichern

Ausprägungen (1)

- Makrosubstitution, z.B. C Präprozessor
  - `#define min(X, Y)  ((X) < (Y) ? (X) : (Y))`
  - `#define COMMAND(NAME) { #NAME, NAME ## _command }`
    ```
    struct command commands[ ] = {
            COMMAND (quit),
            COMMAND (help), ... };
    ```
- C++ Template Programmierung
  - ```
    template <class T> T max(T a, T b) {
       return a > b ? a : b ; }
    ```
- Domänenspezifische Sprachen
  - Bsp.: SQL, GUI-Sprachen, partielle Differentialgleichungen, …
  - Domänen-Analyse → Domänen-Modell → Sprache → Generator

09.10.2006 Markus Pizka 4

Ausprägungen (2) – Model …



Formale Modelle Voraussetzung für Generierung!

09.10.2006 Markus Pizka 5
    ```

Irrelevant

Trennung von Modell und Programm

- es gibt nur Modelle
- meistens textuell repräsentiert (zunehmend Grafik)
- ohnehin Worte einer formalen Sprache

Statik versus Dynamik

- aktuelle Grenze Laufzeit/Übersetzung historisch
- konzeptionell Frage der Auswertungsstrategie

Korrektheit

- wichtig aber wird anderen überlassen
- Konzentration: Umgang mit großem Volumen

09.10.2006 Markus Pizka 6

Programm-Transformation

Programm P: strukturierte Einheit mit Verhalten (Semantik)

- Struktur erlaubt Umformung von P in P´
- Semantik erlaubt
  - Vergleich von P und P´
  - Beurteilung der Zulässigkeit der Transformation
- Semantik hier: von außen beobachtbares Verhalten

Programm-Transformation ist das Überführen eines Programms P in ein anderes Programm P´

t: Q → Z; t(P) = P‘

P ∈ Q (Quellsprache), P‘∈ Z (Zielsprache)

09.10.2006 Markus Pizka 7

Babel

- weitere Bezeichnungen:
  - Meta programming
  - Software generation
  - Generative programming
  - Program synthesis
  - Program refinement
  - Program calculation
  - Refactoring
- charakteristisches Merkmal Q = Z ?

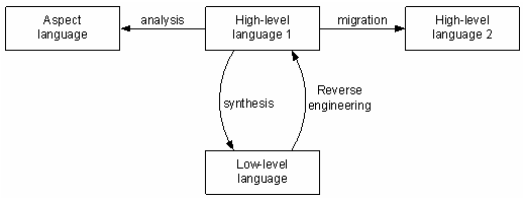
  Q ≠ Z: Übersetzung (Translation)

  Q = Z: Umformulierung (Rephrasing)

Q = Z ⇨ Umformulierung

- Gleiches in anderen Worten ausdrücken (i.d.R. Änderung der Semantik)
- Varianten
  - Normalisierung ~ Überführung in einheitliche Form
  - Optimierung (Zeit/Platz), z.B.
    - Falten von Konstanten, 3 + 5 ⇨ 8
    - Eliminieren von nicht erreichbarem Code
  - Refactoring ~ Verbesserung des Designs
  - Verschleierung ~ Lesbarkeit vermindern
  - Reflektion ~ Berechnung einer P-Eigenschaft z.B. Ausführungsprotokoll
  - Renovierung ~ Fehlerbehebung, Restrukturierung

Q ≠ Z ⇨Übersetzung



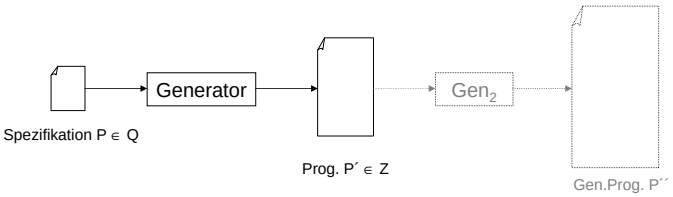
sei A(L) Abstraktionsniveau der Sprache L

- Migration: A(Q) = A(Z); z.B. C++ → Java
- Reverse Engineering: A(Q) < A(Z)
- Programm-Analyse: Z ⊂ Q (d.h. Konzentration auf einen Aspekt)
- Synthese: A(Q) > A(Z); z.B. Java-Übersetzer

09.10.2006 Markus Pizka 10

Programm-Synthese

Idee: Generierung von Programmen aus Spezifikation auf höherer Abstraktionsordnung



Chancen:

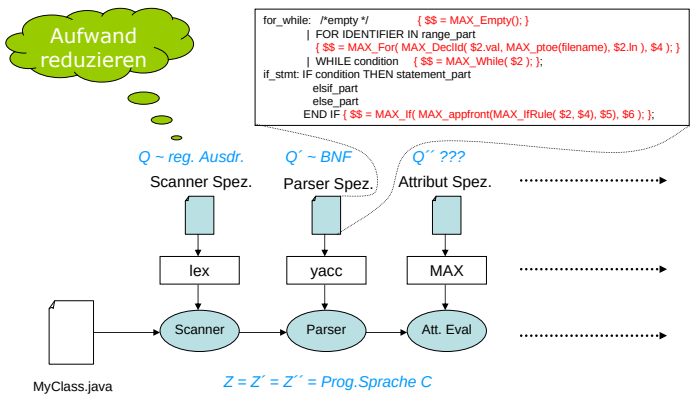
- Qualität: Fehler bei manueller Verfeinerung verhindern
- Produktivität: falls |P| < |P´| < |P´´|

09.10.2006 Markus Pizka 11

A(Q) > A(Z) Risiken

- Q abhängig vom Generator
  Änderung Generator ⇨ Änderung aller q ∈ Q
- Q mit wachsendem A.-Niveau domänenspezifisch
  - keine schlanke high-level Universalsprache
  - Kenntnis zahlreicher verschiedener Sprachen notwendig
  - Sinkende Stabilität
- Automatisierung limitiert (s. Berechenbarkeit)

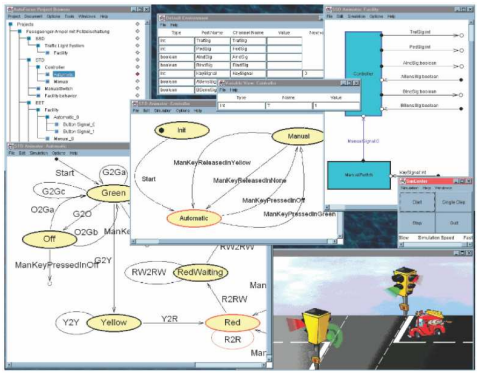
Success Story Übersetzerbau



Success Story Embedded

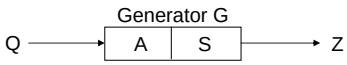
Korrektheit

Code-Generierung aus formalen Modellen (z.B. Autofocus)



09.10.2006 Markus Pizka 14

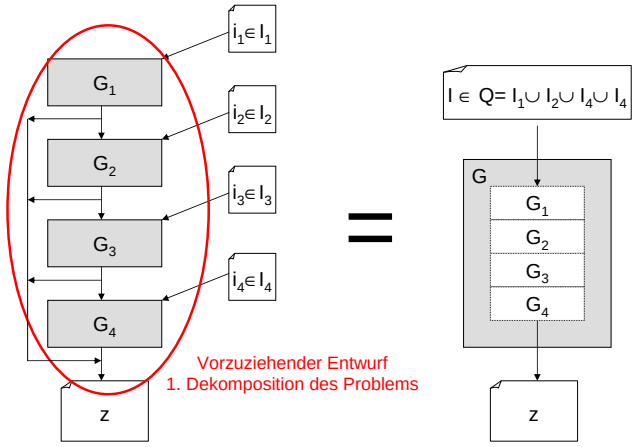
Programm-Generatoren



- Analyse
  - Information beschaffen ~ Elemente, Abh., Bedingungen, …
- Synthese
  - Transformations-Strategie die z aus {q, A} generiert
- Nutzen
  - Abstraktionsniveau L(Q) > L(Z)
  - **Treffen optimierter Entscheidungen durch S**
    keine unmodifizierter Reuse sondern Alternativen verweben
- Konflikt
  - L(Q) = L(Z): keine Entscheidungen (Migration): kein Nutzen
  - L(Q) >> L(Z): wenig Info, schwierige Entscheidung: niedrige Qualität

09.10.2006 Markus Pizka 15

Abgestufte Generierung

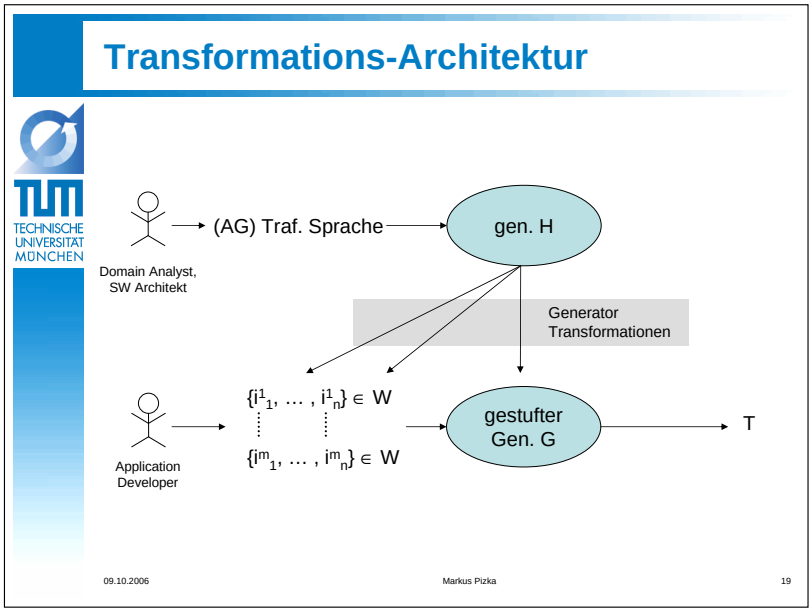


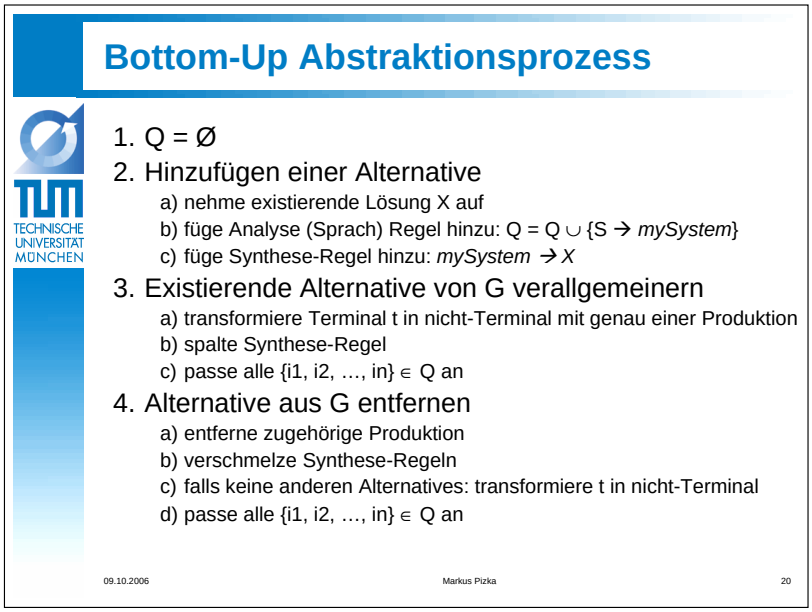
09.10.2006 Markus Pizka 16

Woher kommt G_i?

- Wie werden die Generatoren selbst entwickelt?
  - Je abstrakter desto schwieriger zu entwickeln
  - Je abstrakter desto spezifischer
  - Je spezifischer desto geringer ist die Wiederverwendung
- Wie werden die Generatoren gepflegt?
  - Entscheidungsfindung selbst wird sich über die Zeit ändern
  - Was passiert mit den bereits existierenden $q \in Q$?
- Ansatz:
  generiere G_i mit Generator-Generator H

09.10.2006 Markus Pizka 18


Transformations-Architektur
TECHNISCHE UNIVERSITÄT MÜNCHEN
(AG) Traf. Sprache
gen. H
Domain Analyst, SW Architekt
Generator Transformationen
{i1 1, … , i1 n} ∈ W
{im 1, … , im n} ∈ W
gestufter Gen. G
T
Application Developer
09.10.2006
Markus Pizka
19


Bottom-Up Abstraktionsprozess
TECHNISCHE UNIVERSITÄT MÜNCHEN
1. Q = Ø
2. Hinzufügen einer Alternative
a) nehme existierende Lösung X auf
b) füge Analyse (Sprach) Regel hinzu: Q = Q ∪ {S → mySystem}
c) füge Synthese-Regel hinzu: mySystem → X
3. Existierende Alternative von G verallgemeinern
a) transformiere Terminal t in nicht-Terminal mit genau einer Produktion
b) spalte Synthese-Regel
c) passe alle {i1, i2, …, in} ∈ Q an
4. Alternative aus G entfernen
a) entferne zugehörige Produktion
b) verschmelze Synthese-Regeln
c) falls keine anderen Alternatives: transformiere t in nicht-Terminal
d) passe alle {i1, i2, …, in} ∈ Q an
09.10.2006
Markus Pizka
20

Co-Evolution

Veränderung von G transformiert Q in Q_{neu}

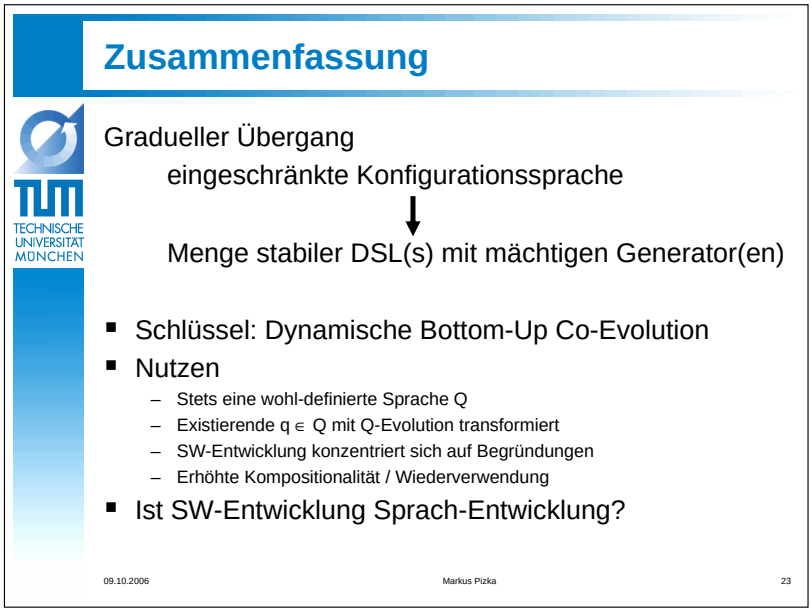
- Grammatik von G (Analyse)
  - automatisch per Definition
    (Eingabe von H ist Sprachtransformation)
- Transformation existierender Instanzen $q \in Q$
  - Wunsch: (semi-)automatische Transformation
  - Idee: Generierung der Baumtransformation für $Q \rightarrow Q_{neu}$
  - Erstellung und Transformation der Parse-Bäume aller $q \in Q$
- Transformation der Synthese-Regeln
  - Semantik der Sprache ~ inhärent manuell
  - Werkzeuggestützt durch Impact-Analyse

09.10.2006 Markus Pizka 21

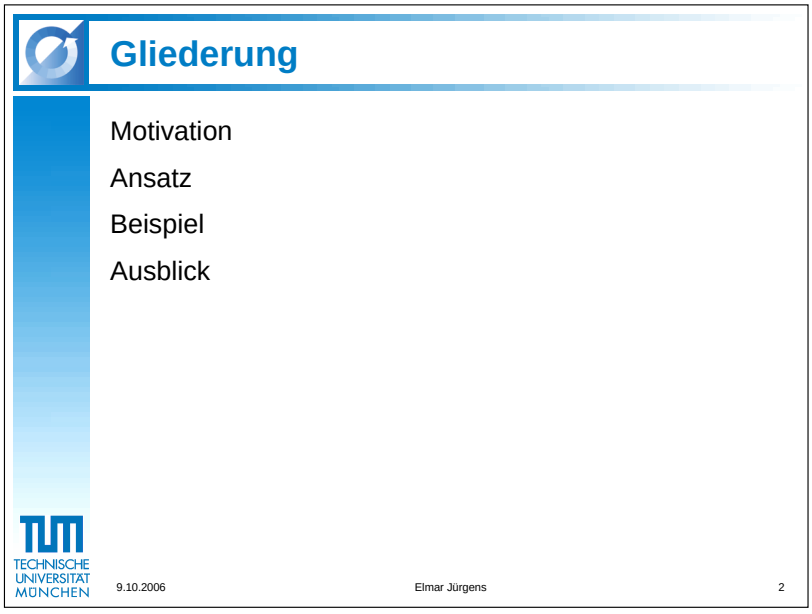
Pushing The Syntax Limits

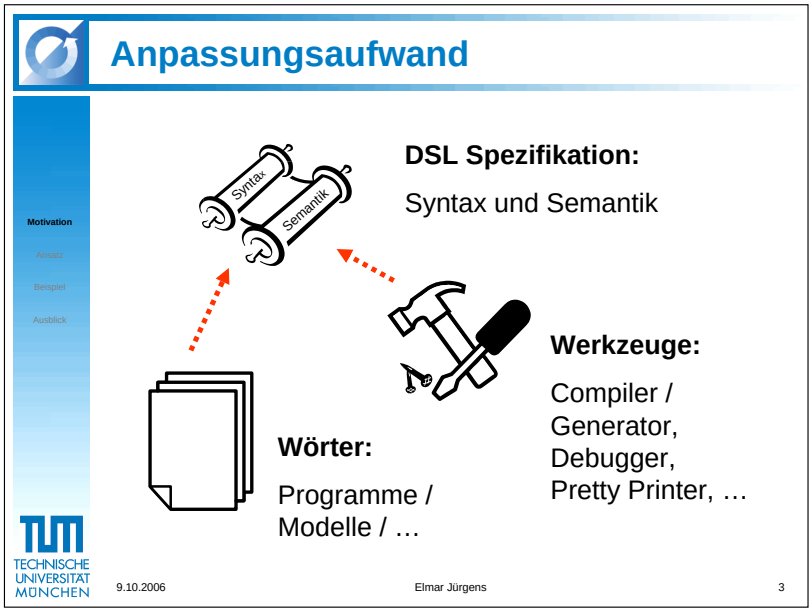
- Syntax
  - Analyse-Anteil von G
  - Voll-automatische Transformation (kfS)
- Semantik
  - Synthese-Teil von G
  - Attributgrammatik / -Auswertung
  - Manuelle Transformation
- Strategie
  - Strikte Trennung von Sytax und Semantik
  - So viel wie möglich über Syntax abdecken (kfG)

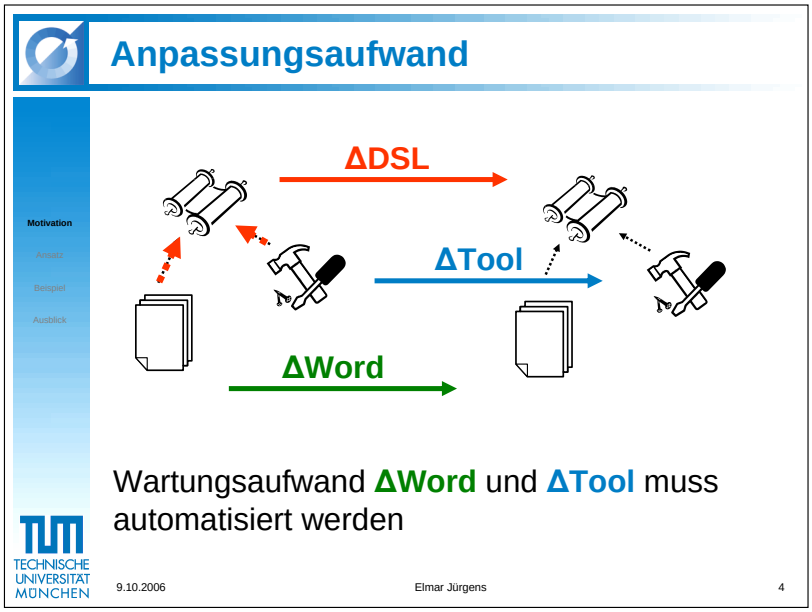
09.10.2006 Markus Pizka 22

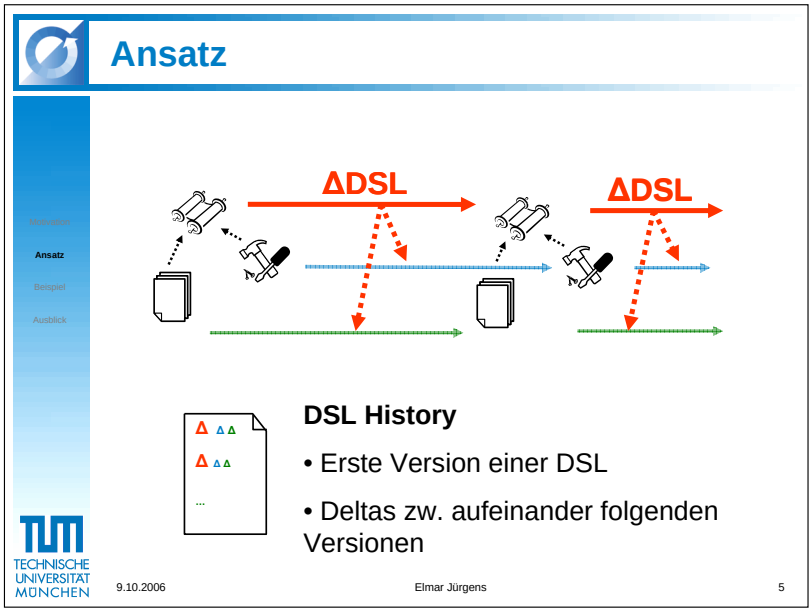

Zusammenfassung
TECHNISCHE UNIVERSITÄT MÜNCHEN
Gradueller Übergang
eingeschränkte Konfigurationssprache
Menge stabiler DSL(s) mit mächtigen Generator(en)
Schlüssel: Dynamische Bottom-Up Co-Evolution
Nutzen
Stets eine wohl-definierte Sprache Q
Existierende q ∈ Q mit Q-Evolution transformiert
SW-Entwicklung konzentriert sich auf Begründungen
Erhöhte Kompositionalität / Wiederverwendung
Ist SW-Entwicklung Sprach-Entwicklung?
09.10.2006
Markus Pizka
23

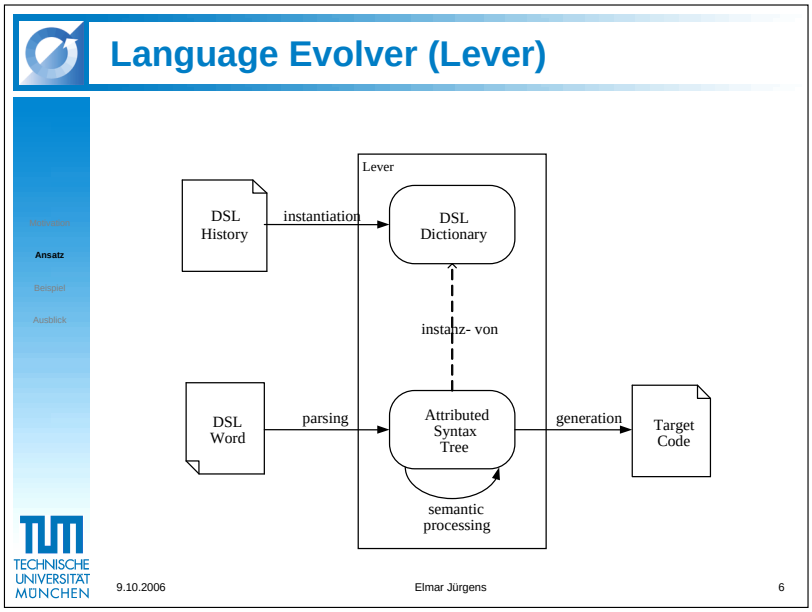
Elmar Juergens: „Werkzeuggestützte Sprachevolution“




Anpassungsaufwand
Motivation
Ansatz
Beispiel
Ausblick
Syntax
Semantik
DSL Spezifikation:
Syntax und Semantik
Werkzeuge:
Compiler /
Generator,
Debugger,
Pretty Printer, …
Wörter:
Programme /
Modelle / …
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
3


Anpassungsaufwand
Motivation
Ansatz
Beispiel
Ausblick
ΔDSL
ΔTool
ΔWord
Wartungsaufwand ΔWord und ΔTool muss automatisiert werden
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
4


Ansatz
Motivation
Ansatz
Beispiel
Ausblick
ΔDSL
ΔDSL
DSL History
• Erste Version einer DSL
• Deltas zw. aufeinander folgenden Versionen
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
5


Language Evolver (Lever)
Motivation
Ansatz
Beispiel
Ausblick
Lever
DSL History
instantiation
DSL Dictionary
instanz- von
DSL Word
parsing
Attributed Syntax Tree
generation
Target Code
semantic processing
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
6

Schwierigkeiten

Motivation
Ansatz
Beispiel
Ausblick

Toolanpassung

- Modularisierung von Compilern ungeeignet für Sprachevolution
- Parsing nicht modular
- Ad-Hoc Programmierung von Code Generatoren zu variantenreich für automatische Anpassung

Kopplungsdilemma bei Evolutionsoperationen:

- Separate Evolution fein steuerbar und aufwändig
- Gekoppelte Evolution komfortabel aber unflexibel

TECHNISCHE UNIVERSITÄT MÜNCHEN

9.10.2006 Elmar Jürgens 7

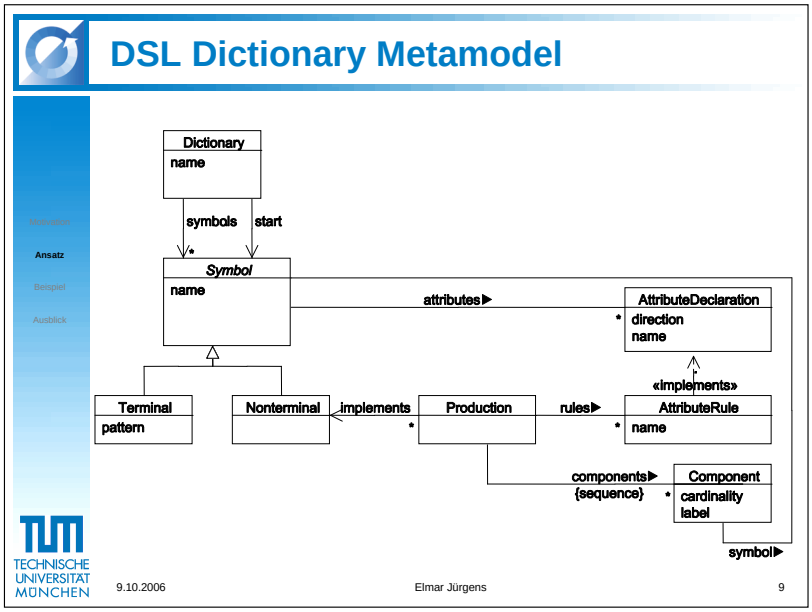
DSL Dictionary

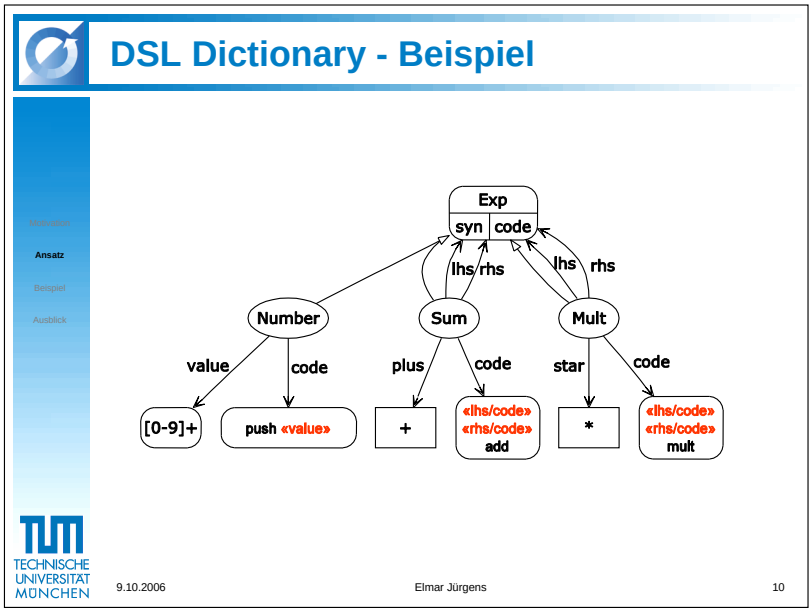
Motivation
Ansatz
Beispiel
Ausblick

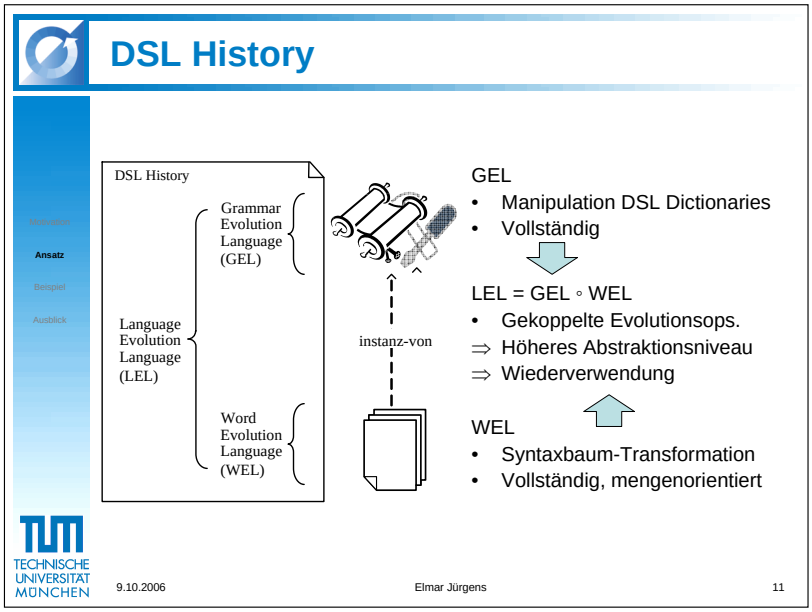
- Vollständige DSL Spezifikation
  - konkrete Syntax
  - abstrakte Syntax
  - statische Semantik
  - translationale Semantik
- Modularisiert entlang von Sprachelementen
- Explizit, veränderbar
