openArchitectureWare

Enrico Schnepel

EAS, FHTW-Berlin

07.06.2007

Gliederung

Gliederung

Einleitung

Große Softwaresysteme:

- oft schwer zu verwalten
- hohe Komplexität
- mit normalen IDEs oft nicht mehr verwaltbar

Einsatz von Modellen:

- Konzentration auf die wesentlichen Merkmale einer funktionalen Komponente (z.B EJB)
- Verwendung spezialisierter Editoren
- erhebliche Reduktion der Gesamtkomplexität eines Systems

Metamodell:

- Domänenspezifische Sprache (Domain-Specific Language, DSL)
- formale Beschreibung des Modells

Generatoren:

- Generierung großer Teile von komplexen Softwaresystemen
- architekturzentrierter Ansatz, MDA der OMG, MDSD
- "Infrastrukturcode"

Gliederung

Analogien - ”wie aus dem Leben gegriffen” (1/2)

Übertragene Bedeutung auf einen Bauunternehmer:

Modell

- Bauzeichnung eines Hauses
- Technische Zeichnungen von Türen, Fenstern, etc.

Metamodell

- Beschreibung zur Anfertigung von Bauzeichnungen für Häuser bzw. von technischen Zeichnungen
- DIN- bzw. ISO-Normen

Generator

- Wertschaffende Berufe (Maurer, Dachdecker, Maler, Tischler)
- Baumaterialien (Beton, Holz, Außenwandmalfarbe)
- entsprechende Geräte (Betonmischer, Kran, Leiter)

Plattform und Frameworks

- Fundament - Es muss bereits vorhanden sein

Analogien - ”wie aus dem Leben gegriffen” (2/2)

Metamodelle

- Haus (DIN ISO 128)
- Fenster und Tür (DIN ISO 128)
- Fenstergriff und Türgriff (DIN ISO 128)
- Fussbodenbelag (Bendienungsanleitung des Fotoaparates)
- Wandfarbe (Farbtabelle)

Referenzen

Haus $\Rightarrow$ Fenster $\Rightarrow$ Fenstergriff
$\Rightarrow$ Tür $\Rightarrow$ Türgriff
$\Rightarrow$ Teppich
$\Rightarrow$ Wandfarbe

Generatoren

Möglichkeiten der Nutzung von Generatoren in der Bauwirtschaft:

- Erzeugung von beliebig vielen Häusern der gleichen Bauart
- Einbau neuer Fenster, ohne den Innenausbau anzufassen bzw. die Hausfassade neu zu streichen.
- Anbau eines neuen Seitenflügels an das bestehende Haus mit
  - bereits gestrichenen Wänden
  - bereits verlegtem Teppichbelag
  - ohne Inneneinrichtung
- eine mit Graffiti beschmierte Wand, marode Bausubstanz oder auch der inzwischen durchgelaufene Teppichbelag wird einfach neu generiert. Die Inneneinrichtung wird nicht verändert.

Nachteil:

- Weltweit steigende Arbeitslosenzahlen

Gliederung

Was ist openArchitectureWare?

"openArchitectureWare" (kurz: "oAW"):

- Framework zum Verarbeiten von strukturierten Informationen
- Validierung von Modellen mit dem Check-Framework
- Erzeugung von beliebigen Ausgabe-Dateien (Java, XML) mit dem XPand-Framework (der eigentliche Generator)
- Modell-zu-Modell-Transformationen mit dem Extend-Framework
- Modellerstellung durch Parsen von Text-Dateien mit dem XText-Framework

oAW ist in verschiedenen Umgebungen lauffähig:

- direkt aus Eclipse heraus
- als Ant-Task
- direkt aus Java
- als eigenständiges Programm (Shell)

Was ist openArchitectureWare?