- **AXT**: Objektorientierte **A**ttributgrammatiken, validierter **X**Path, **T**emplates

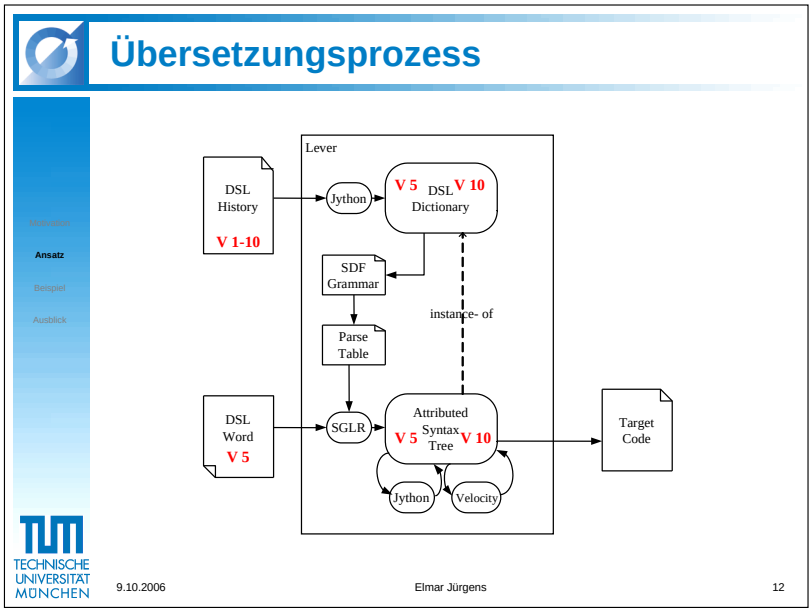
TECHNISCHE UNIVERSITÄT MÜNCHEN

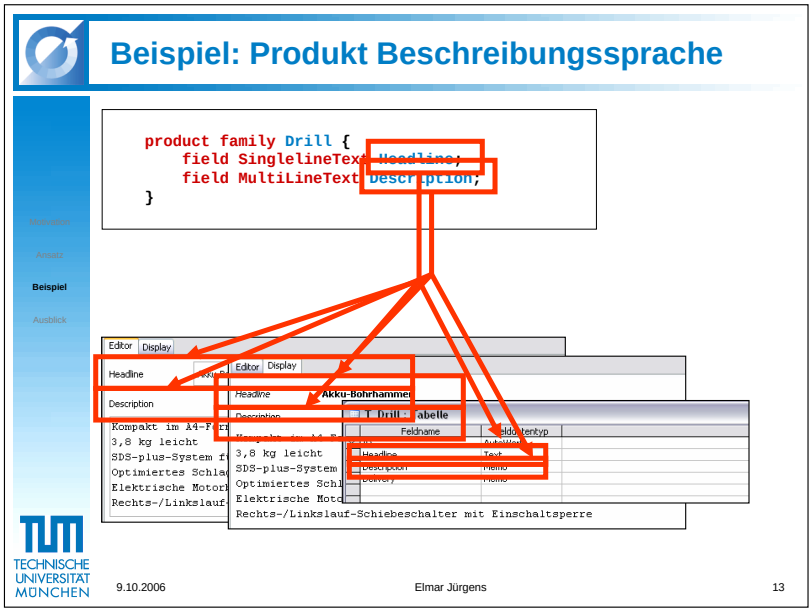
9.10.2006 Elmar Jürgens 8

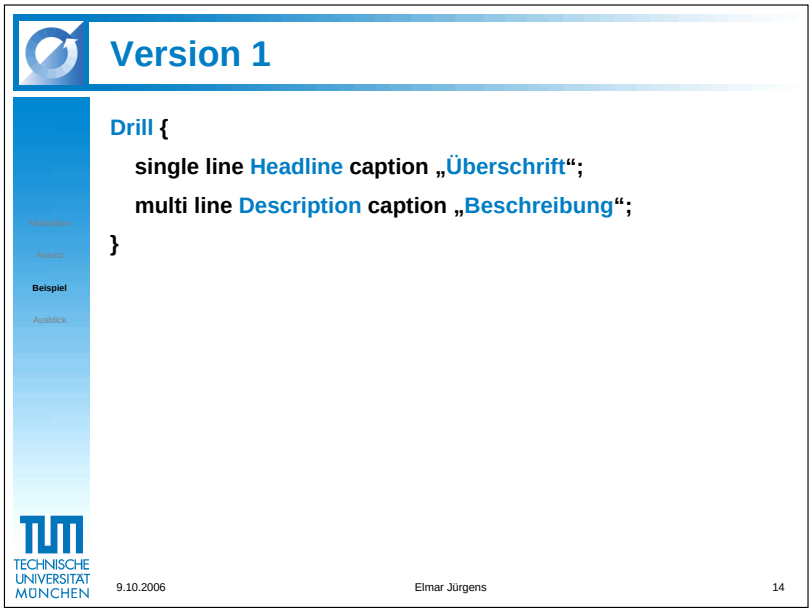

DSL Dictionary Metamodel
Motivation
Ansatz
Beispiel
Ausblick
Dictionary
name
symbols
start
Symbol
name
attributes
AttributeDeclaration
direction
name
«implements»
Terminal
pattern
Nonterminal
implements
Production
rules
AttributeRule
name
components
{sequence}
Component
cardinality
label
symbol
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
9

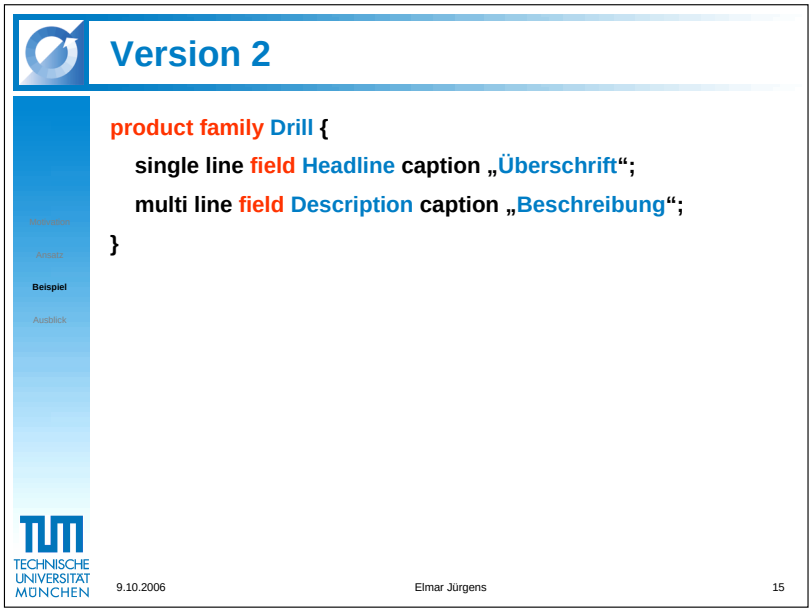

DSL Dictionary - Beispiel
Motivation
Ansatz
Beispiel
Ausblick
Exp
syn
code
lhs
rhs
lhs
rhs
Number
Sum
Mult
value
code
plus
code
star
code
[0-9]+
push «value»
+
«lhs/code»
«rhs/code»
add
*
«lhs/code»
«rhs/code»
mult
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
10

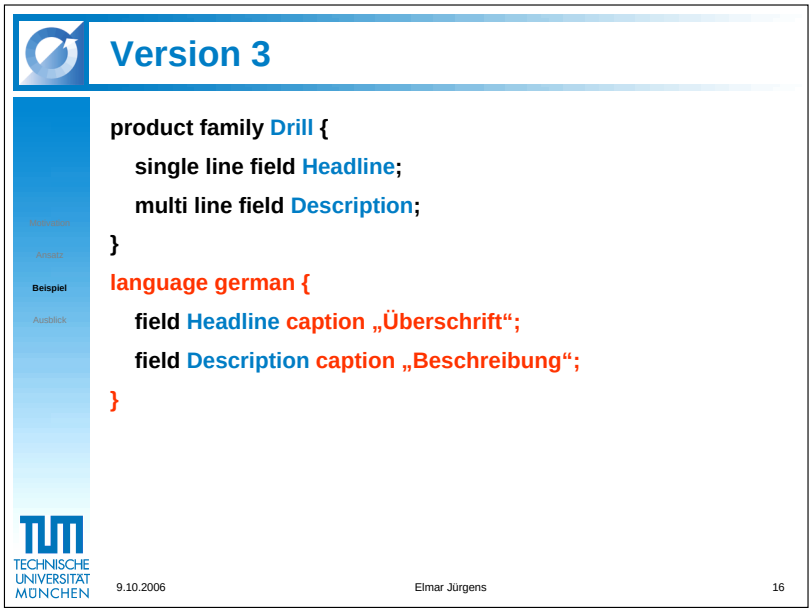

DSL History
Motivation
Ansatz
Beispiel
Ausblick
DSL History
Grammar Evolution Language (GEL)
Language Evolution Language (LEL)
Word Evolution Language (WEL)
instanz-von
GEL
• Manipulation DSL Dictionaries
• Vollständig
LEL = GEL ◦ WEL
• Gekoppelte Evolutionsops.
⇒ Höheres Abstraktionsniveau
⇒ Wiederverwendung
WEL
• Syntaxbaum-Transformation
• Vollständig, mengenorientiert
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
11

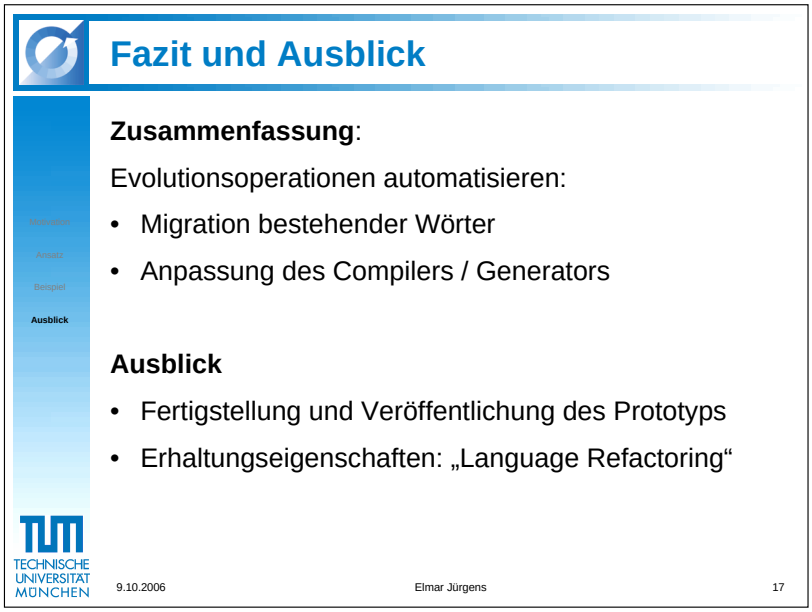

Übersetzungsprozess
Motivation
Ansatz
Beispiel
Ausblick
Lever
DSL History
V 1-10
Jython
V 5 DSL V 10
Dictionary
SDF Grammar
instance- of
Parse Table
DSL Word
V 5
SGLR
Attributed Syntax Tree
V 5 V 10
Target Code
Jython
Velocity
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
12

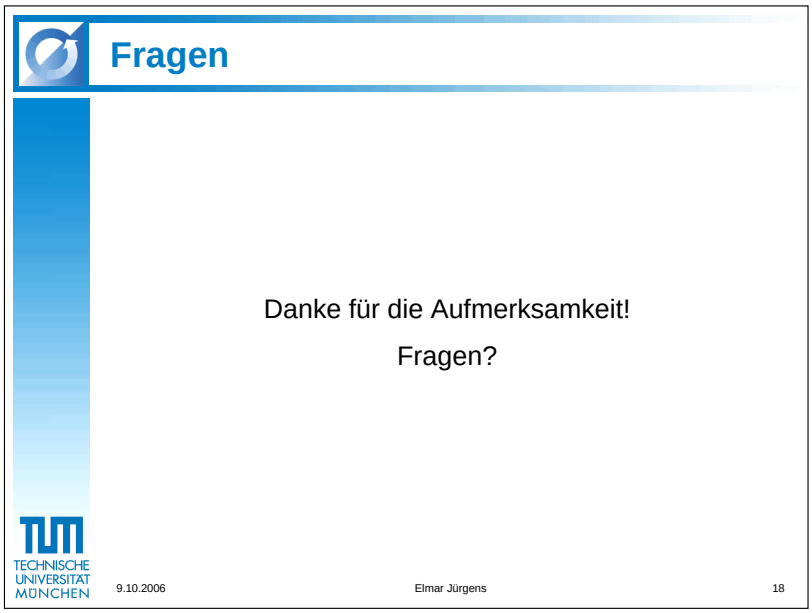

Beispiel: Produkt Beschreibungssprache
product family Drill {
field SinglelineText Headline;
field MultiLineText Description;
}
Motivation
Ansatz
Beispiel
Ausblick
Editor
Display
Headline
Description
Akku-Bohrhammer
3,8 kg leicht
SDS-plus-System
Optimiertes Schl
Elektrische Motor
Rechts-/Linkslauf-Schiebeschalter mit Einschaltsperre
T_Drill - Tabelle
Feldname
Feldtyp
Text
Memo
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
13


Version 1
Drill {
single line Headline caption „Überschrift“;
multi line Description caption „Beschreibung“;
}
Motivation
Ansatz
Beispiel
Ausblick
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
14


Version 2
product family Drill {
single line field Headline caption „Überschrift“;
multi line field Description caption „Beschreibung“;
}
Motivation
Ansatz
Beispiel
Ausblick
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
15


Version 3
product family Drill {
single line field Headline;
multi line field Description;
}
language german {
field Headline caption „Überschrift“;
field Description caption „Beschreibung“;
}
Motivation
Ansatz
Beispiel
Ausblick
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
16


Fazit und Ausblick
Motivation
Ansatz
Beispiel
Ausblick
Zusammenfassung:
Evolutionsoperationen automatisieren:
• Migration bestehender Wörter
• Anpassung des Compilers / Generators
Ausblick
• Fertigstellung und Veröffentlichung des Prototyps
• Erhaltungseigenschaften: „Language Refactoring“
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
17


Fragen
Danke für die Aufmerksamkeit!
Fragen?
TECHNISCHE UNIVERSITÄT MÜNCHEN
9.10.2006
Elmar Jürgens
18

4.3.3 Diskussion

- *Probleme in der derzeitigen Praxis*
  Nach einer Veränderung des Meta-Modells fließt in Projekten derzeit mehr Aufwand für die Anpassung von Parser, Editor, ... als in die Anpassung der Generator-Templates. Deshalb sind Ansätze, welche diese Artefakte direkt aus den Meta-Modell-Beschreibungen generieren nützlich und notwendig.

- *Top-Down vs. Evolutionäre Entwicklung*
  In der Literatur wird vor der Entwicklung einer DSL oftmals eine umfangreiche Domänen-Analyse vorausgesetzt, welche u.a. die Bestimmung der Variabilitätspunkte einer Domäne zum Ziel hat. Erst anschließend sollte mit der Implementierung der Sprache und des Generators begonnen werden. Dieses Vorgehen erinnert allerdings oftmals an klassisches Top-Down Vorgehen mit all den damit verbundenen Schwächen. Im allgemeinen kann man nicht davon ausgehen, dass durch eine Domänen-Analyse die Domäne auch vollständig erfasst wurde und darauf aufbauend anschließend in Wasserfall-artiger Vorgehensweise eine Sprache samt Generator lediglich implementiert werden kann.
  Eine evolutionäre Herangehensweise trägt dem sukkzessiven Wissengewinn während des Projektverlaufs Rechnung, macht es aber unumgänglich, das Schemaevolutionsproblem (Konsistenthaltung der Worte bei Änderung der Sprache) in den Griff zu bekommen.

- *Variabilitätsmodellierung*
  Jede Form von Software-Architektur versucht bestimmte Variabilitätspunkte zu definieren. Als Beispiel hierfür könnte man Schichtenarchitekturen anführen, die eine möglichst flexible Austauschbarkeit der einzelnen Schichten gewährleisten soll. Bei der Entwicklung einer DSL sollten gerade derartige Variabilitäten in der Domäne identifiziert und über die Sprache konfigurierbar gemacht werden.

4.4 Gunther Lenz: „Software Factories, von der Theorie zur Praxis ...“

4.4.1 Abstract

In this session I will examine the practical application of Software Factories (as proposed by Jack Greenfield and Keith Short). I will discuss the current state of our implementation as well as the ins and outs of the methodology from a practitioner's perspective. Aspects that will be discussed are Software Factory Methodology, Domain Specific Languages (DSLs), and integration into the development process, product line development, automation tools, and many more.

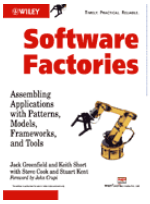
4.4.2 Folien

SIEMENS

SIEMENS CORPORATE RESEARCH

Software Factories – How we got Started

- ".Net – A Complete Development Cyle"
- 2004 Software Factories Book by Jack Greenfield and Keith Short
- First Prototype
  - Well, at least we tried ;-)



SCR©

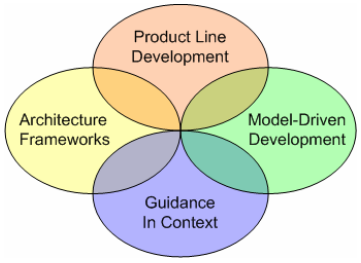
Page 2

SIEMENS

SIEMENS CORPORATE RESEARCH

Software Factories Combine Four Concepts to Address the Problems in Software Development Today

- Increasing demand of software
- One off development
  → **Product Line Development**
- Working on low level of abstraction
- Using generalized tools to develop software
  → **Model Driven Development**
- Unlimited degree of freedom of developers
- Lack of best practices
  → **Guidance in Context**
- Monolithic architectures
- Multiple non-uniform architectures
  → **Architectural Frameworks**



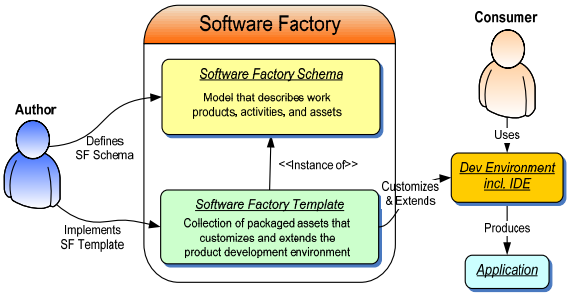
SCR©

Page 3

SIEMENS

SIEMENS CORPORATE RESEARCH

Software Factory 101 – Overview



- **Perspectives:**
  - Author
  - Consumer
- **Deliverables:**
  - Software Factory Schema
  - Software Factory Template

SCR©

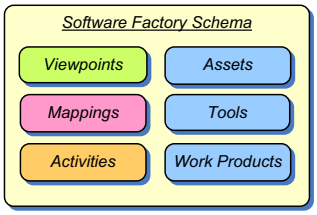
Page 4

SIEMENS

SIEMENS CORPORATE RESEARCH

Software Factory 101 – Software Factory Schema

- Uses viewpoints to describe the architecture
- Provides separation of concerns
- Incorporates the application development process
- Defines application and development core assets
  - Assets
  - Tools
  - Work products
- Describes mapping between viewpoints and artifacts
- Defines automation possibilities and tools



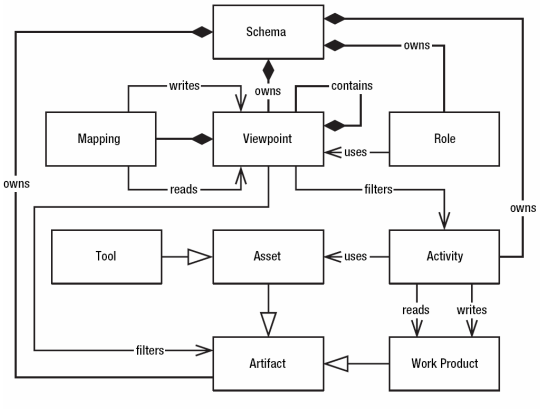
SCR©

Page 5

SIEMENS

SIEMENS CORPORATE RESEARCH

SF Schema – Simplified Meta-Model



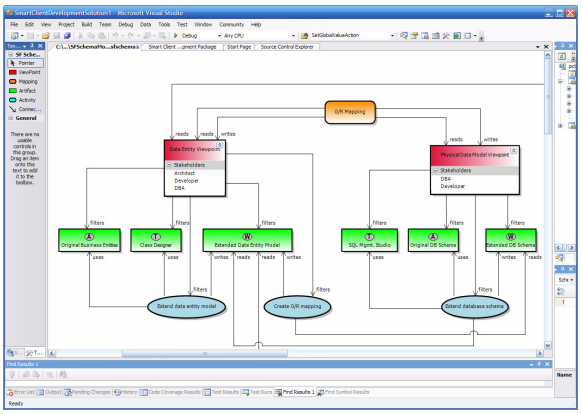
SCR©

Page 6

SIEMENS

SIEMENS CORPORATE RESEARCH

SF Schema DSL example



SCR©

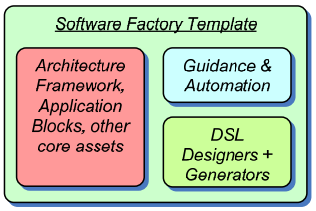
Page 7

SIEMENS

SIEMENS CORPORATE RESEARCH

Software Factory 101 – Software Factory Template

- Implements schema
- Installable package that contains the core assets
- Provides contextual guidance in form of wizards, tools, documents, guidelines, etc.
- Integrates into IDE and customizes & extends it for efficient product development
- Allows for rapidly building product line members



SCR©

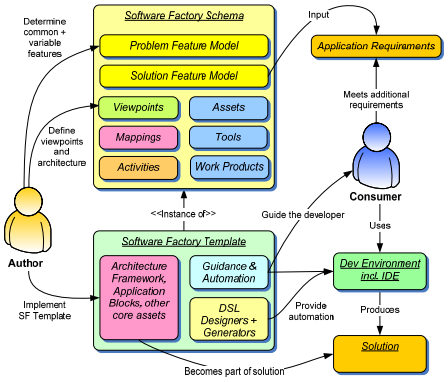
Page 8

SIEMENS

SIEMENS CORPORATE RESEARCH

Software Factory 101 – Detailed Overview

- **Author**
  - Determines common and variable features
  - Defines viewpoints and architecture
  - Implements template
- **Consumer**
  - Uses extended and customized IDE
  - Efficiently produces product line members



SCR©

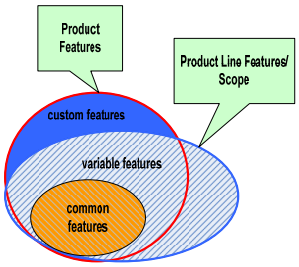
Page 9

SIEMENS

SIEMENS CORPORATE RESEARCH

Challenges

- **Product Line Development**
  - Many, fairly new concepts
  - Additional complexity
  - Commitment to product lines from the whole organization
- **Model-Driven Software Development**
  - Defining efficient Domain-Specific Languages (DSLs)
  - Building Generators
  - Keeping models in sync with evolving DSLs
- **Guidance in Context**
  - Definition of application development process
  - Automation of guidance



SCR©

Page 10

SIEMENS

SIEMENS CORPORATE RESEARCH

Challenges cont.

- **Software Factory**
  - New paradigm
  - No good examples
  - Complexity
- **Software Factory Schema**
  - Complexity
  - No tool support
  - No existing examples
  - New concept

SCR©

Page 11

SIEMENS

SIEMENS CORPORATE RESEARCH

Next Steps

- Pilot Project
  - Show benefits and drawbacks on real product
  - Quantify benefits and cost
  - Proof or disproof the following ;-)

SCR©

Page 12

SIEMENS

SIEMENS CORPORATE RESEARCH

Benefits of using Software Factories

- **Product Lines and Architectural Frameworks**
  - Increased reuse – possibly up to 70% or more
  - Higher quality
  - Faster time to market
  - Uniform architecture and user experience
  - One code base to maintain
  - Defined extensibility points
  - Reuse not only of code and architecture but also requirements etc.
  - Highly automated product line development
  - Incorporate best practices
- **Model-Driven Development**
  - Highly efficient for narrow domains
  - Use concepts from domain, therefore intuitive and precise
  - Model validation
  - Nokia case study: 5 – 10 times more productive using Domain Specific Languages
- **Guidance in Context**
  - Automate error prone and menial tasks
  - Avoid common errors
  - Incorporate best practices
  - Reduce learning curve

SCR©

Page 13

SIEMENS

SIEMENS CORPORATE RESEARCH

The End... For Now

Questions?

SCR®

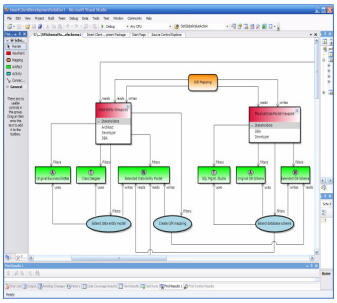
Page 14

SIEMENS

SIEMENS CORPORATE RESEARCH

We Offer:

- Definition of iterative process that fits your project
- Consulting to define a Software Factory schema
  - Based on existing assets
  - Incorporate the development process
  - Define core assets, dependencies and identify automation opportunities
- Support to implement the Software Factory template
  - Guidance Automation Toolkit
  - Domain Specific Languages
  - Code and artifact Generators
- Mentoring through parts or entire process
- Tutorials and classes on Software Factories



SCR®

Page 15

4.4.3 Diskussion

- *Planung einer Produktlinie anhand existierender Produkte*
  Ein sinnvolles Vorgehen zur Entwicklung einer Produktlinie ist das Heranziehen bereits existierender Produkte in der Domäne. Dabei würde die DSL genau die Variabilität/die Unterschiede zwischen den Produkten beschreiben. Jede weitere Entwicklung in dieser Domäne kann nun mithilfe der DSL begonnen werden und Variabilitäten, die sich aus speziellen neuen Problemstellung ergeben, führen zu einer sukzessiven Erweiterung der Ausdrucksmöglichkeiten der DSL.

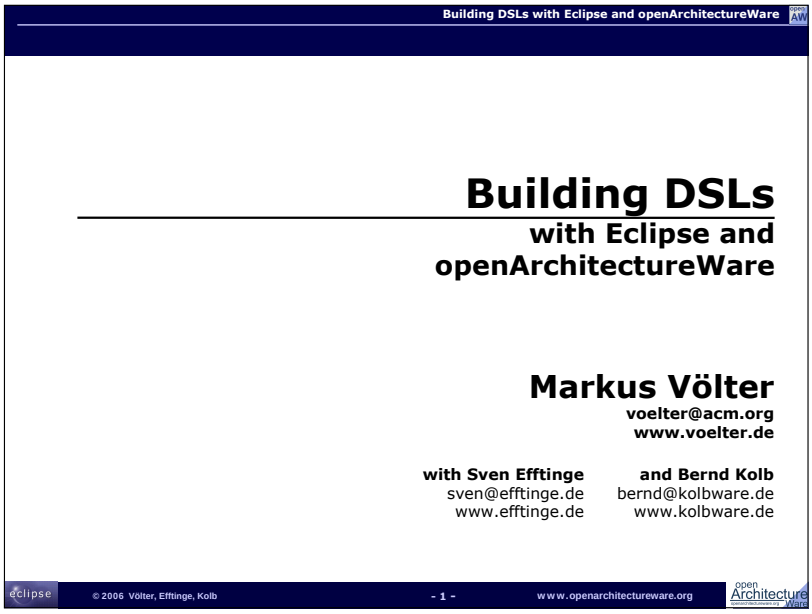
- *DSLs als Werkzeug im Einzelprojekt vs. DSLs zur Produktlinienentwicklung*
  DSLs werden oftmals als Mittel zur Realisierung einer Software-Produktlinie angesehen. Wohingegen Model-Driven-Architecture Ansätze ähnliche (meist auf UML-Spezialisierung durch UML-Profile) Techniken auch für Projekte in Anspruch nehmen, die lediglich die Entwicklung eines einzelnen Software-Systems zum Ziel haben. Hierbei dienen DSLs lediglich als Mittel zur Erarbeitung und Formalisierung des Domänenwissens sowie zur Unterstützung der Kommunikation mit Fachexperten. DSLs sind in dieser Verwendungsform ein Werkzeug während des Entwicklungsprozesses und ihre Definition ein Zwischenprodukt auf dem Weg zu einem Endprodukt. Die DSL soll hierbei die Änderbarkeit des zu entwicklenden Softwaresystems erleichtern. Auch eine Vereinfachung der Wartung und Anpassung während der Betriebsphase eines Software-Systems ist einer der erhofften Vorteile. Problematisch ist hierbei, dass sich weder zukünftige Wartungsfälle noch Änderungen in den Anforderungen während der Entwicklung exakt antizipieren lassen und man somit nicht zwingend davon ausgehen kann, dass der künftige Änderungsbedarf auch durch den Sprachschatz der DSL formulierbar ist. Zudem könnten Projekte in denen DSL-Technologien in dieser Form eingesetzt werden, an einer Tendenz zum Over-Engineering leiden und dadurch zusätzliche Kosten entstehen.

4.5 Markus Völter: „Building DSLs with Eclipse and openArchitectureWare“

4.5.1 Abstract

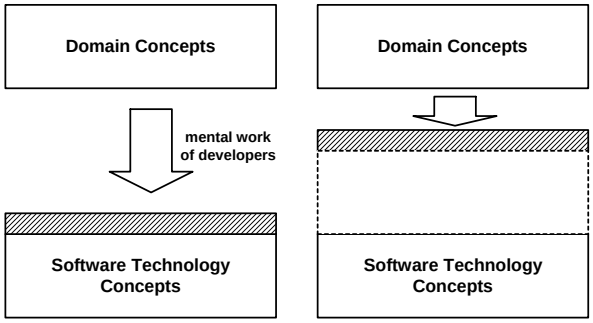
Bei der Erstellung (externer) DSLs gilt es eine ganze Reihe von Herausforderungen zu meistern. Primär stellt sich die Frage, wie man zu einem - grafischen oder textuellen - Editor für die konkrete Syntax der DSL kommt, um die konkreten Modelle zu erfassen. Damit ist es allerdings nicht getan.
Die Modelle müssen verifiziert und transformiert werden, manchmal in andere Modelle, meist aber in ausführbaren Code. In diesem Vortrag möchte ich kurz zeigen, wie sich all diese Dinge mit Open Source Werkzeugen, konkret: Eclipse und openArchitectureWare, effizient und elegant lösen lassen.

4.5.2 Folien

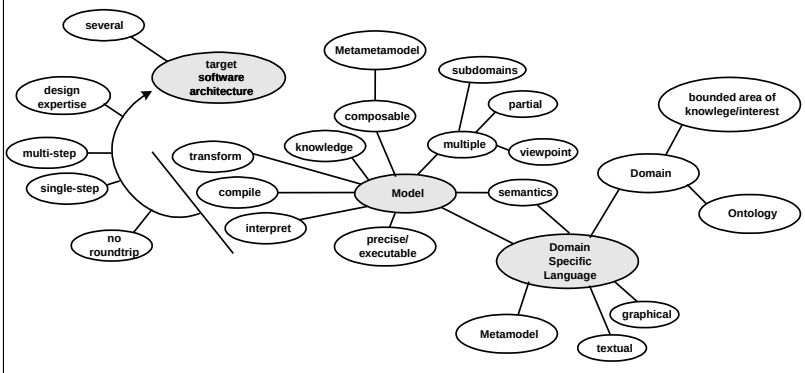


Model Driven Development

- Model Driven Development is about making software development more **domain-related** as opposed to **computing related**. It is also about making software development in a certain domain **more efficient**.

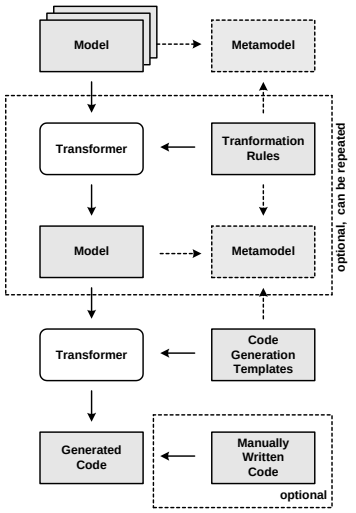


MDSD Core Concepts



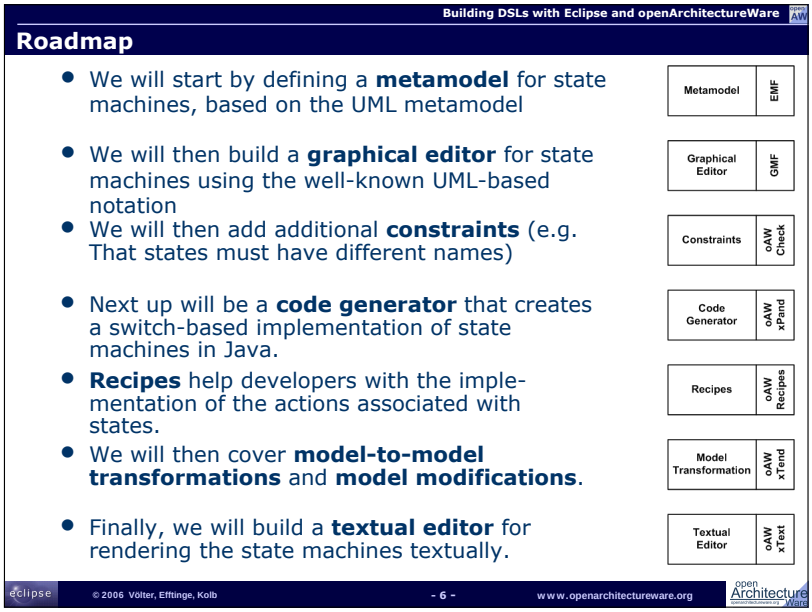
How does MDSD work?

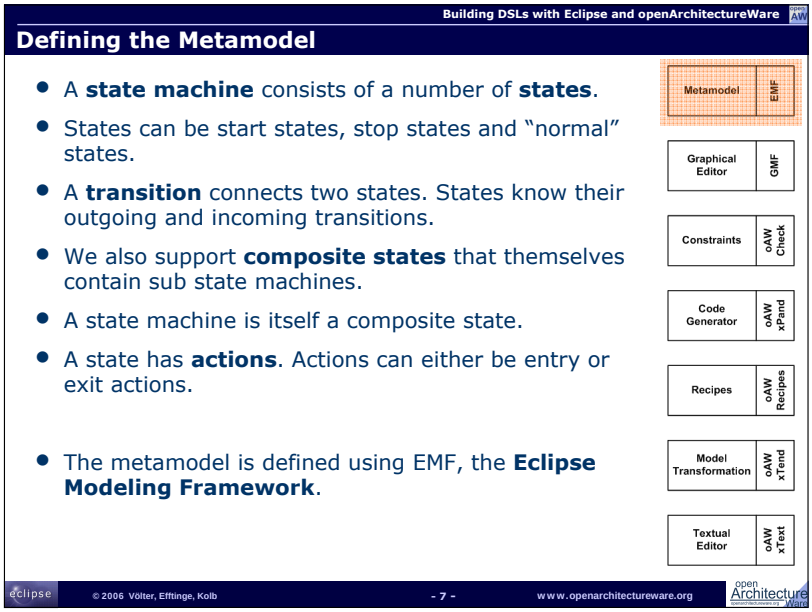
- Developer develops **model(s)** based on certain metamodel(s).
- Using **code generation templates**, the model is transformed to executable code.
- Optionally, the **generated code is merged** with manually written code.
- One or more **model-to-model transformation steps** may precede code generation.

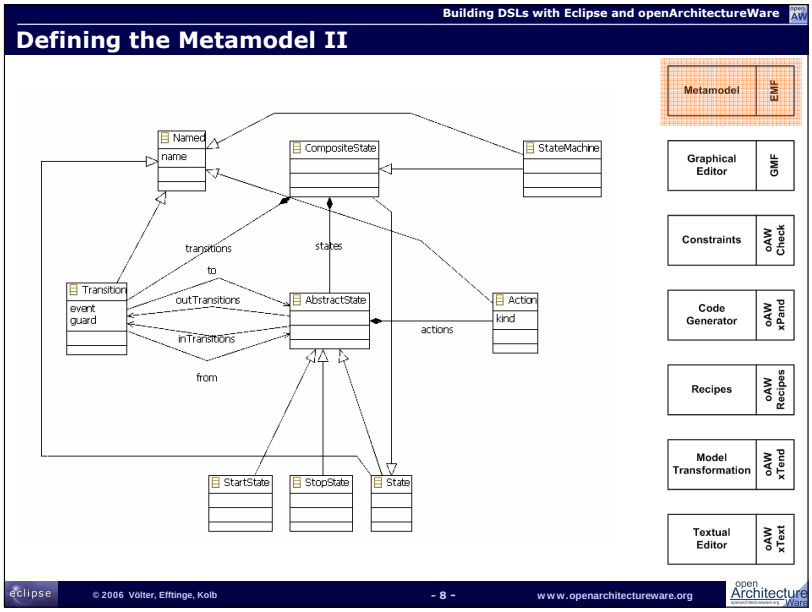


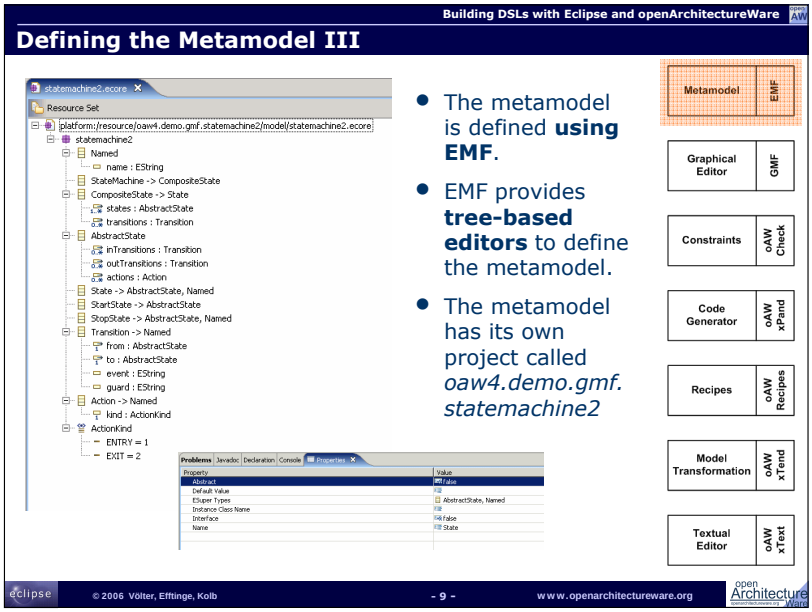
Goals & Challenges

- **Goals:**
  - We need an **end-to-end tool chain** that allows us to build models, verify them and generate various artifacts from them.
  - All of this should happen in a homogeneous environment, namely Eclipse.
- **Challenges:**
  - **Good Editors** for your models
  - **Verifying** the models as you build them
  - **Transforming/Modifying** models
  - **Generating** Code
  - **Integrating** generated and non-generated code

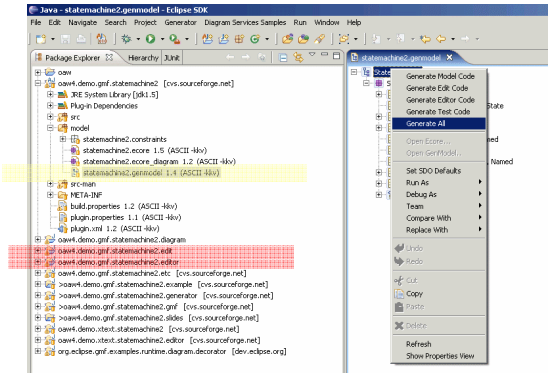

Building DSLs with Eclipse and openArchitectureWare
Roadmap
We will start by defining a metamodel for state machines, based on the UML metamodel
We will then build a graphical editor for state machines using the well-known UML-based notation
We will then add additional constraints (e.g. That states must have different names)
Next up will be a code generator that creates a switch-based implementation of state machines in Java.
Recipes help developers with the imple-mentation of the actions associated with states.
We will then cover model-to-model transformations and model modifications.
Finally, we will build a textual editor for rendering the state machines textually.
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 6 -
www.openarchitectureware.org
open ArchitectureWare


Building DSLs with Eclipse and openArchitectureWare
Defining the Metamodel
A state machine consists of a number of states.
States can be start states, stop states and "normal" states.
A transition connects two states. States know their outgoing and incoming transitions.
We also support composite states that themselves contain sub state machines.
A state machine is itself a composite state.
A state has actions. Actions can either be entry or exit actions.
The metamodel is defined using EMF, the Eclipse Modeling Framework.
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 7 -
www.openarchitectureware.org
open ArchitectureWare


Building DSLs with Eclipse and openArchitectureWare
Defining the Metamodel II
Named
name
CompositeState
StateMachine
transitions
states
to
Transition
event
guard
outTransitions
AbstractState
Action
kind
actions
inTransitions
from
StartState
StopState
State
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 8 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Defining the Metamodel III
statemachine2.ecore
Resource Set
platform:/resource/oaw4.demo.gmf.statemachine2/model/statemachine2.ecore
statemachine2
Named
name : EString
StateMachine -> CompositeState
CompositeState -> State
states : AbstractState
transitions : Transition
AbstractState
inTransitions : Transition
outTransitions : Transition
actions : Action
State -> AbstractState, Named
StartState -> AbstractState
StopState -> AbstractState, Named
Transition -> Named
from : AbstractState
to : AbstractState
event : EString
guard : EString
Action -> Named
kind : ActionKind
ActionKind
ENTRY = 1
EXIT = 2
The metamodel is defined using EMF.
EMF provides tree-based editors to define the metamodel.
The metamodel has its own project called oaw4.demo.gmf.statemachine2
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 9 -
www.openarchitectureware.org
open Architecture Ware

Defining the Metamodel IV



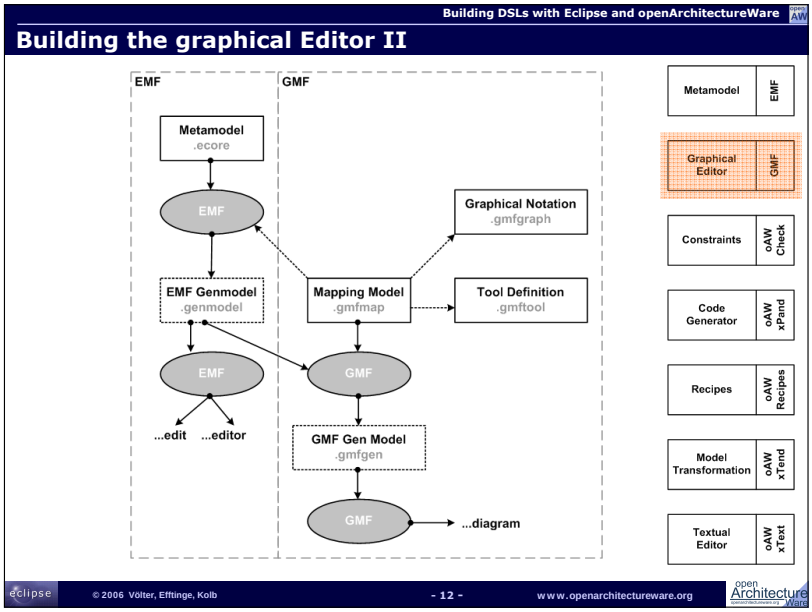
- Note that we have to create the **genmodel** as well as the **.edit** and **.editor** projects from the ecore model.
- This is necessary for the graphical editor to work.

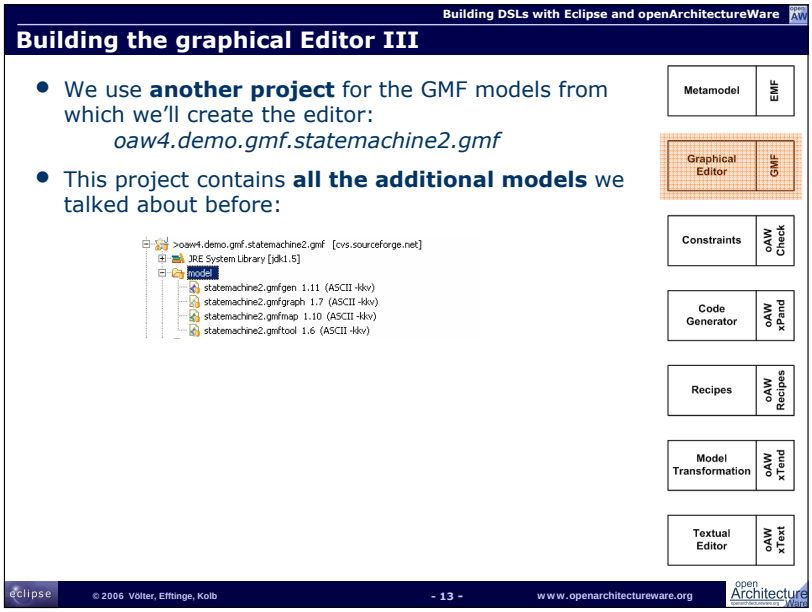
Metamodel | EMF
Graphical Editor | GMF
Constraints | oAW Check
Code Generator | oAW xPand
Recipes | oAW Recipes
Model Transformation | oAW xTend
Textual Editor | oAW xText