Desweiteren ist oAW:

- ein Top-Level-Projekt im Rahmen von Eclipse (http://www.eclipse.org/gmt/oaw/),
- gut in Eclipse integriert und
- im Vergleich zu anderen Eclipse-basierten Templating-Engines (z.B. JET) einfach zu benutzen und trotzdem komplex ausbaufähig.

Um oAW in Eclipse verwenden zu können werden verschiedene Plugins benötigt:

- oAW-Plugins (ohne die geht es nicht)
- EMF (UML-ähnliche Notation von Metamodellen, DSL-Baumeditoren)
- GMF (für die Erstellung von grafischen DSL-Editoren)

Gliederung

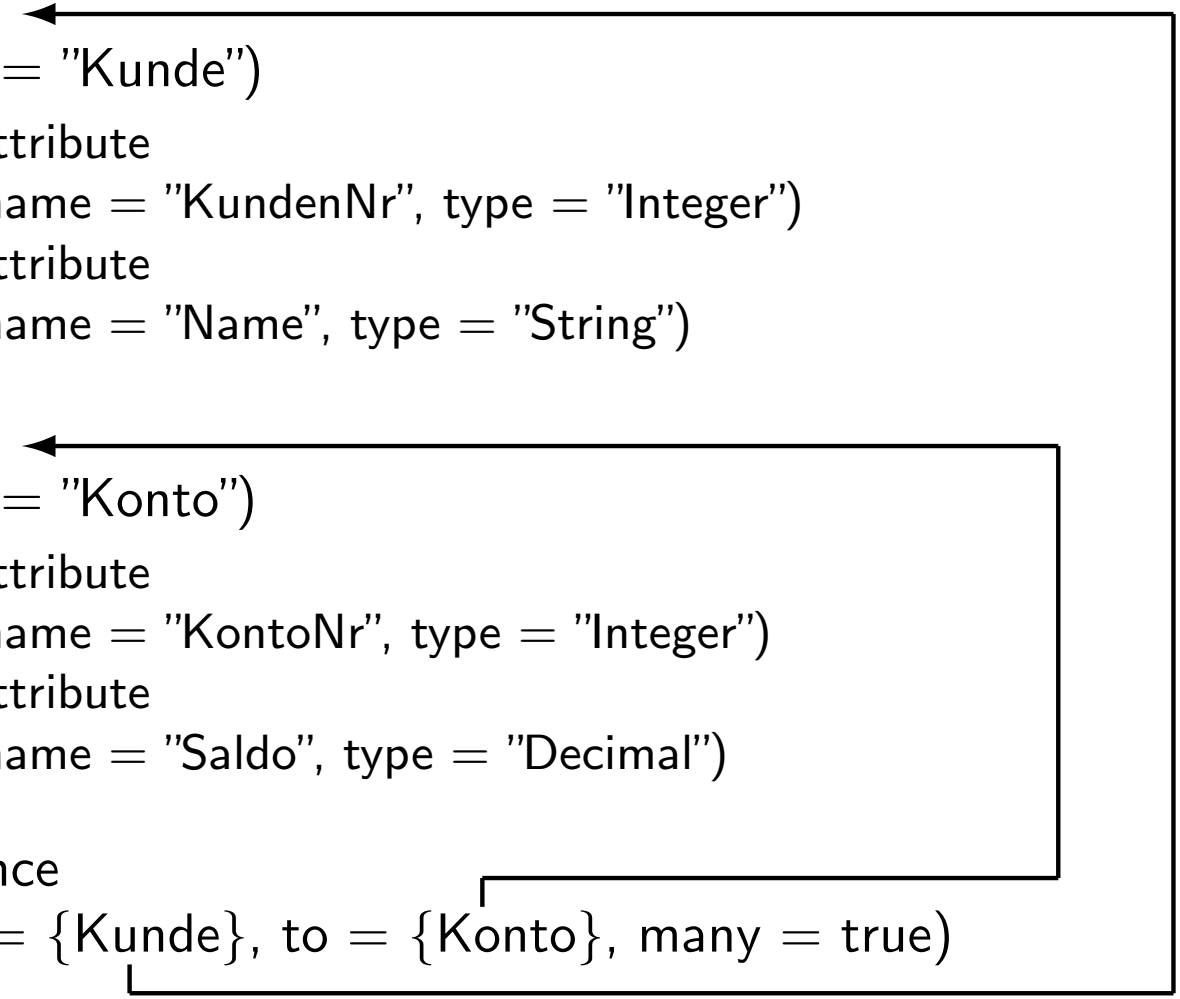
Beispiel - Metamodell

Folgendes Metamodell bildet ein ERM ab:

- Root-Type Domain
  - name (Type: String)
  - entities (Type: List<Entity>, 1..many)
  - references (Type: List<Reference>, 0..many)
- Entity
  - name (Type: String)
  - attributes (Type: List<Attribute>, 1..many)
- Attribute
  - name (Type: String)
  - type (Type: String)
- Reference
  - from (Type: $\rightarrow$ Entity)
  - to (Type: $\rightarrow$ Entity)
  - many (Type: Boolean)

Beispiel - Modell

- Domain
  (name = "Kunde-Konto")
  - Entity
    (name = "Kunde")
    - Attribute
      (name = "KundenNr", type = "Integer")
    - Attribute
      (name = "Name", type = "String")
  - Entity
    (name = "Konto")
    - Attribute
      (name = "KontoNr", type = "Integer")
    - Attribute
      (name = "Saldo", type = "Decimal")
  - Reference
    (from = {Kunde}, to = {Konto}, many = true)



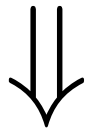
Gliederung

Workflow - Komponentenbasierte Modellverarbeitung

- Konfiguration der Verarbeitungsschritte für ein oder mehrere Modelle nach dem EVA-Prinzip
  - Eingabe (XMIReader, EMFReader, XText)
  - Verarbeitung (Check, Extend, Java)
  - Ausgabe (XPand, XMIWriter, EMFWriter)
- Nutzung von Unter-Workflows (Komponenten) und Cartridges

org.open architectureware.emf.XmiReader

Die Daten werden aus einer Modell-Datei gelesen und im Speicher unter dem Namen 'model' abgelegt.



org.openarchitectureware.XPand2.Generator

Aus dem im Speicher befindlichen Modell werden Dateien generiert.

Workflow - Komponentenbasierte Modellverarbeitung

- Konfiguration der Verarbeitungsschritte für ein oder mehrere Modelle nach dem EVA-Prinzip
  - Eingabe (XMIReader, EMFReader, XText)
  - Verarbeitung (Check, Extend, Java)
  - Ausgabe (XPand, XMIWriter, EMFWriter)
- Nutzung von Unter-Workflows (Komponenten) und Cartridges

Beispiel

```
<workflow>
    <property name='genPath' value='src-gen'/>
    <property name='model' value='model/kunde-konto.xmi'/>
    <component class='org.openarchitectureware.emf.XmiReader'>
        <modelFile value='$model'/>
        <outputSlot value="model"/>
    </component>
    <component class='org.openarchitectureware.XPand2.Generator'>
        <eXPand value="JavaDomain::JavaDomain FOR model"/>
        <genPath value='$genPath'/>
    </component>
</workflow>
```

Gliederung

XPand - eine Templatesprache

- Generierung von Text-Dateien aus beliebigen Modellen; z.B.:
  - Quellcode-Dateien (z.B. Java), welche Grundfunktionalität zur Verfügung stellen
  - XML-Konfigurationsdateien