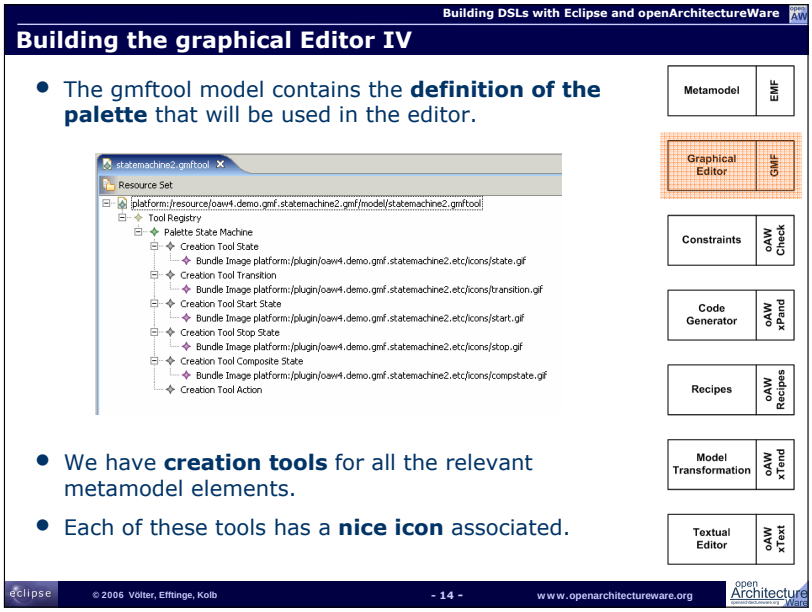
Building the graphical Editor

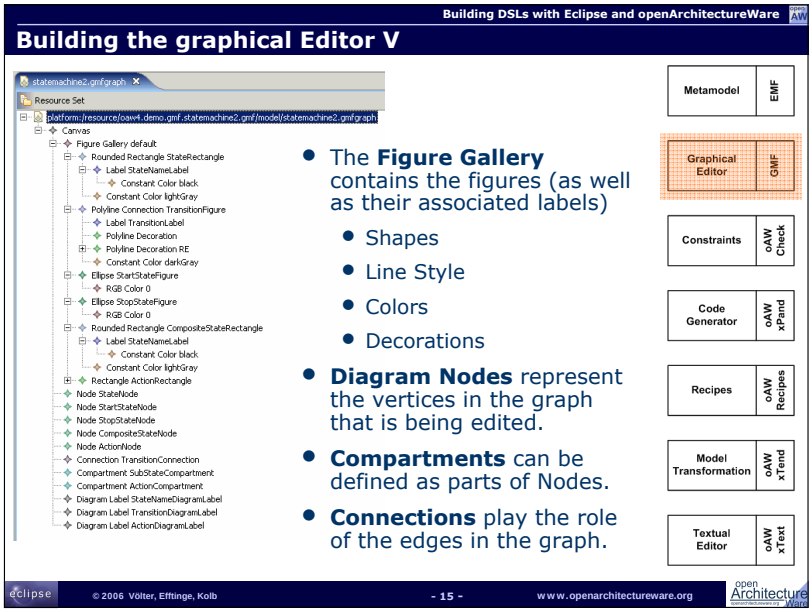
- The editor is **based on the metamodel** defined before.
- A number of additional models has to be defined:
  - A model defining the **graphical notation**
  - A model for the editor's **palette** and other tooling
  - A **mapping model** that binds these two models to the domain metamodel
- A **generator** generates the concrete editor based on these models.
- The editor is built with the Eclipse GMF, the **Graphical Modelling Framework**.

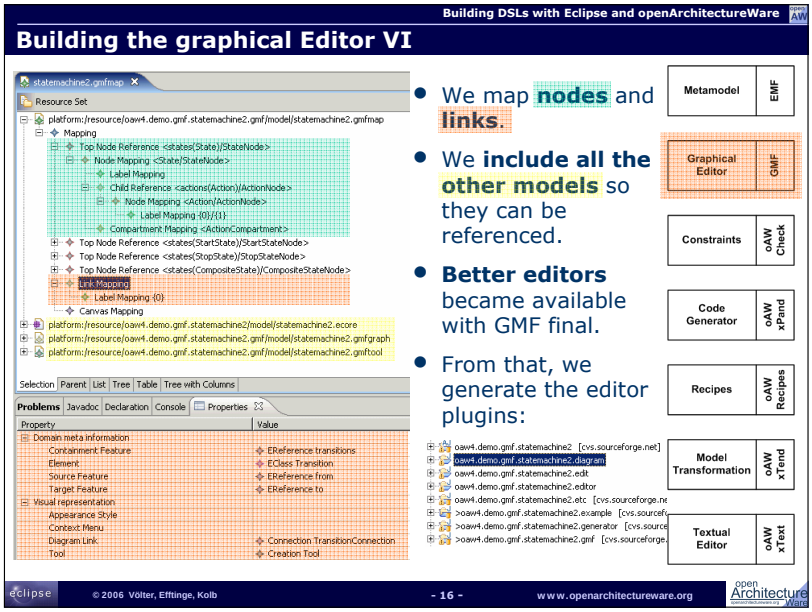
Metamodel | EMF
Graphical Editor | GMF
Constraints | oAW Check
Code Generator | oAW xPand
Recipes | oAW Recipes
Model Transformation | oAW xTend
Textual Editor | oAW xText

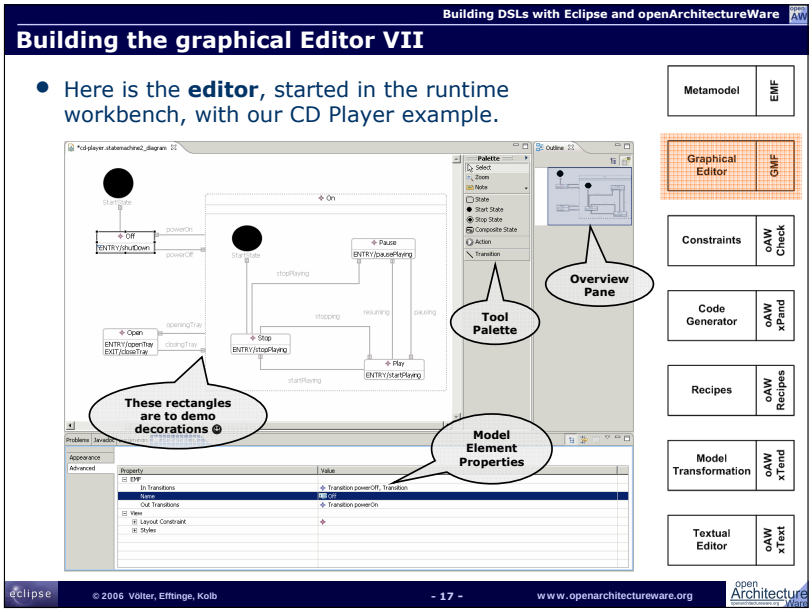

Building DSLs with Eclipse and openArchitectureWare
Building the graphical Editor II
EMF
GMF
Metamodel
.ecore
EMF
Graphical Notation
.gmfgraph
EMF Genmodel
.genmodel
Mapping Model
.gmfmap
Tool Definition
.gmftool
EMF
GMF
...edit
...editor
GMF Gen Model
.gmfgen
GMF
...diagram
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 12 -
www.openarchitectureware.org
open Architecture Ware

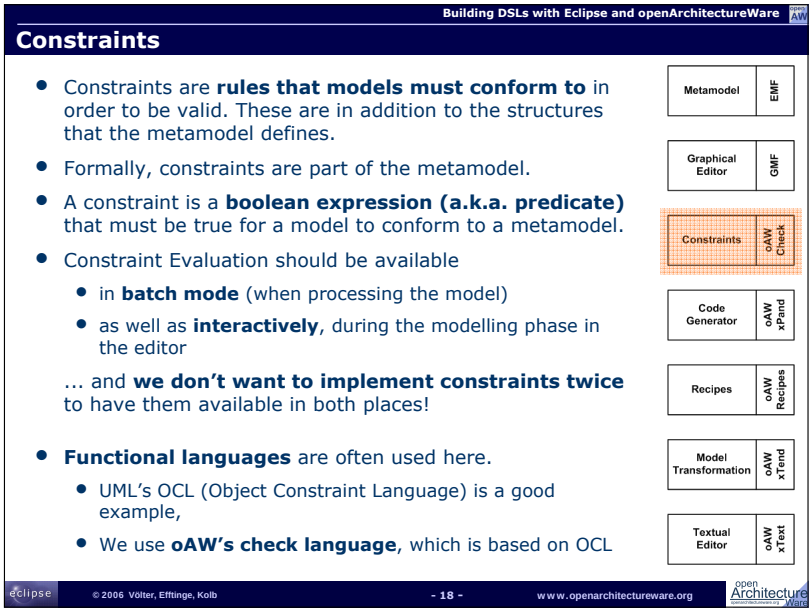

Building DSLs with Eclipse and openArchitectureWare
Building the graphical Editor III
We use another project for the GMF models from which we'll create the editor:
oaw4.demo.gmf.statemachine2.gmf
This project contains all the additional models we talked about before:
>oaw4.demo.gmf.statemachine2.gmf [cvs.sourceforge.net]
JRE System Library [jdk1.5]
model
statemachine2.gmfgen 1.11 (ASCII -kkv)
statemachine2.gmfgraph 1.7 (ASCII -kkv)
statemachine2.gmfmap 1.10 (ASCII -kkv)
statemachine2.gmftool 1.6 (ASCII -kkv)
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 13 -
www.openarchitectureware.org
open Architecture Ware

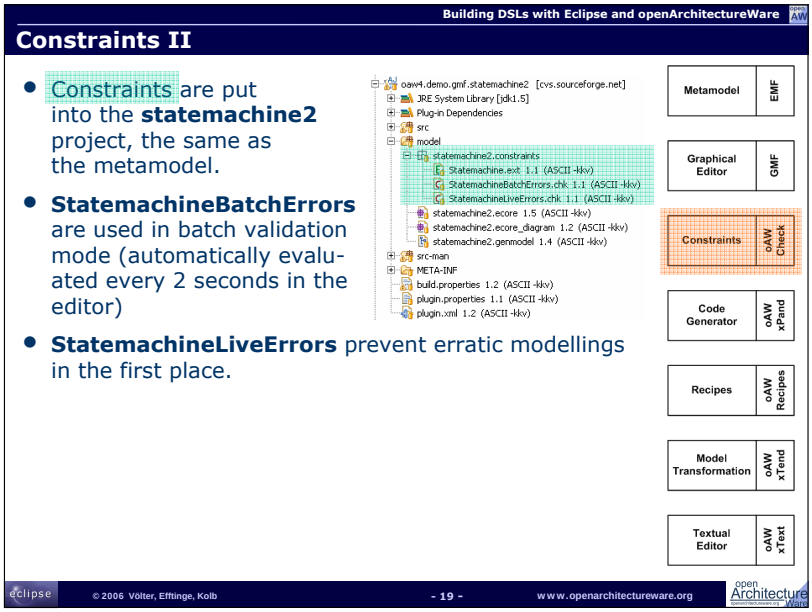

Building DSLs with Eclipse and openArchitectureWare
Building the graphical Editor IV
The gmftool model contains the definition of the palette that will be used in the editor.
statemachine2.gmftool
Resource Set
platform:/resource/oaw4.demo.gmf.statemachine2.gmf/model/statemachine2.gmftool
Tool Registry
Palette State Machine
Creation Tool State
Bundle Image platform:/plugin/oaw4.demo.gmf.statemachine2.etc/icons/state.gif
Creation Tool Transition
Bundle Image platform:/plugin/oaw4.demo.gmf.statemachine2.etc/icons/transition.gif
Creation Tool Start State
Bundle Image platform:/plugin/oaw4.demo.gmf.statemachine2.etc/icons/start.gif
Creation Tool Stop State
Bundle Image platform:/plugin/oaw4.demo.gmf.statemachine2.etc/icons/stop.gif
Creation Tool Composite State
Bundle Image platform:/plugin/oaw4.demo.gmf.statemachine2.etc/icons/compstate.gif
Creation Tool Action
We have creation tools for all the relevant metamodel elements.
Each of these tools has a nice icon associated.
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 14 -
www.openarchitectureware.org
open Architecture Ware

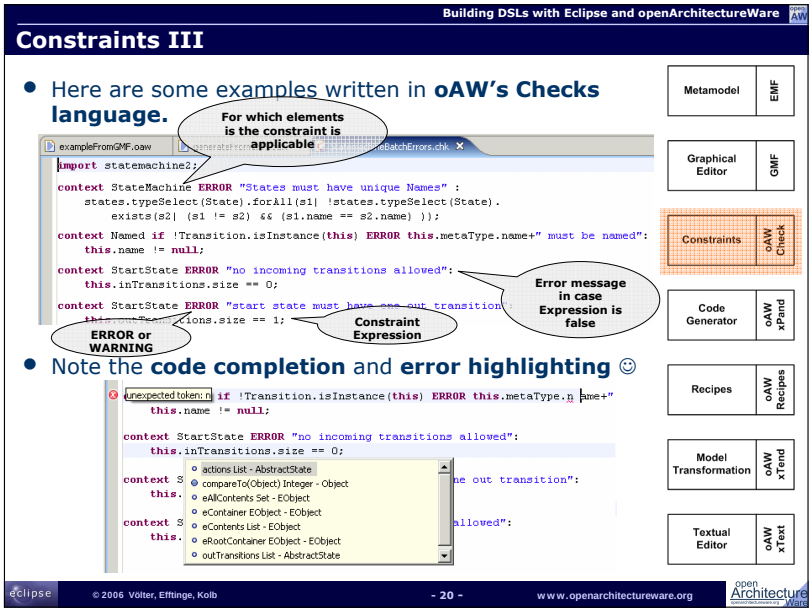

Building DSLs with Eclipse and openArchitectureWare
Building the graphical Editor V
statemachine2.gmfgraph
Resource Set
platform:/resource/oaw4.demo.gmf.statemachine2.gmf/model/statemachine2.gmfgraph
Canvas
Figure Gallery default
Rounded Rectangle StateRectangle
Label StateNameLabel
Constant Color black
Constant Color lightGray
Polyline Connection TransitionFigure
Label TransitionLabel
Polyline Decoration
Polyline Decoration RE
Constant Color darkGray
Ellipse StartStateFigure
RGB Color 0
Ellipse StopStateFigure
RGB Color 0
Rounded Rectangle CompositeStateRectangle
Label StateNameLabel
Constant Color black
Constant Color lightGray
Rectangle ActionRectangle
Node StateNode
Node StartStateNode
Node StopStateNode
Node CompositeStateNode
Node ActionNode
Connection TransitionConnection
Compartment SubStateCompartment
Compartment ActionCompartment
Diagram Label StateNameDiagramLabel
Diagram Label TransitionDiagramLabel
Diagram Label ActionDiagramLabel
The Figure Gallery contains the figures (as well as their associated labels)
Shapes
Line Style
Colors
Decorations
Diagram Nodes represent the vertices in the graph that is being edited.
Compartments can be defined as parts of Nodes.
Connections play the role of the edges in the graph.
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 15 -
www.openarchitectureware.org
open Architecture Ware

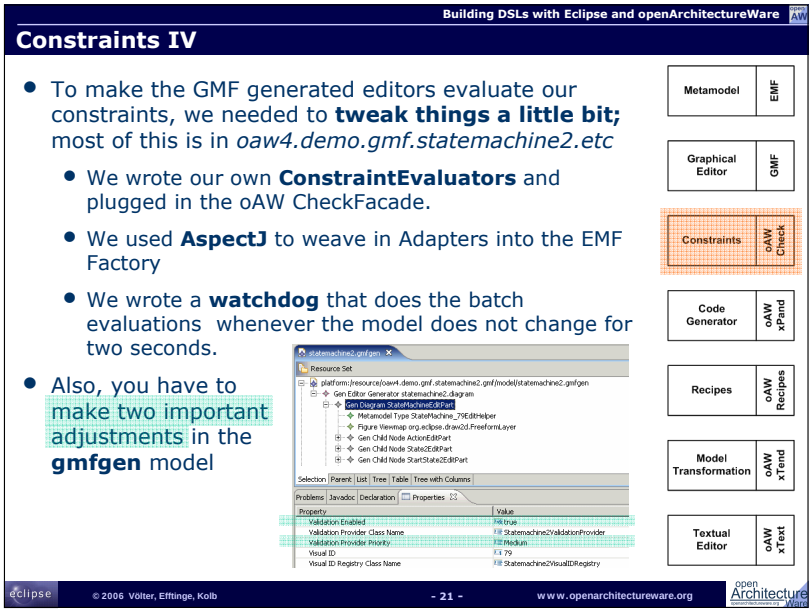

Building DSLs with Eclipse and openArchitectureWare
Building the graphical Editor VI
We map nodes and links.
We include all the other models so they can be referenced.
Better editors became available with GMF final.
From that, we generate the editor plugins:
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 16 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Building the graphical Editor VII
Here is the editor, started in the runtime workbench, with our CD Player example.
Overview Pane
Tool Palette
These rectangles are to demo decorations
Model Element Properties
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 17 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Constraints
Constraints are rules that models must conform to in order to be valid. These are in addition to the structures that the metamodel defines.
Formally, constraints are part of the metamodel.
A constraint is a boolean expression (a.k.a. predicate) that must be true for a model to conform to a metamodel.
Constraint Evaluation should be available
in batch mode (when processing the model)
as well as interactively, during the modelling phase in the editor
... and we don't want to implement constraints twice to have them available in both places!
Functional languages are often used here.
UML's OCL (Object Constraint Language) is a good example,
We use oAW's check language, which is based on OCL
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 18 -
www.openarchitectureware.org
open Architecture Ware

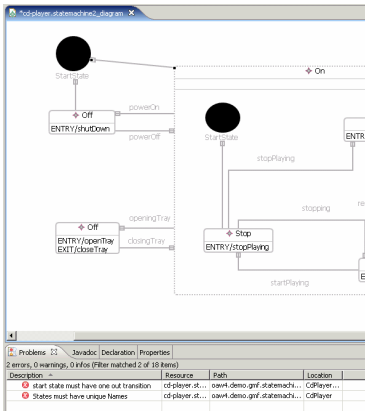

Building DSLs with Eclipse and openArchitectureWare
Constraints II
Constraints are put into the statemachine2 project, the same as the metamodel.
StatemachineBatchErrors are used in batch validation mode (automatically evaluated every 2 seconds in the editor)
StatemachineLiveErrors prevent erratic modellings in the first place.
oaw4.demo.gmf.statemachine2 [cvs.sourceforge.net]
JRE System Library [jdk1.5]
Plug-in Dependencies
src
model
statemachine2.constraints
Statemachine.ext 1.1 (ASCII -kkv)
StatemachineBatchErrors.chk 1.1 (ASCII -kkv)
StatemachineLiveErrors.chk 1.1 (ASCII -kkv)
statemachine2.ecore 1.5 (ASCII -kkv)
statemachine2.ecore_diagram 1.2 (ASCII -kkv)
statemachine2.genmodel 1.4 (ASCII -kkv)
src-man
META-INF
build.properties 1.2 (ASCII -kkv)
plugin.properties 1.1 (ASCII -kkv)
plugin.xml 1.2 (ASCII -kkv)
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 19 -
www.openarchitectureware.org
open Architecture Ware

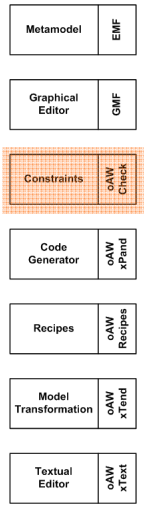

Building DSLs with Eclipse and openArchitectureWare
Constraints III
Here are some examples written in oAW's Checks language.
For which elements is the constraint is applicable
ERROR or WARNING
Constraint Expression
Error message in case Expression is false
Note the code completion and error highlighting ☺
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 20 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Constraints IV
To make the GMF generated editors evaluate our constraints, we needed to tweak things a little bit; most of this is in oaw4.demo.gmf.statemachine2.etc
We wrote our own ConstraintEvaluators and plugged in the oAW CheckFacade.
We used AspectJ to weave in Adapters into the EMF Factory
We wrote a watchdog that does the batch evaluations whenever the model does not change for two seconds.
Also, you have to make two important adjustments in the gmfgen model
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 21 -
www.openarchitectureware.org
open Architecture Ware

Constraints V

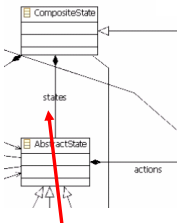
- In this model there are **two errors**
  - There are two states with the same name (Off)
  - The start state has more than one out-Transition
- The validation is executed automatically
- Clicking the error message **selects** the respective "broken" **model element** in the diagram.



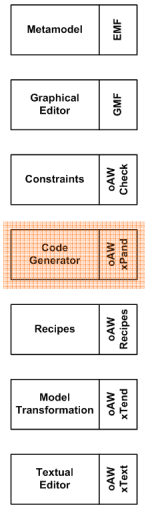


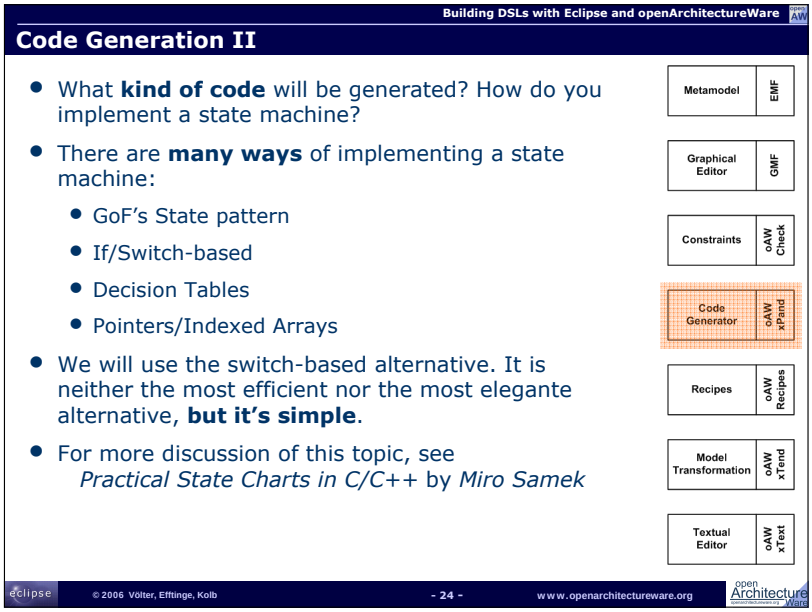
Code Generation

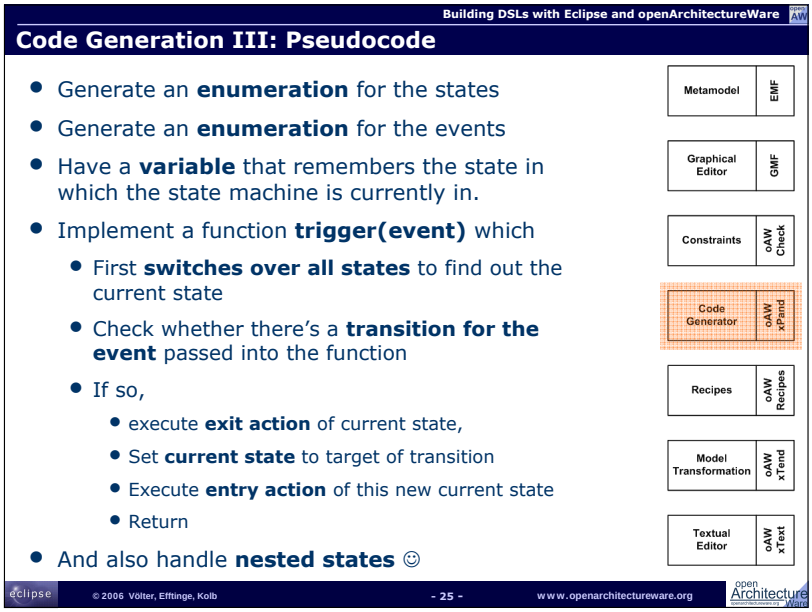
- Code Generation is used to **generate executable code** from models.
- Code Generation is **based on the metamodel** and uses **templates** to attach to-be-generated source code.
- In openArchitectureWare, we use a **template language** called **xPand**.
- It provides a number of **advanced features** such as polymorphism, AO support and a powerful integrated expression language.
- Templates can access **metamodel properties** seamlessly

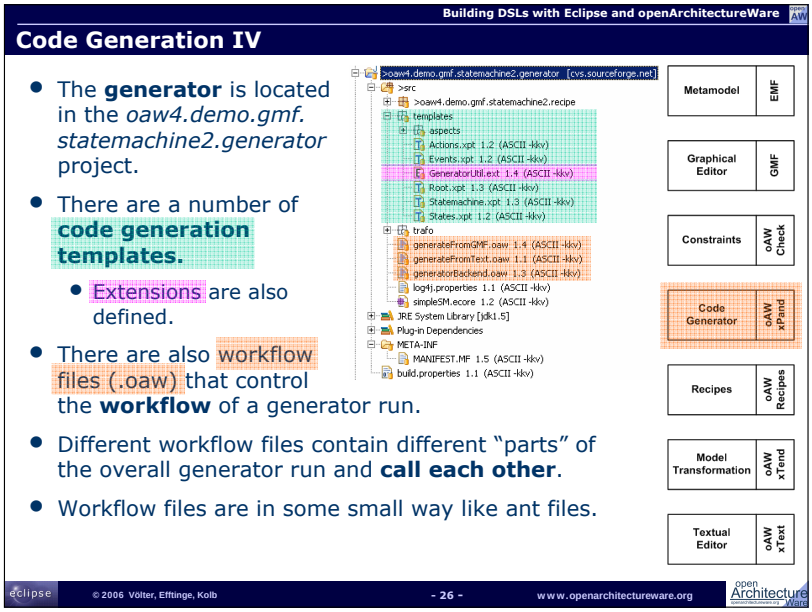


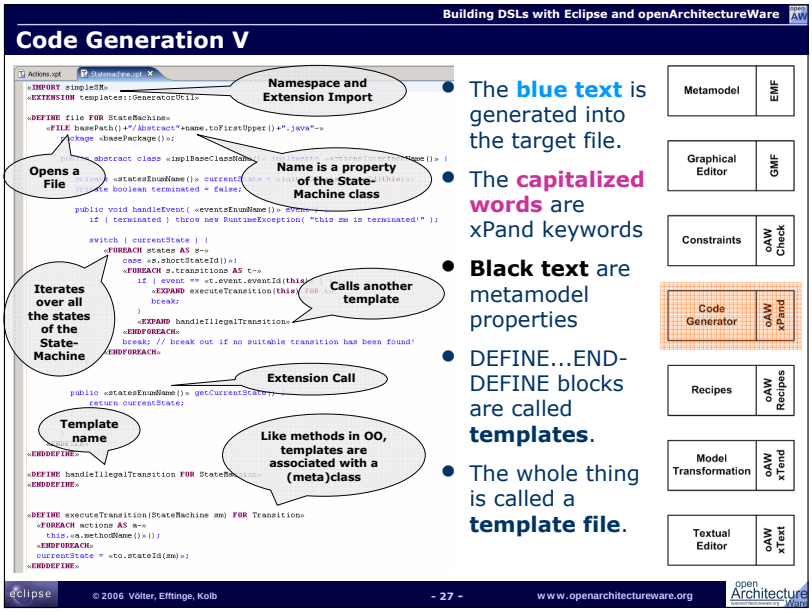

```
«DEFINE SwitchBasedImpl FOR StateMachine»
    «FOREACH states.typeSelect(State) AS s
        public static final int «s.constant
    «ENDFOREACH»
```

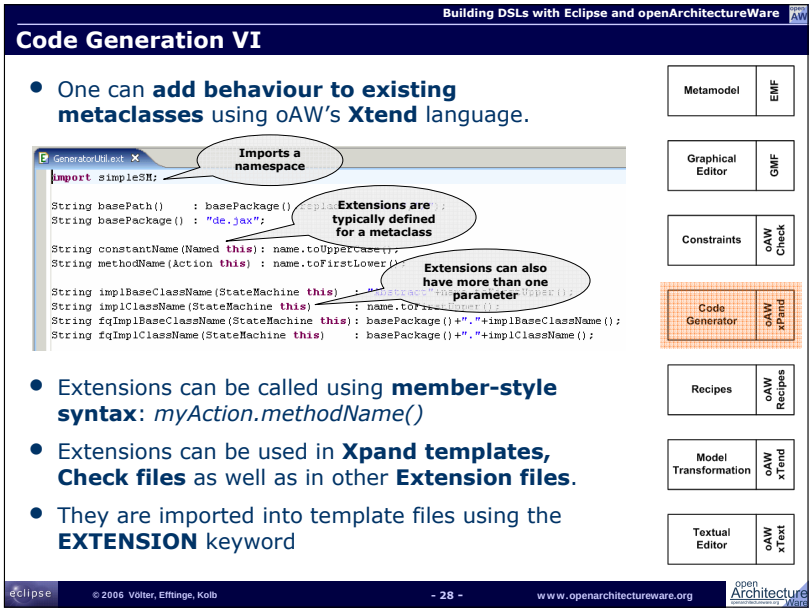


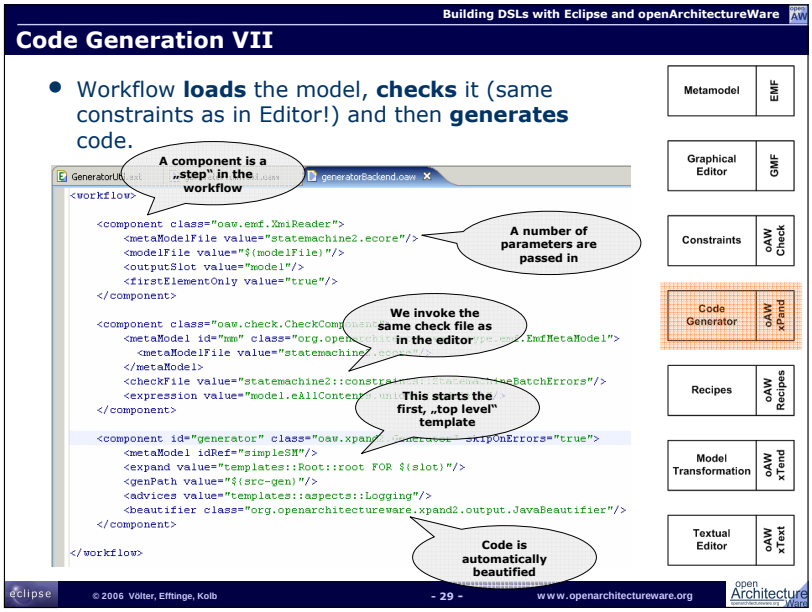

Building DSLs with Eclipse and openArchitectureWare
Code Generation II
What kind of code will be generated? How do you implement a state machine?
There are many ways of implementing a state machine:
GoF's State pattern
If/Switch-based
Decision Tables
Pointers/Indexed Arrays
We will use the switch-based alternative. It is neither the most efficient nor the most elegante alternative, but it's simple.
For more discussion of this topic, see Practical State Charts in C/C++ by Miro Samek
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 24 -
www.openarchitectureware.org
open Architecture Ware

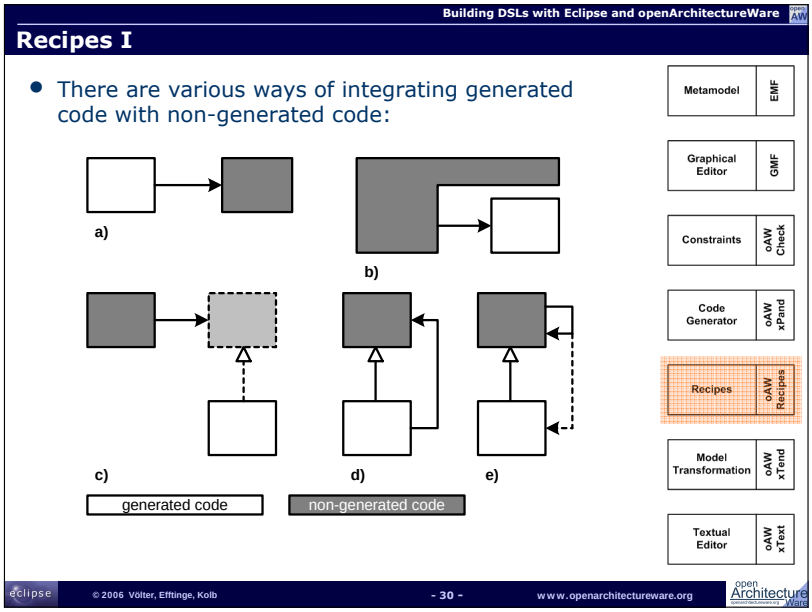

Building DSLs with Eclipse and openArchitectureWare
Code Generation III: Pseudocode
Generate an enumeration for the states
Generate an enumeration for the events
Have a variable that remembers the state in which the state machine is currently in.
Implement a function trigger(event) which
First switches over all states to find out the current state
Check whether there's a transition for the event passed into the function
If so,
execute exit action of current state,
Set current state to target of transition
Execute entry action of this new current state
Return
And also handle nested states ☺
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 25 -
www.openarchitectureware.org
open Architecture Ware

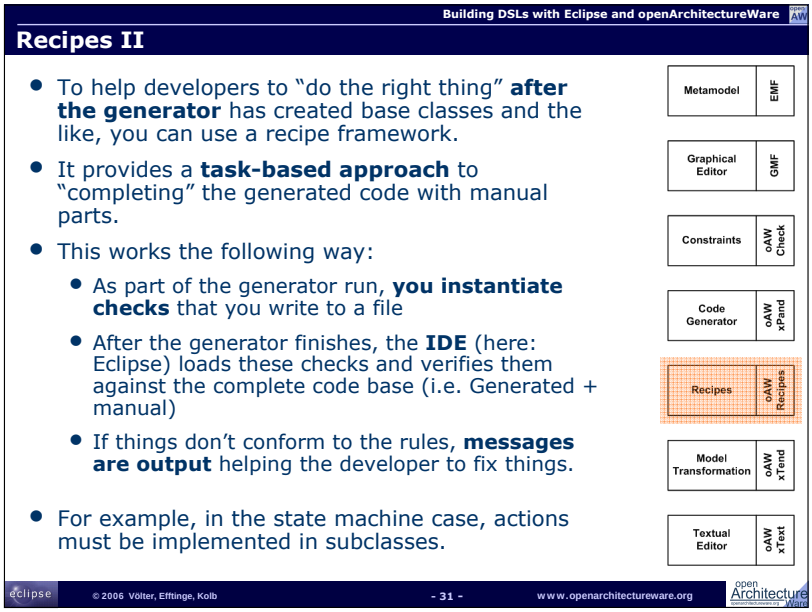

Building DSLs with Eclipse and openArchitectureWare
Code Generation IV
The generator is located in the oaw4.demo.gmf.statemachine2.generator project.
There are a number of code generation templates.
Extensions are also defined.
There are also workflow files (.oaw) that control the workflow of a generator run.
Different workflow files contain different "parts" of the overall generator run and call each other.
Workflow files are in some small way like ant files.
>oaw4.demo.gmf.statemachine2.generator [cvs.sourceforge.net]
>src
>oaw4.demo.gmf.statemachine2.recipe
templates
aspects
Actions.xpt 1.2 (ASCII -kkv)
Events.xpt 1.2 (ASCII -kkv)
GeneratorUtil.ext 1.4 (ASCII -kkv)
Root.xpt 1.3 (ASCII -kkv)
Statemachine.xpt 1.3 (ASCII -kkv)
States.xpt 1.2 (ASCII -kkv)
trafo
generateFromGMF.oaw 1.4 (ASCII -kkv)
generateFromText.oaw 1.1 (ASCII -kkv)
generatorBackend.oaw 1.3 (ASCII -kkv)
log4j.properties 1.1 (ASCII -kkv)
simpleSM.ecore 1.2 (ASCII -kkv)
JRE System Library [jdk1.5]
Plug-in Dependencies
META-INF
MANIFEST.MF 1.5 (ASCII -kkv)
build.properties 1.1 (ASCII -kkv)
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 26 -
www.openarchitectureware.org
openArchitectureWare

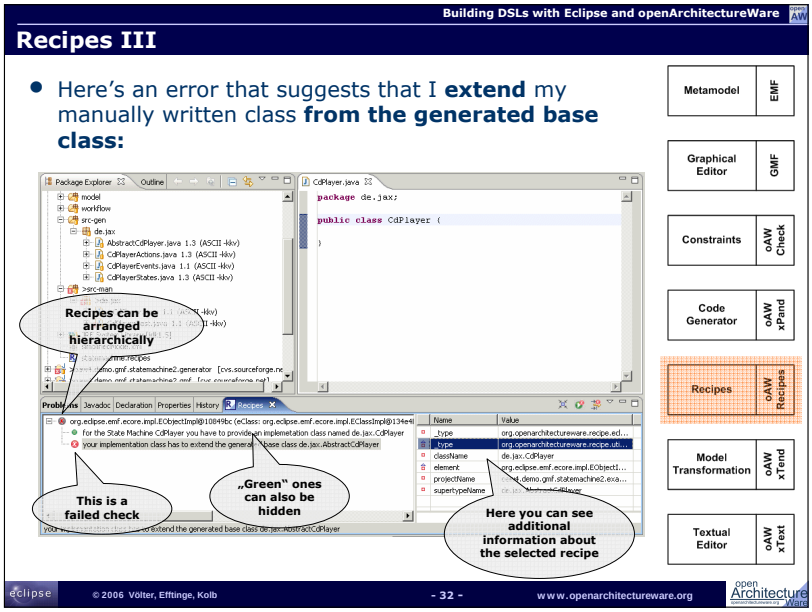

Building DSLs with Eclipse and openArchitectureWare
Code Generation V
Namespace and Extension Import
Opens a File
Name is a property of the State-Machine class
Iterates over all the states of the State-Machine
Calls another template
Extension Call
Template name
Like methods in OO, templates are associated with a (meta)class
The blue text is generated into the target file.
The capitalized words are xPand keywords
Black text are metamodel properties
DEFINE...END-DEFINE blocks are called templates.
The whole thing is called a template file.
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 27 -
www.openarchitectureware.org
openArchitectureWare

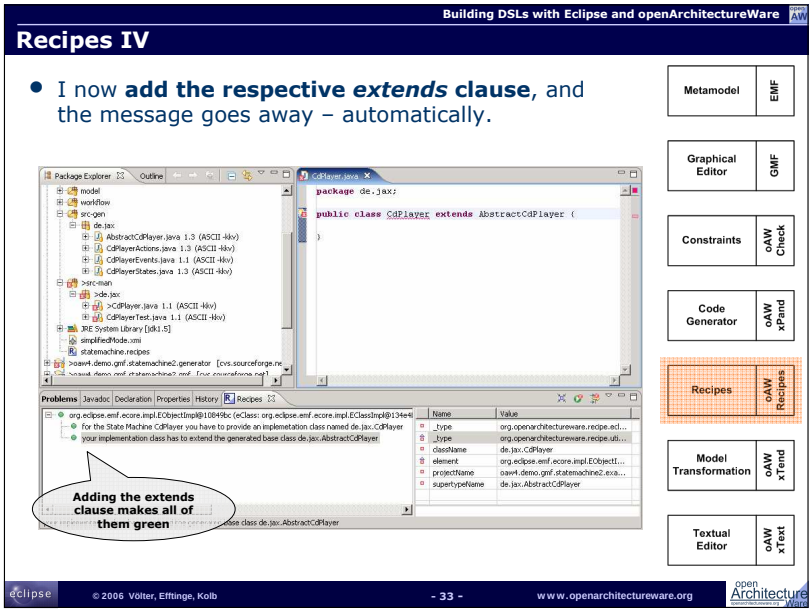

Building DSLs with Eclipse and openArchitectureWare
Code Generation VI
One can add behaviour to existing metaclasses using oAW's Xtend language.
GeneratorUtil.ext
Imports a namespace
Extensions are typically defined for a metaclass
Extensions can also have more than one parameter
import simpleSM;
String basePath() : basePackage().replaceAll("\\.","/");
String basePackage() : "de.jax";
String constantName(Named this): name.toUpperCase();
String methodName(Action this) : name.toFirstLower();
String implBaseClassName(StateMachine this) : "Abstract"+name.toFirstUpper();
String implClassName(StateMachine this) : name.toFirstUpper();
String fqImplBaseClassName(StateMachine this): basePackage()+"."+implBaseClassName();
String fqImplClassName(StateMachine this) : basePackage()+"."+implClassName();
Extensions can be called using member-style syntax: myAction.methodName()
Extensions can be used in Xpand templates, Check files as well as in other Extension files.
They are imported into template files using the EXTENSION keyword
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 28 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Code Generation VII
Workflow loads the model, checks it (same constraints as in Editor!) and then generates code.
generatorBackend.oaw
A component is a „step" in the workflow
A number of parameters are passed in
We invoke the same check file as in the editor
This starts the first, „top level" template
Code is automatically beautified
<workflow>
<component class="oaw.emf.XmiReader">
<metaModelFile value="statemachine2.ecore"/>
<modelFile value="${modelFile}"/>
<outputSlot value="model"/>
<firstElementOnly value="true"/>
</component>
<component class="oaw.check.CheckComponent">
<metaModel id="mm" class="org.openarchitectureware.type.emf.EmfMetaModel">
<metaModelFile value="statemachine2.ecore"/>
</metaModel>
<checkFile value="statemachine2::constraints::CheckBatchErrors"/>
<expression value="model.eAllContents.uniq()"/>
</component>
<component id="generator" class="oaw.xpand2.Generator" skipOnErrors="true">
<metaModel idRef="simpleSM"/>
<expand value="templates::Root::root FOR ${slot}"/>
<genPath value="${src-gen}"/>
<advices value="templates::aspects::Logging"/>
<beautifier class="org.openarchitectureware.xpand2.output.JavaBeautifier"/>
</component>
</workflow>
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 29 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Recipes I
There are various ways of integrating generated code with non-generated code:
a)
b)
c)
d)
e)
generated code
non-generated code
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 30 -
www.openarchitectureware.org
open Architecture Ware

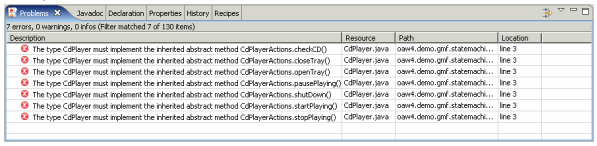

Building DSLs with Eclipse and openArchitectureWare
Recipes II
To help developers to “do the right thing” after the generator has created base classes and the like, you can use a recipe framework.
It provides a task-based approach to “completing” the generated code with manual parts.
This works the following way:
As part of the generator run, you instantiate checks that you write to a file
After the generator finishes, the IDE (here: Eclipse) loads these checks and verifies them against the complete code base (i.e. Generated + manual)
If things don’t conform to the rules, messages are output helping the developer to fix things.
For example, in the state machine case, actions must be implemented in subclasses.
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 31 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Recipes III
Here's an error that suggests that I extend my manually written class from the generated base class:
Recipes can be arranged hierarchically
This is a failed check
„Green" ones can also be hidden
Here you can see additional information about the selected recipe
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 32 -
www.openarchitectureware.org
open Architecture Ware

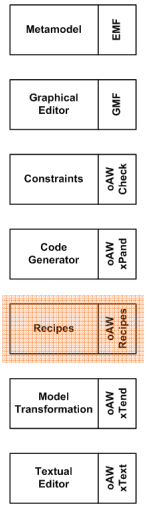

Building DSLs with Eclipse and openArchitectureWare
Recipes IV
I now add the respective extends clause, and the message goes away – automatically.
Adding the extends clause makes all of them green
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 33 -
www.openarchitectureware.org
open Architecture Ware

Recipes V

- Now I get a number of compile errors because I have to **implement the abstract methods** defined in the super class:

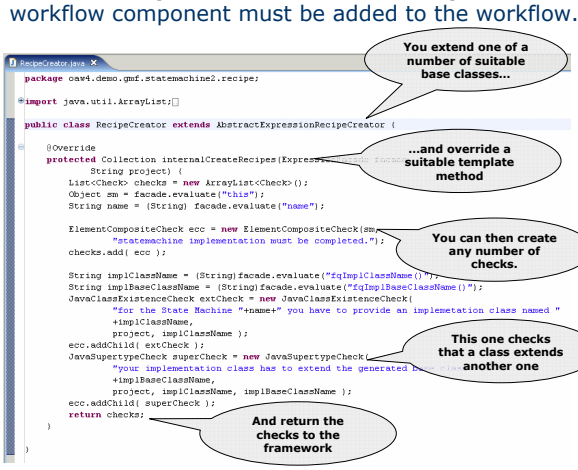


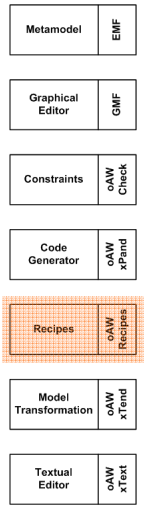
- I finally implement them sensibly, and everything is ok.
- The Recipe Framework and the Compiler have **guided me through the manual implementation steps.**
  - If I didn't like the compiler errors, we could also add recipe tasks for the individual operations.
  - oAW comes with a number of **predefined recipe checks for Java**. But you can also define your own checks, e.g. to verify C++ code.

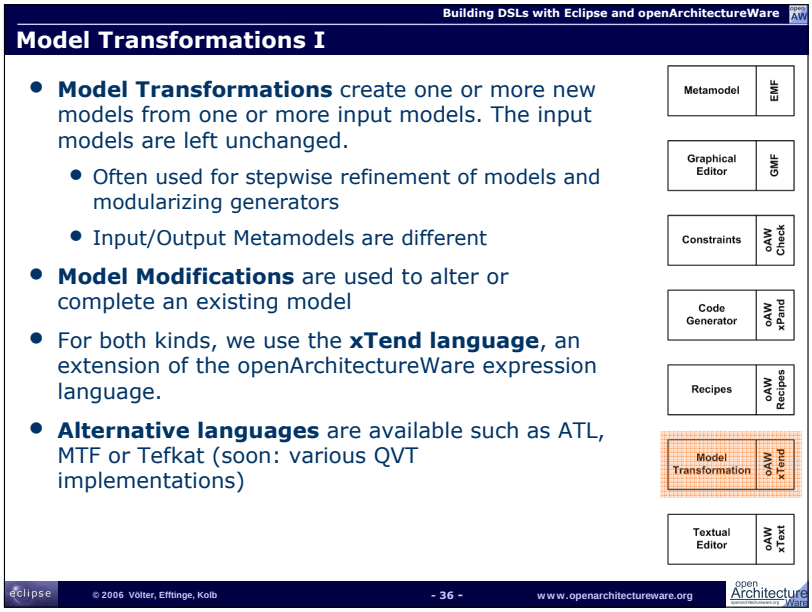


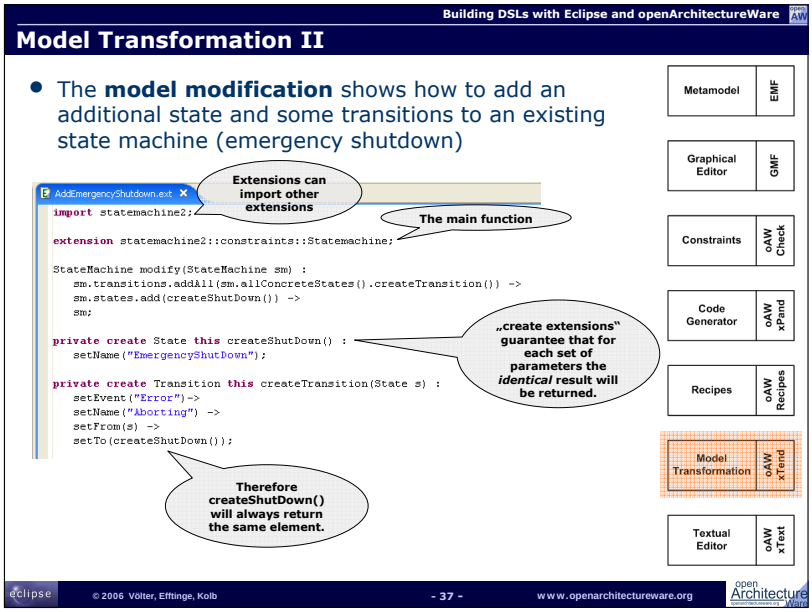
Recipes VI

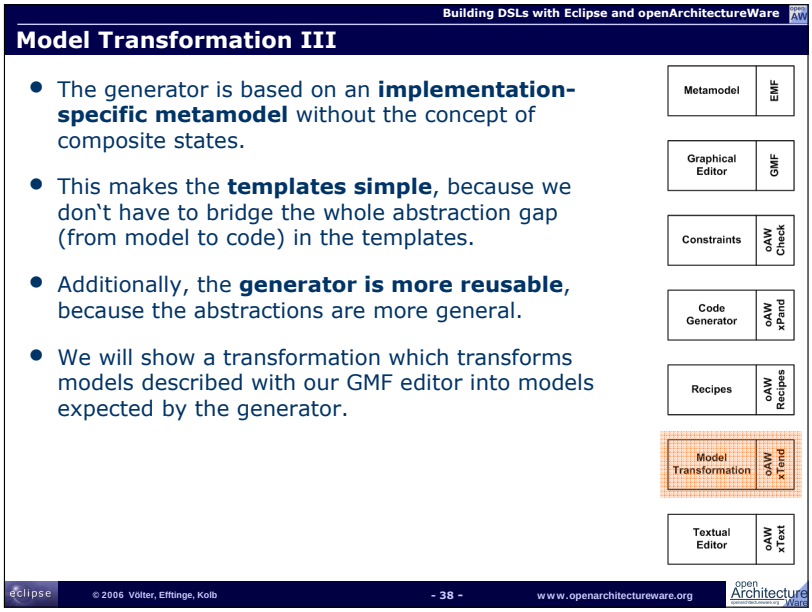
- Here's the **implementation of the Recipes**. This workflow component must be added to the workflow.

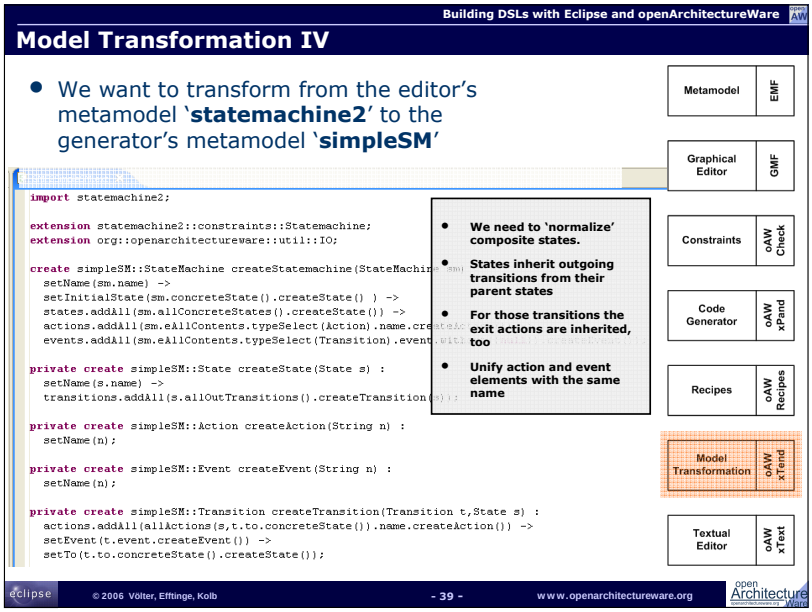


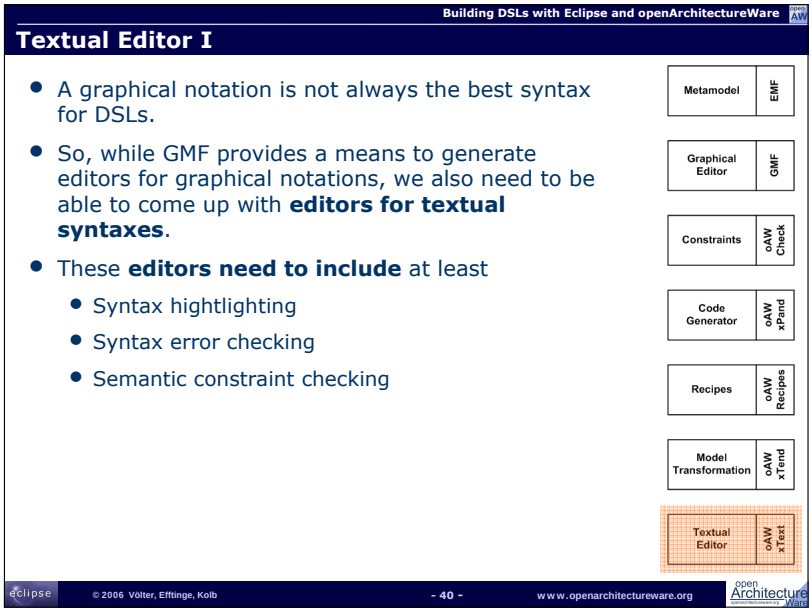


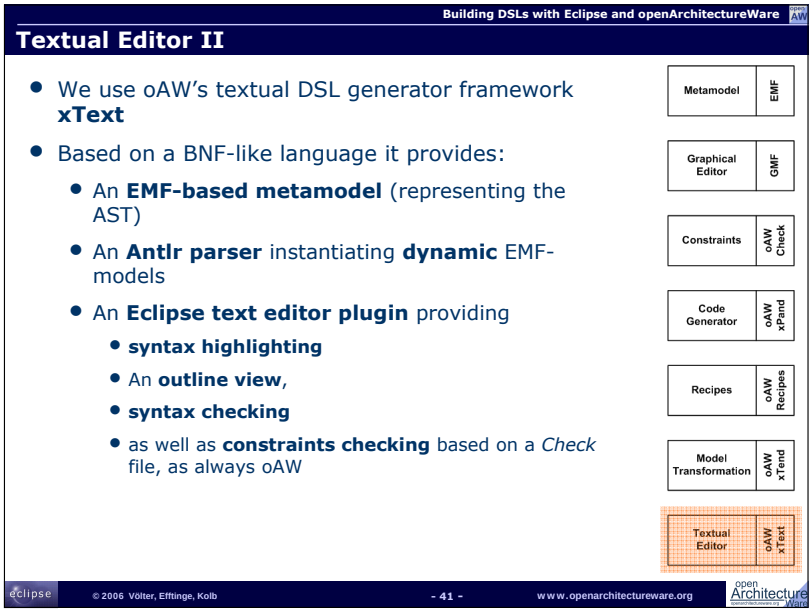

Building DSLs with Eclipse and openArchitectureWare
Model Transformations I
Model Transformations create one or more new models from one or more input models. The input models are left unchanged.
Often used for stepwise refinement of models and modularizing generators
Input/Output Metamodels are different
Model Modifications are used to alter or complete an existing model
For both kinds, we use the xTend language, an extension of the openArchitectureWare expression language.
Alternative languages are available such as ATL, MTF or Tefkat (soon: various QVT implementations)
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 36 -
www.openarchitectureware.org
open Architecture Ware

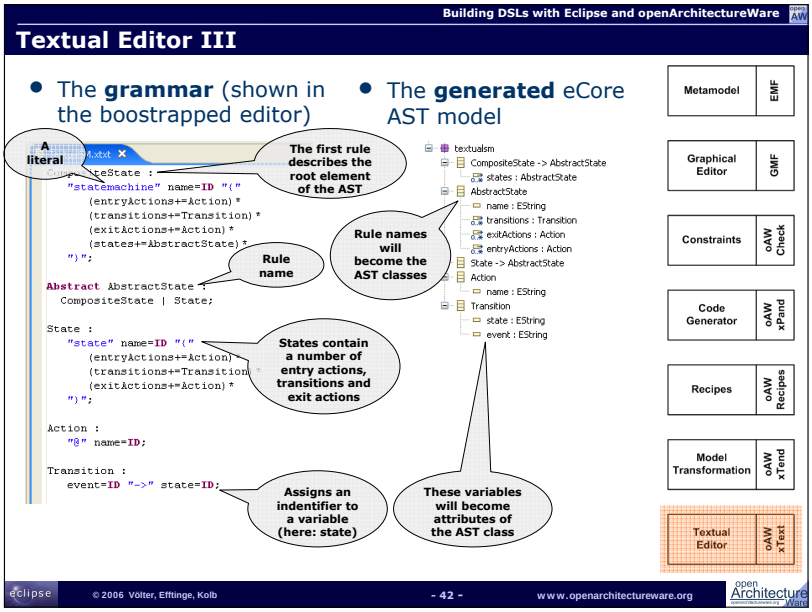

Building DSLs with Eclipse and openArchitectureWare
Model Transformation II
The model modification shows how to add an additional state and some transitions to an existing state machine (emergency shutdown)
AddEmergencyShutdown.ext
Extensions can import other extensions
The main function
import statemachine2;
extension statemachine2::constraints::Statemachine;
StateMachine modify(StateMachine sm) :
sm.transitions.addAll(sm.allConcreteStates().createTransition()) ->
sm.states.add(createShutDown()) ->