Beispiel

```
≪DEFINE JavaDomain FOR Domain≫
    ≪EXPand JavaEntity FOREACH entities≫
≪ENDDEFINE≫

≪DEFINE JavaEntity FOR Entity≫
≪FILE "mein/paket/fuer/entitaeten/" + this.name + ".java"≫
package mein.paket.fuer.entitaeten;
class ≪this.name≫ {
≪EXPand JavaAttribute FOREACH attributes≫
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE JavaAttribute FOR Attribute≫
    private ≪this.type≫ ≪this.name≫;
≪ENDDEFINE≫
```

XPand - eine Templatesprache

- Generierung von Text-Dateien aus beliebigen Modellen; z.B.:
  - Quellcode-Dateien (z.B. Java), welche Grundfunktionalität zur Verfügung stellen
  - XML-Konfigurationsdateien

Datei "src-gen/mein/paket/fuer/entitaeten/Kunde.java"

```
package mein.paket.fuer.entitaeten;
class Kunde {
    private Integer KundenNr;
    private String Name;
}
```

Datei "src-gen/mein/paket/fuer/entitaeten/Konto.java"

```
package mein.paket.fuer.entitaeten;
class Konto {
    private Integer KontoNr;
    private Decimal Saldo;
}
```

Gliederung

Extend - Modell-Verarbeitung

- Modell-zu-Modell-Transformationen
- Informations-"Anreicherung" im Modell
- Erweiterbar mit Java-Klassen
- Hilfsfunktionen für die Verarbeitung von Daten mit dem XPand- oder Check-Framework

Beispiel

```
String javaClassName(Entity entity) :
  entity.name.toFirstUpper()
;
String javaFileName(Entity entity) :
  "mein/paket/fuer/entitaeten/"+javaClassName(entity)+".java"
;
```

Anwendung

```
«FILE "mein/paket/fuer/entitaeten/" + this.name + ".java"»
class «this.name» {
```

Extend - Modell-Verarbeitung

- Modell-zu-Modell-Transformationen
- Informations-"Anreicherung" im Modell
- Erweiterbar mit Java-Klassen
- Hilfsfunktionen für die Verarbeitung von Daten mit dem XPand- oder Check-Framework

Beispiel

```
String javaClassName(Entity entity) :
  entity.name.toFirstUpper()
;
String javaFileName(Entity entity) :
  "mein/paket/fuer/entitaeten/"+javaClassName(entity)+".java"
;
```

Anwendung

```
«FILE this.javaFileName»
class «this.javaClassName» {
```

Gliederung

Check - Modellvalidierung

- Spezielle Constraints, die nicht durch das getypte Metamodell abgedeckt werden.
- Ähnlich der Object Constraint Language (OCL)

Beispiel

```
context Entity
ERROR 'an entity must have a name!':
    this.name != null
;
context Attribute
ERROR 'an attribute of entity '+eContainer.name+' must have a name!':
    this.name != null
;
context Entity
ERROR 'the entity name ' + name + 'must be unique':
    ((Domain)eContainer).entities.name.collect(n|n==name).size==1
;
```

Gliederung

XText - Text-Verarbeitung

- Generierung von Modellen anhand einer textuellen Formatbeschreibung (Ähnlich EBNF)
- Import von beliebigen Datenquellen
  - CSV-Daten
  - Datenextraktion aus HTML

Modellbeschreibung

```
entity Kunde {
    Integer KundenNr;
    String Name;
}

entity Konto {
    Integer KontoNr;
    Decimal Saldo;
}
```

XText-Syntax

```
Entity :
    "entity" name=ID "{"
        (attributes+=Attribute)+
    "}"
Attribute :
    type=ID name=ID ";"
```

Gliederung

Zusammenfassung

openArchitectureWare ist:

- gut in Eclipse integriert
- erleichtert den Aufbau komplexer Softwaresysteme

- URL: http://www.eclipse.org/gmt/oaw/

- Diese Präsentation wurde mit wiki2beamer erstellt und steht unter der CreativeCommons by-nc-sa 2.0 Germany Lizenz (http://creativecommons.org/licenses/by-nc-sa/2.0/de/deed.de).