sm;
private create State this createShutDown() :
setName("EmergencyShutDown");
private create Transition this createTransition(State s) :
setEvent("Error") ->
setName("Aborting") ->
setFrom(s) ->
setTo(createShutDown());
„create extensions" guarantee that for each set of parameters the identical result will be returned.
Therefore createShutDown() will always return the same element.
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 37 -
www.openarchitectureware.org
open Architecture Ware

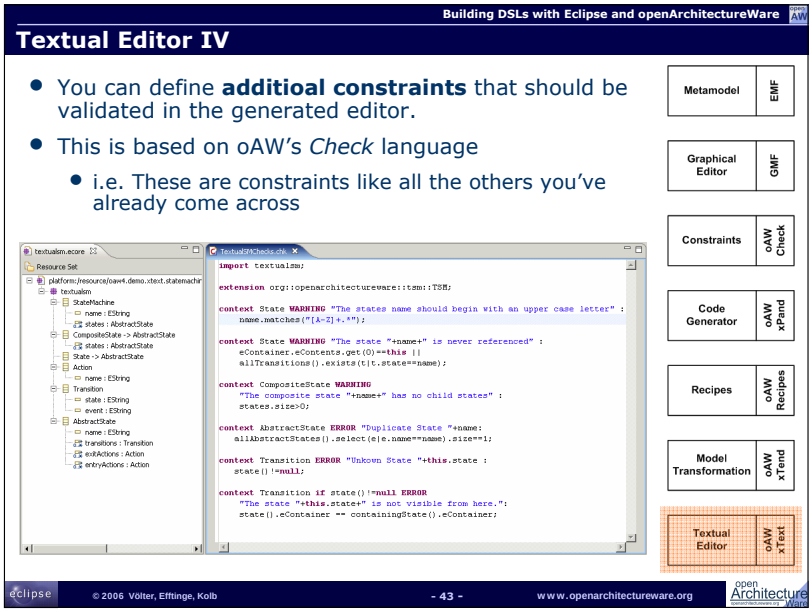

Building DSLs with Eclipse and openArchitectureWare
Model Transformation III
The generator is based on an implementation-specific metamodel without the concept of composite states.
This makes the templates simple, because we don't have to bridge the whole abstraction gap (from model to code) in the templates.
Additionally, the generator is more reusable, because the abstractions are more general.
We will show a transformation which transforms models described with our GMF editor into models expected by the generator.
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 38 -
www.openarchitectureware.org
open Architecture Ware

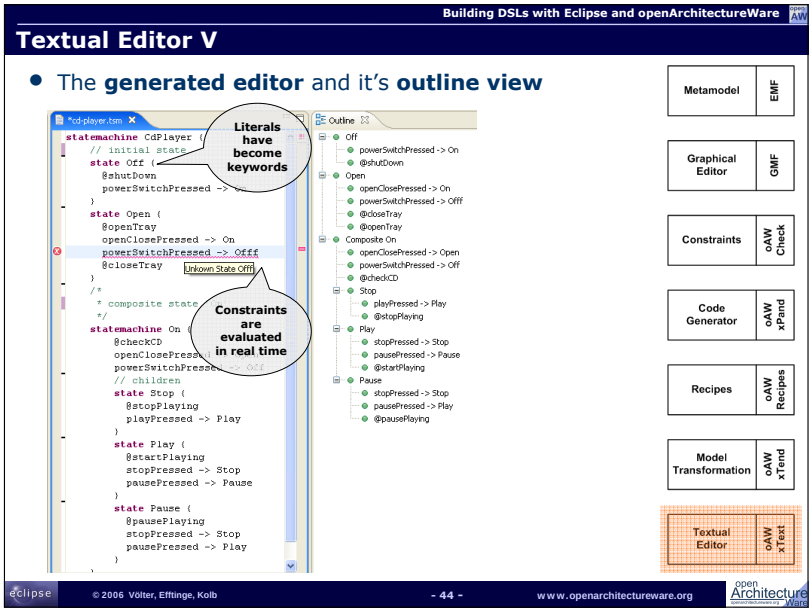

Building DSLs with Eclipse and openArchitectureWare
Model Transformation IV
We want to transform from the editor's metamodel 'statemachine2' to the generator's metamodel 'simpleSM'
import statemachine2;
extension statemachine2::constraints::Statemachine;
extension org::openarchitectureware::util::IO;
create simpleSM::StateMachine createStatemachine(StateMachine sm) :
setName(sm.name) ->
setInitialState(sm.concreteState().createState()) ->
states.addAll(sm.allConcreteStates().createState()) ->
actions.addAll(sm.eAllContents.typeSelect(Action).name.createAction()) ->
events.addAll(sm.eAllContents.typeSelect(Transition).event.withoutNull().createEvent());
private create simpleSM::State createState(State s) :
setName(s.name) ->
transitions.addAll(s.allOutTransitions().createTransition());
private create simpleSM::Action createAction(String n) :
setName(n);
private create simpleSM::Event createEvent(String n) :
setName(n);
private create simpleSM::Transition createTransition(Transition t,State s) :
actions.addAll(allActions(s,t.to.concreteState()).name.createAction()) ->
setEvent(t.event.createEvent()) ->
setTo(t.to.concreteState().createState());
We need to 'normalize' composite states.
States inherit outgoing transitions from their parent states
For those transitions the exit actions are inherited, too
Unify action and event elements with the same name
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 39 -
www.openarchitectureware.org
open Architecture Ware

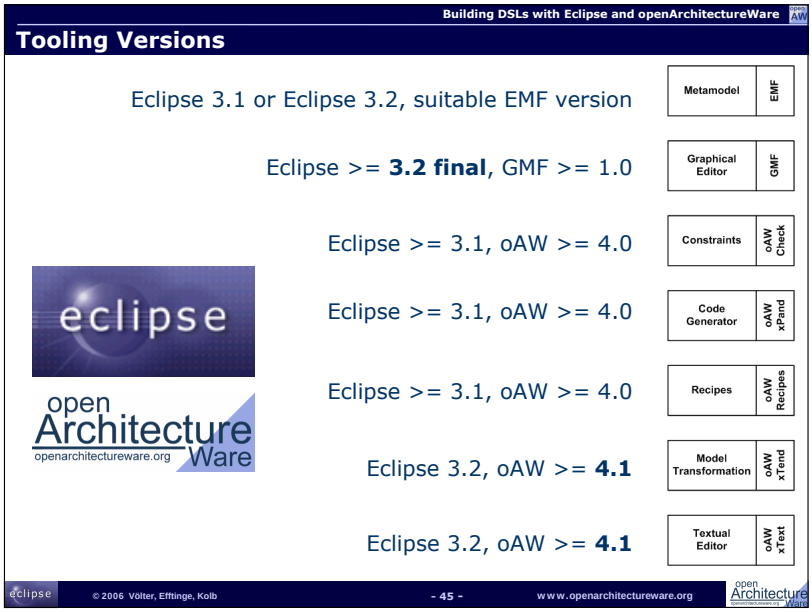

Building DSLs with Eclipse and openArchitectureWare
Textual Editor I
A graphical notation is not always the best syntax for DSLs.
So, while GMF provides a means to generate editors for graphical notations, we also need to be able to come up with editors for textual syntaxes.
These editors need to include at least
Syntax hightlighting
Syntax error checking
Semantic constraint checking
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 40 -
www.openarchitectureware.org
open Architecture Ware

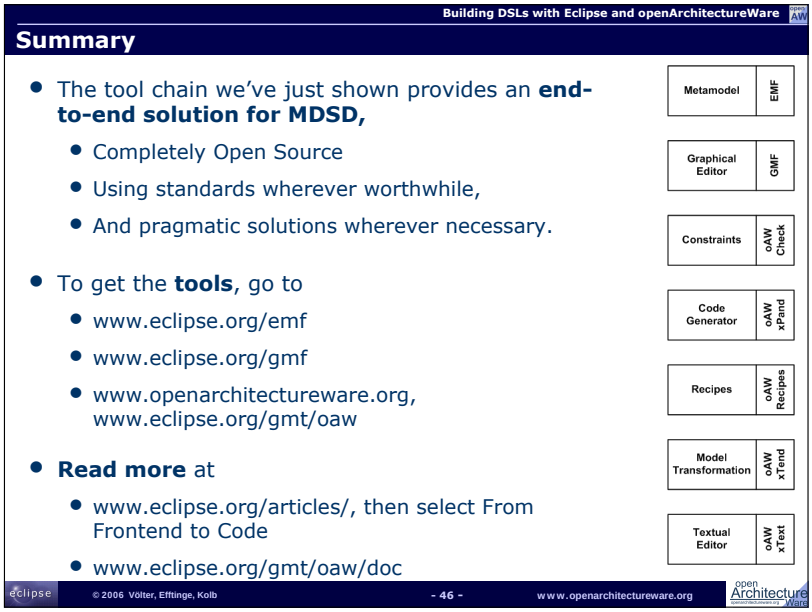

Building DSLs with Eclipse and openArchitectureWare
Textual Editor II
We use oAW's textual DSL generator framework xText
Based on a BNF-like language it provides:
An EMF-based metamodel (representing the AST)
An Antlr parser instantiating dynamic EMF-models
An Eclipse text editor plugin providing
syntax highlighting
An outline view,
syntax checking
as well as constraints checking based on a Check file, as always oAW
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 41 -
www.openarchitectureware.org
open Architecture Ware

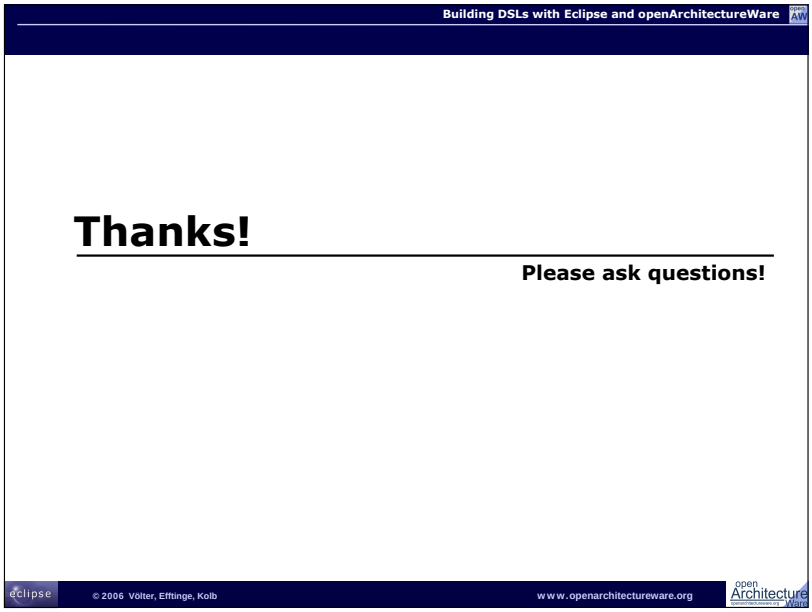

Building DSLs with Eclipse and openArchitectureWare
Textual Editor III
The grammar (shown in the boostrapped editor)
The generated eCore AST model
A literal
The first rule describes the root element of the AST
Rule name
Rule names will become the AST classes
States contain a number of entry actions, transitions and exit actions
Assigns an indentifier to a variable (here: state)
These variables will become attributes of the AST class
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 42 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Textual Editor IV
You can define additioal constraints that should be validated in the generated editor.
This is based on oAW's Check language
i.e. These are constraints like all the others you've already come across
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 43 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Textual Editor V
The generated editor and it's outline view
Literals have become keywords
Constraints are evaluated in real time
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 44 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Tooling Versions
Eclipse 3.1 or Eclipse 3.2, suitable EMF version
Eclipse >= 3.2 final, GMF >= 1.0
Eclipse >= 3.1, oAW >= 4.0
Eclipse >= 3.1, oAW >= 4.0
Eclipse >= 3.1, oAW >= 4.0
Eclipse 3.2, oAW >= 4.1
Eclipse 3.2, oAW >= 4.1
eclipse
open Architecture Ware
openarchitectureware.org
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 45 -
www.openarchitectureware.org
open Architecture Ware


Building DSLs with Eclipse and openArchitectureWare
Summary
The tool chain we've just shown provides an end-to-end solution for MDSD,
Completely Open Source
Using standards wherever worthwhile,
And pragmatic solutions wherever necessary.
To get the tools, go to
www.eclipse.org/emf
www.eclipse.org/gmf
www.openarchitectureware.org, www.eclipse.org/gmt/oaw
Read more at
www.eclipse.org/articles/, then select From Frontend to Code
www.eclipse.org/gmt/oaw/doc
Metamodel
EMF
Graphical Editor
GMF
Constraints
oAW Check
Code Generator
oAW xPand
Recipes
oAW Recipes
Model Transformation
oAW xTend
Textual Editor
oAW xText
eclipse
© 2006 Völter, Efftinge, Kolb
- 46 -
www.openarchitectureware.org
open ArchitectureWare


Building DSLs with Eclipse and openArchitectureWare
Thanks!
Please ask questions!
eclipse
© 2006 Völter, Efftinge, Kolb
www.openarchitectureware.org
open ArchitectureWare

Some advertisement ☺

- For those, who speak (or rather, read) german:

  Völter, Stahl:

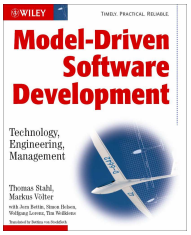
  Modellgetriebene Softwareentwicklung
  Technik, Engineering, Management

  dPunkt, 2005

  www.mdsd-buch.de

- An **very much updated** translation is under way:
  Model-Driven Software Development,
  Wiley, Q2 2006

  www.mdsd-book.org



4.5.3 Diskussion

- *Sollte zukünftig jeder Entwickler seine Sprachen selbst bauen?*
  Oftmals wird der Eindruck erweckt, dass sich als Resultat einer zunehmenden Anzahl an DSL-Entwicklungen ein wild wachsender Zoo an Sprachen ergibt. Das ist ein durchaus ernst zu nehmendes Problem im eigentlichen Scope der DSL-Technologie. Eine Lösung dieses Problems wäre die Standardisierung von Branchen-spezifischen Lösungen, mit allen bereits erwähnten Vor- und Nachteilen.
  Wenn man sich heutzutage die verschiedenen Bibliotheken und Frameworks betrachtet, die von vielen Firmen auch oft ausschließlich für ihre eigenen Belange entwickelt werden, und die Entwicklung von DSLs hier lediglich als eine alternative Technologie betrachtet, erscheint dieser „Sprachenzoo“ in einem anderen Licht.

- *Graphische vs. textuelle DSLs*
  Mit den Strömungen der Modellgetriebenen Software-Entwicklung und der UML wurden graphische (Modellierungs-)Sprachen als Alternative zu textuellen Beschreibungsmitteln immer beliebter. Auch die meisten aktuellen DSL-Werkzeuge bieten vorwiegend graphische Modellierungsmethoden an. Mittlerweile werden allerdings oftmals auch die spezifischen methodischen Vorteile textueller Sprachen wieder entdeckt: Man benötigt keinen speziellen graphischen Editoren, sondern lediglich einen gewöhnlichen Texteditor; Bildliche Darstellungen drängen oftmals semantische Assoziationen auf, die unter Umständen nicht konform zur Sprachsemantik sind.

- *Modulare Sprachentwicklung*
  Die meisten in der Praxis verwirklichten DSLs sind sich in ihrer Struktur recht ähnlich. Sehr häufig trifft man beispielsweise auf typische Architekturbeschreibungen in denen logische Komponenten, die über bestimmte Konnektoren verbunden sind, schließlich auf eine Beschreibung der Topologie abgebildet werden. Die Metamodelle derartiger DSLs sind sich dementsprechend ebenso ähnlich und werden trotzdem immer wieder von neuem entwickelt. In Anbetracht dessen wäre ein Modularisierungskonzept auf Sprach- bzw. Metamodellebene wünschenswert, das es erlaubt, Sprachen aus Bausteinen (Modulen) zusammenzusetzen.

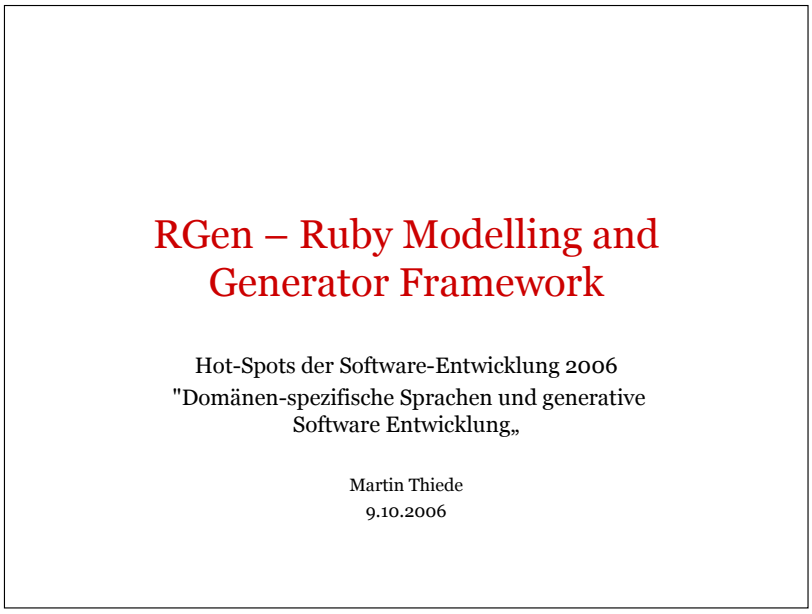
4.6 Martin Thiede: „RGen – Ruby Modelling and Generator Framework“

4.6.1 Abstract

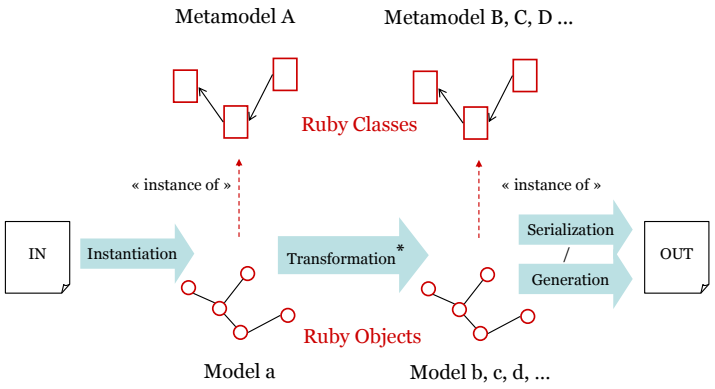
This presentation shows how Ruby's characteristic language features can be used to make model transformation and code generation more powerful and simple. The required concepts have been integrated into the open source RGen framework. Parts of the framework itself can be seen as Domain Specific Languages implemented in Ruby.

4.6.2 Folien

RGen



RGen - The Big Picture



Why RGen ?

- Flexibility
  - Simple customization
  - Support unforeseen cases
- Simplicity
  - Easy to setup up
  - Few things to remember
  - Small footprint
  - Easy to maintain
- Ruby‘s Role
  - One single language
  - Code is specification
- Use Cases
  - Smaller, non-GUI projects
  - Converters
  - Generators
  - Fast prototyping

Ruby Primer

- Concise
  - Things can be expressed with very little code
- No inherent type enforcement
  - „Duck Typing“ is a feature, not a bug
- Almost everything is an object
  - Classes are objects, 3 is an object
- Meta Programming
  - Add and remove classes and methods at runtime
- Additional powerful features
  - Mixins, Iterators, Continuations, ..
- Interpreted, Multi Platform, Open Source

Ruby Primer
Blocks / Iterators

```
a = [ 1, 2.3, „TestString“ ]
```

Array‘s `find` method

Block

```
a.find { |e| e.is_a?(Float) }
   => 2.3
```

Block Parameter

```
a.find do |e| e.is_a?(Float) end
   => 2.3
```

Possible Implementations:

```
class Array

  def find
    each { |e|
      return if yield(e)
    }
  end

end
```

Call the method‘s block

Proc Object

```
class Array

  def find(&b)
    each { |e|
      return if b.call(e)
    }
  end

end
```

Call the block using Proc object

Ruby Primer
Evaluation of Class Definition

- No direct attribute access
- Accessor methods
- Macros to build accessors

```
class Person

   attr_accessor :name

end

p = Person.new
p.name = „Tom“
p.name
  => „Tom“
```

creates

Class Method, called when class definition is evaluated

Equivalent:

```
class Person

   def name
      @name
   end
   def name=(n)
      @name = n
   end

end
```


Ruby Primer
Meta Programming

Possible implementation of `attr_accessor`:

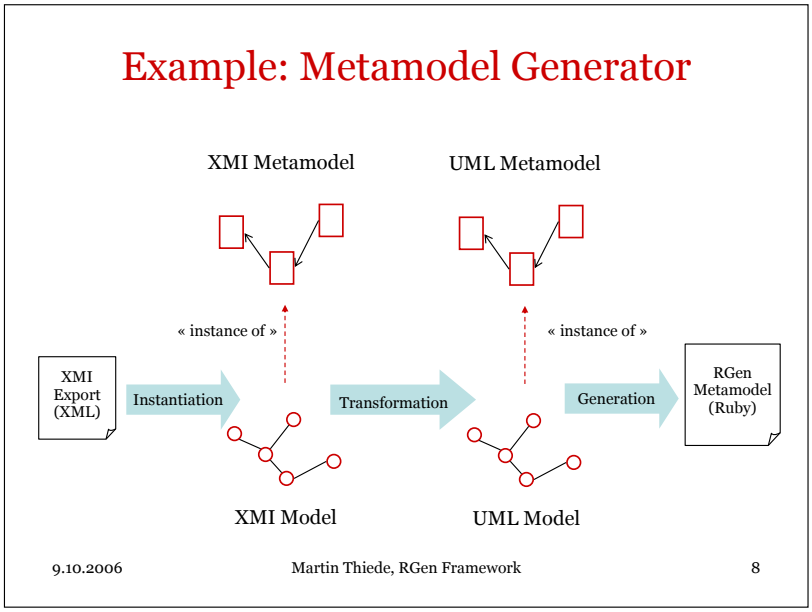
```
class Class

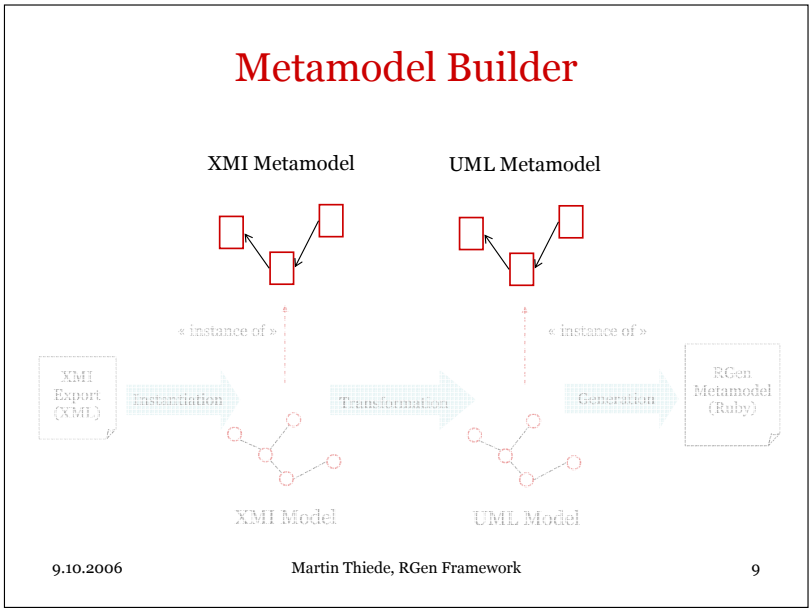
   def self.attr_accessor(sym)
      module_eval <<-CODE
         def #{sym}
            @#{sym}
         end
         def #{sym}=(v)
            @#{sym} = v
         end
      CODE
   end

end
```

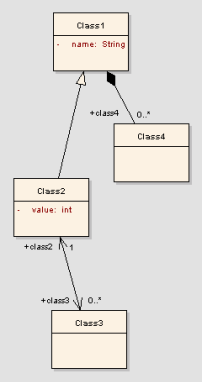
Class Method

Methods to be added when the class method is called


Example: Metamodel Generator
XMI Metamodel
UML Metamodel
« instance of »
« instance of »
XMI Export (XML)
Instantiation
Transformation
Generation
RGen Metamodel (Ruby)
XMI Model
UML Model
9.10.2006
Martin Thiede, RGen Framework
8


Metamodel Builder
XMI Metamodel
UML Metamodel
« instance of »
« instance of »
XMI Export (XML)
Instantiation
Transformation
Generation
RGen Metamodel (Ruby)
XMI Model
UML Model
9.10.2006
Martin Thiede, RGen Framework
9

Metamodel Builder
Builder Methods




```
class Class1 < MMBase
   has_one 'name', String
end

class Class2 < Class1
   has_one 'value', Integer
end

class Class3 < MMBase
end

class Class4 < MMBase
end

Class1.one_to_many, 'class4', Class4, 'class1'
Class2.one_to_many, 'class3', Class3, 'class2'
```

More Builder Methods

```
has_many, one_to_one, many_to_many, ..
```


Metamodel Builder
A Simple UML Metamodel

```
class UMLPackage < MMBase
   has_one 'name', String
end

class UMLClass < MMBase
   has_one 'name', String
   has_many 'stereotypes', String
end

class UMLAttribute < MMBase
   has_one 'name', String
   has_one 'type', String
end

class UMLAssociation < MMBase
end

class UMLAggregation <
   UMLAssociation
end

class UMLAssociationEnd < MMBase
   has_one 'multiplicity', String
   has_one 'navigable'
   has_one 'composite'
   has_one 'role', String
end

UMLPackage.one_to_many 'subpackages', UMLPackage, 'superpackage'
UMLPackage.one_to_many 'classes', UMLClass, 'package'
UMLClass.many_to_many 'superclasses', UMLClass, 'subclasses'
UMLClass.one_to_many 'assocEnds', UMLAssociationEnd, 'clazz'
UMLClass.one_to_many 'attributes', UMLAttribute, 'clazz'
UMLAssociation.one_to_one 'endA', UMLAssociationEnd, 'assocA'
UMLAssociation.one_to_one 'endB', UMLAssociationEnd, 'assocB'
```

Metamodel Builder
Navigating the Model

```
class UMLClass < MMBase
  def remoteNavigableEnds
    assocEnds.otherEnd.select { |e| e.navigable }
  end
end

class UMLAssociationEnd < MMBase
  has_one 'navigable'
  def assoc
    assocA || assocB
  end
  def otherEnd
    return unless assoc
    assoc.endA == self ? assoc.endB : assoc.endA
  end
end

UMLClass.one_to_many 'assocEnds', UMLAssociationEnd, 'clazz'
```

„otherEnd“ is called on the result of „assocEnds“, which is an array

Metamodel Builder
Calling Methods on Array Elements

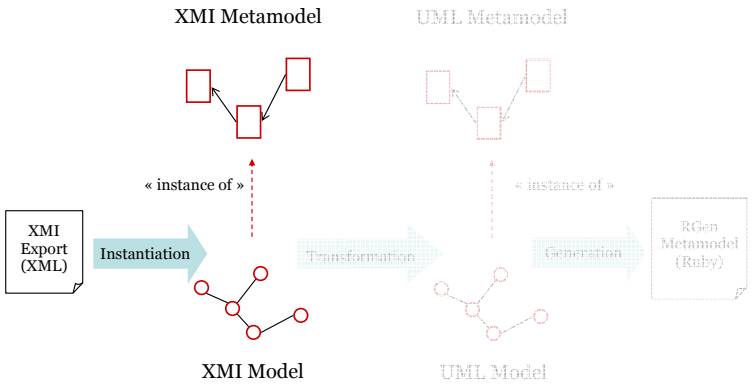
- Ruby‘s built-in `Array` class is extended (Open Classes)
- Ruby‘s `method_missing` is used to catch calls to unknown methods
- Calls are delegated to the Arrays‘ elements
- Results are combined into one single set and returned

```
class Array

   def method_missing(m, *args)
      super unless size == 0 or compact.any?{|e| e.is_a? MMBase}
      compact.inject([]) { |r,e|
         if e.is_a? MMBase
            r | ( (o=e.send(m)).is_a?(Array) ? o : [o] )
         else
            raise StandardError.new("Trying to call ..“)
         end
      }.compact
   end

end
```

XML Instantiator



9.10.2006 Martin Thiede, RGen Framework 14

XML Instantiator
Automatic Metamodel

XML Input

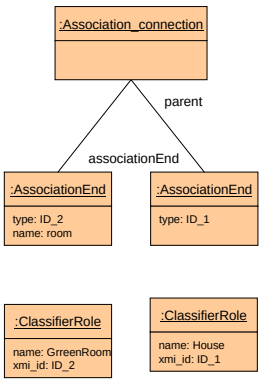
```
<UML:Association xmi.id=„ID_3 />
  <UML:Association.connection>
    <UML:AssociationEnd name="room" type=„ID_2„ />
    <UML:AssociationEnd type=„ID_1„ />
  </UML:Association.connection>
</UML:Association>

<UML:ClassifierRole name="House" xmi.id=„ID_1„>
</UML:ClassifierRole>

<UML:ClassifierRole name="GreenRoom" xmi.id=„ID_2„>
</UML:ClassifierRole>
```

Calling the Instantiator

```
env = RGen::Environment.new
i = MyInstantiator.new(env, XMIMetaModel, true)
i.instantiate_file("testmodel.xml")
```



9.10.2006 Martin Thiede, RGen Framework 15

XML Instantiator
Resolving References

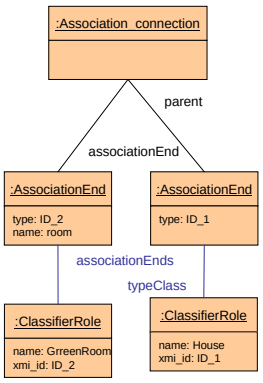
Partial Metamodel

```
module XMIMetaModel
  module UML
    class ClassifierRole < MMBase; end
    class AssociationEnd < MMBase; end
    ClassifierRole.one_to_many 'associationEnds',
      AssociationEnd, 'typeClass,
  end
end
```

Custom Instantiator

```
class XMIClassInstantiator < RGen::XMLInstantiator
  map_tag_ns "omg.org/UML1.3", XMIMetaModel::UML

  resolve_by_id :typeClass, :src => :type,
    :id => :xmi_id
end
```



XML Instantiator
More Customizing

```
class MyInstantiator < RGen::XMLInstantiator

  def assoc_p2c(parent, child)
    parent.object.addGeneric(saneMethodName(child), child.object)
    child.object.setGeneric("parent", parent.object)
  end

  def create_p2c_assoc(parent, child)
    parent.object.class.has_many(saneMethodName(child), child.object.class)
    child.object.class.has_one("parent", RGen::MetamodelBuilder::MMBase)
  end

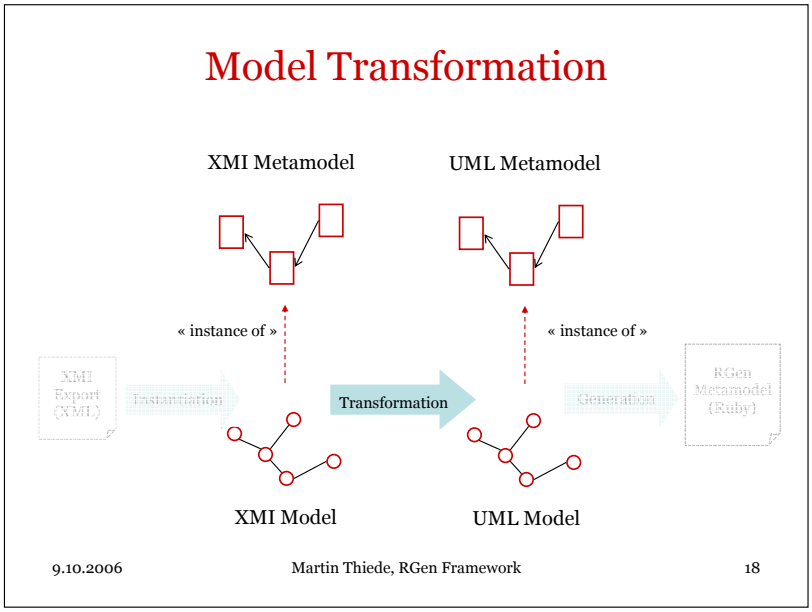
  resolve :referencedElement, :class => Element do
    @env.find(:class => Element, :name => extractName(ref)).first
  end

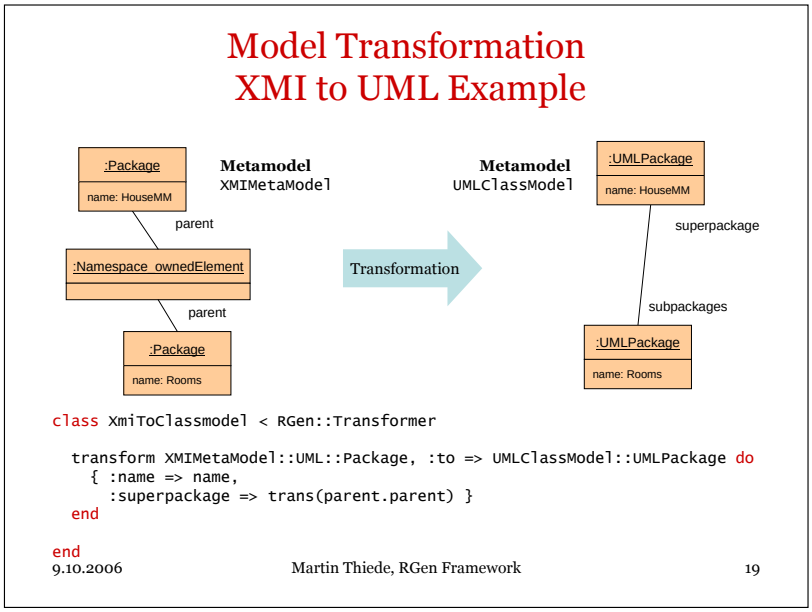
end
```

Parent to Child Association

Parent to Child Association Metamodel

Resolver for Attribute „referencedElement“


Model Transformation
XMI Metamodel
UML Metamodel
« instance of »
« instance of »
XMI Export (XML)
Instantiation
Transformation
Generation
RGen Metamodel (Ruby)
XMI Model
UML Model
9.10.2006
Martin Thiede, RGen Framework
18


Model Transformation
XMI to UML Example
:Package
name: HouseMM
Metamodel
XMIMetaModel
parent
:Namespace_ownedElement
parent
:Package
name: Rooms
Transformation
Metamodel
UMLClassModel
:UMLPackage
name: HouseMM
superpackage
subpackages
:UMLPackage
name: Rooms
class XmiToClassmodel < RGen::Transformer
transform XMIMetaModel::UML::Package, :to => UMLClassModel::UMLPackage do
{ :name => name,
:superpackage => trans(parent.parent) }
end
end
9.10.2006
Martin Thiede, RGen Framework
19

Model Transformation
Ruby as a Specification Language

```
class XmiToClassmodel < RGen::Transformer
  def transform
    trans(:class => XMIMetaModel::UML::Clazz)
  end

  transform XMIMetaModel::UML::Package :to => UMLClassModel::UMLPackage do
    { :name => name,
      :superpackage => trans(parent.parent) }
  end

  transform XMIMetaModel::UML::Clazz, :to => UMLClass do
    { :name => name, :package => trans(parent.parent),
      :attributes => trans(classifier_feature.attribute),
      :operations => trans(classifier_feature.operation),
      :subclasses =>  trans(generalizationsAsSupertype.subtypeClass),
      :assocEnds => trans(associationEnds) }
  end

  transform XMIMetaModel::UML::Attribute, :to => UMLAttribute do
    tv = TaggedValueHelper.new(@current_object)
    { :name => name, :type => tv['type'] }
  end
end
```

Source Class

Target Class

Methods called on source object

Arbitrary Ruby Code

Model Transformation
Making a Copy

A rule to copy `UMLClass` elements

```
transform UMLClass, :to => UMLClass do
  { :name => name,
    :package => trans(package),
    # ...
    :operations => trans
end
```

Implementation

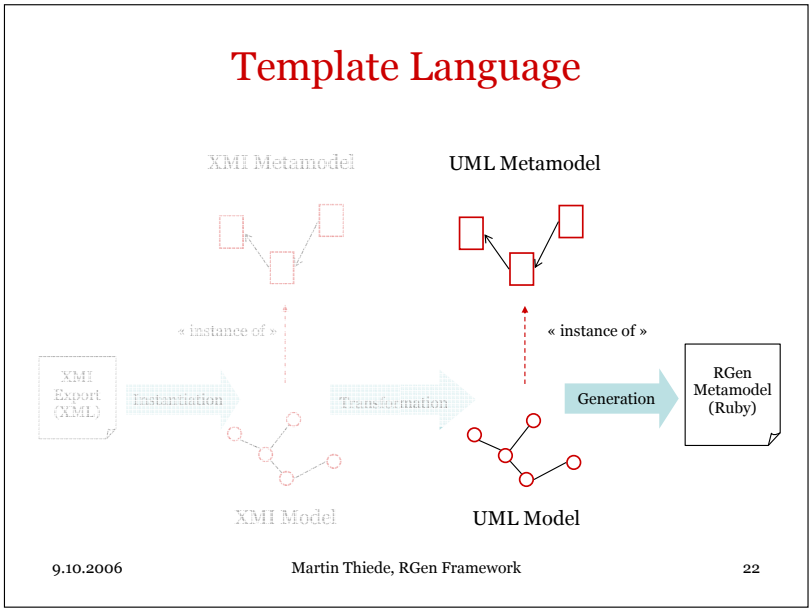
```
def self.copy(from, to=nil)
  transform(from, :to => to || from) do
    Hash[*(@current_object.class.one_attributes +
    @current_object.class.many_attributes).inject([]) {|l,a|
      l + [a.to_sym, trans(@current_object.send(a))]
    }]
  end
end
```

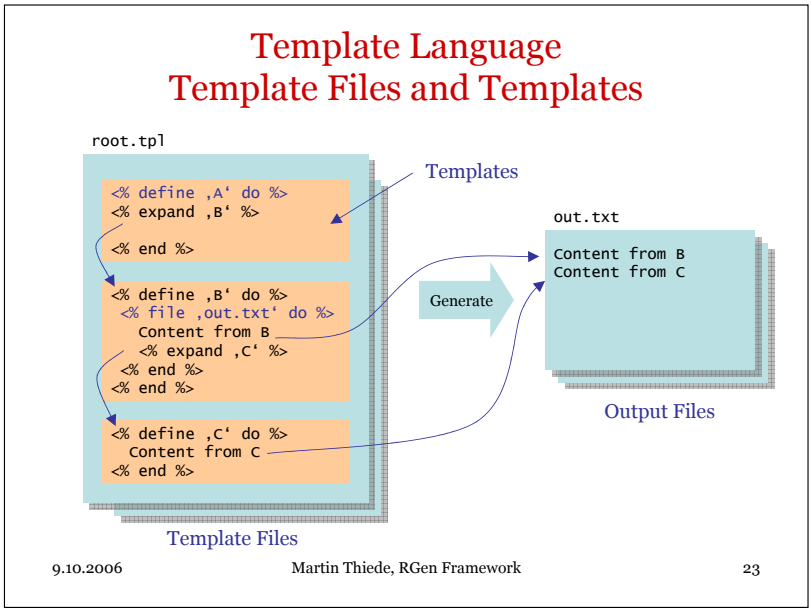
Equivalent

```
copy UMLClassModel
```

Copy a whole UML model

```
class CopyTransformer < RGen::Transformer
  include UMLClassModel
  def transform
    trans(:class => UMLPackage)
  end
  constants.each{|c| copy const_get(c) if c =~ /^UML/}
end
```


Template Language
XMI Metamodel
UML Metamodel
« instance of »
« instance of »
XMI Export (XML)
Instantiation
Transformation
Generation
RGen Metamodel (Ruby)
XMI Model
UML Model
9.10.2006
Martin Thiede, RGen Framework
22


Template Language
Template Files and Templates
root.tpl
<% define ‚A‘ do %>
<% expand ‚B‘ %>
<% end %>
<% define ‚B‘ do %>
<% file ‚out.txt‘ do %>
Content from B
<% expand ‚C‘ %>
<% end %>
<% end %>
<% define ‚C‘ do %>
Content from C
<% end %>
Templates
Generate
out.txt
Content from B
Content from C
Output Files
Template Files
9.10.2006
Martin Thiede, RGen Framework
23

Template Language
Generating a RGen Metamodel

```
<% define 'GeneratePackage', :for => UMLPackage do |modules| %>
  module <%= moduleName %><% iinc %>
    <% for c in sortedClasses %><%nl%>
      <% expand 'ModuleHeader', modules, :for => c %><%iinc%>
        <% (c.superclasses.name & modules).each do |n| %>
          include <%= n %>
        <% end %>
        <% if modules.include?(c.className) %>
          extend RGen::MetamodelBuilder::BuilderExtensions
        <% end %>
        has_one 'name', String
        <% for a in c.attributes %>
          has_one '<%= a.name %>', <%= a.RubyType %>
        <% end %><%idec%>
      end
    <% end %><%nl%>
    <% for p in subpackages %>
      <%nl%><% expand 'GeneratePackage', modules, :for => p %>
    <% end %><%idec%>
  end
<% end %>
```

Context Type

Template Parameters

Whitespace Control

RGen
Getting Started

- RGen is Open Source (MIT License)
  http://rubyforge.org/projects/rgen
- Requirements: Ruby 1.8.2 ff
  http://www.ruby-lang.org/
- Download from rubyforge.org using gem (175 kB)
  ```
  > gem install rgen
  ```
- Run the Unit Tests
  ```
  > gem unpack rgen
  > cd rgen-0.3.0
  > ruby test\rgen_test.rb
  ```
- Documentation
  ```
  <ruby_dir>\lib\ruby\gems\1.8\doc\rgen-0.3.0\index.html
  ```

Summary

- Ruby
  - Is a concise way of expressing ideas
  - Can be used to extend the language itself (DSL)
- RGen
  - Lightweight and flexible
  - Can be used to create generators, convertes, etc. quickly
- Drawbacks
  - Name conflicts: Class „Class“, Attribute „id“, ..
- Next Steps
  - Improve Transformer, XML Schema Support, EMF Support
  - More Documentation

A Prototype Communication Layer Generator with RGen

RGen based COM Layer Generator
Martin Thiede
BMW CarIT
09.10.06
Seite 1

A Prototype Communication Layer Generator.

Applying the Ruby Modelling and Generator Framework (RGen).

RGen based COM Layer Generator

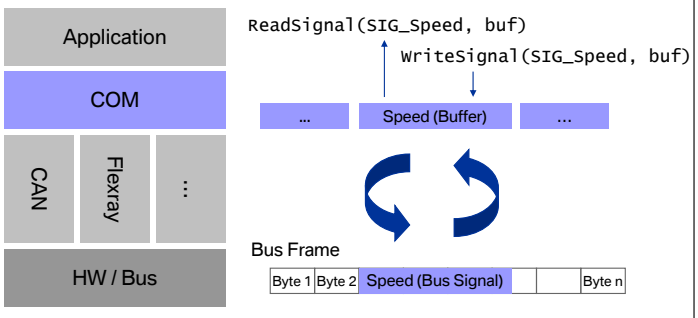
München
Oktober 2006

RGen based COM Layer Generator
Martin Thiede
BMW CarIT
09.10.06
Seite 2

RGen based COM Layer Generator.

The Communication Layer.

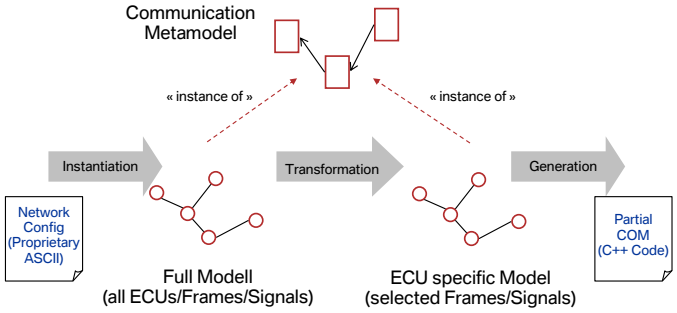
– Embedded Software Module (C/C++)
– Services to send and receive Frames/Signals on the bus
– Buffers and Signal Handles are application specific
=> Parts of the module need to be generated



RGen based COM Layer Generator Martin Thiede BMW CarIT 09.10.06 Seite 3

RGen based COM Layer Generator.
Generation Process Overview.

- Parse network configuration data
- Instantiate the network configuration model
- Transform model to select Frames/Signals for the ECU
- Generate parts of the COM Layer



RGen based COM Layer Generator Martin Thiede BMW CarIT 09.10.06 Seite 4

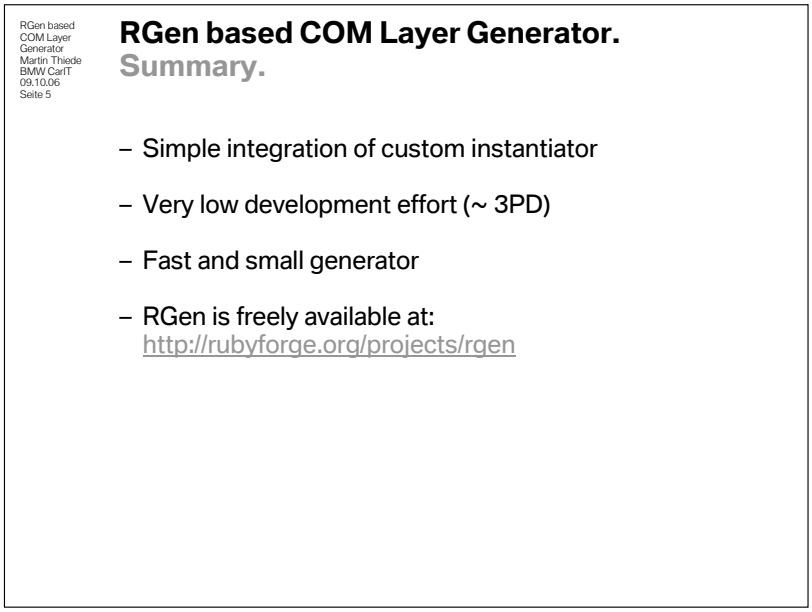
RGen based COM Layer Generator.
Size and Performance.

Generator Code Size (LOC)

– Metamodel	53
– Instantiator	72
– Transformer	35
– Templates	187
– Total	396

Generator Performance (Runtime)

–Instantiation (~1500 Config Nodes)	
–Selection (~ 250 Config Nodes)	
–Generation (~ 1000 LOC Output)	< 1s



4.6.3 Diskussion

- *Vorteile von leichtgewichtigen Generatoren*
  Die meisten DSL-Tools und Generator-Frameworks sind sehr schwergewichtige Werkzeuge, die einiges an Aufwand erfordern, bevor man mit ihnen zu Ergebnissen kommt. Auch die erzeugten Generatoren sind oftmals sehr unhandlich, nicht nur aufgrund ihres großen Footprints. Gerade durch die Nutzung von Metaprogrammierungsfähigkeiten bestimmter Programmiersprachen (Ruby, Smalltalk, ...) ist es möglich, leichtgewichtigere Generatoren zu entwickeln. Dies bietet sich vor allem in Situationen an, in denen kleinere Transformationsaufgaben effizient erfüllt werden sollen.

- *Anforderungen an den Entwickler*
  Als möglicher Nachteil dieses leichtgewichtigen Ansatzes kann die notwendige Qualifikation auf Entwicklerseite gesehen werden: Da die Meta- und die Instanzebene in der gleichen Sprache realisiert sind, wird vom Entwickler eine hohe geistige Flexibilität gefordert, da er sich immer bewußt sein muss, auf welcher Ebene er sich bei der Programmierung gerade bewegt.