openArchitectureWare, Teil 2: Metamodellierung und Tool-Integration

Wenn Architektur zur Ware wird

VON MARKUS GUSKE UND KARSTEN THOMS

Im ersten Artikel dieser Serie [1] haben wir mit einem einfachen Beispielprojekt gezeigt, wie man mit dem openArchitectureWare (oAW) Generator [2] erste Generierungsergebnisse erzielt. Dabei sind wir auf die wichtigsten Konzepte eingegangen, ohne uns zu sehr in Details zu verlieren. In diesem Artikel zeigen wir, wie man eine Anwendungsarchitektur durch ein Metamodell beschreibt und daraus Metaklassen ableitet und wie diese in den Generatorschablonen (Templates) zur Generierung verwendet werden. Im Anschluss daran werden wir das Eclipse Plug-in für openArchitectureWare vorstellen und zeigen, wie es den Einsatz von oAW unterstützt.

In der letzten Ausgabe [1] haben wir bereits die Begriffe Metamodell und Domain Specific Language (DSL) eingeführt. Diese Begriffe wollen wir an dieser Stelle noch etwas genauer betrachten, da das Thema Metamodellierung einen wichtigen Eckpfeiler im MDSD-Prozess darstellt. Ein Metamodell ist ein Modell zur Beschreibung eines Modells [4]. Das einfache OOA-Modellbeispiel in Abbildung 1 enthält zwei Klassen, eine Beziehung und einige Attribute, die mithilfe der UML beschrieben wurden. Dass unser Modell aus diesen Elementen besteht, mag dem Leser gewöhnlich vorkommen. Dabei wurde hier bereits ein Metamodell verwendet, nämlich das Metamodell der UML selbst. Dieses beschreibt, dass Modelle Klassen enthalten, die wiederum Attribute enthalten und deren Klassen über Relationen in Beziehung zueinander stehen können. So wie die UML das Metamodell für unsere Modelle ist, gibt es ein Metamodell für die UML: MOF (Meta Object Facility), die „Mutter aller Modelle". Die OMG [7] hat MOF [8] als „Übermodell" entwickelt, welches die generelle Struktur und Darstellung von Metamodellen festlegt.

Das openArchitectureWare-Metamodell

Beim oAW-Metamodell handelt es sich um eine reduzierte, aber in der Praxis hinreichend umfassende Abbildung des UML-Metamodells. Die Abbildung 2 zeigt einen Ausschnitt daraus, in dem sich die Elemente *Class*, *Operation*, *Attribute*, *Association*, *AssociationEnd* etc. und ihre Beziehungen zueinander befinden. Neben den Modellelementen zur Abbildung statischer Modelle (also Klassendiagramme) enthält das oAW-Metamodell auch Metaklassen zur Abbildung dynamischer Modelle (Aktivitätsdiagramme und Zustandsdiagramme). Damit bietet openArchitectureWare die Möglichkeit, aus der Beschreibung dynamischer Aspekte eines Modells Code zu generieren. Einen Einstieg in dieses Thema findet sich in den Artikeln [9] und [10].

Von der Architektur zum Metamodell

Bei vielen Anwendungen wird man auf folgende oder ähnliche Architekturelemente treffen: Geschäftsobjekt/Entität, Geschäftsprozess, Schlüsselattribute, Suchmethoden/Finder usw. Diese Elemente stehen i.d.R. in folgender Beziehung zueinander: Geschäftsobjekte sind dadurch charakterisiert, dass sie eine Menge von Attributen besitzen, von denen eine Untermenge Schlüsselattribute sind. Über Schlüsselattribute lassen sich Instanzen eindeutig identifizieren. Um Instanzen von Geschäftsobjekten zu finden, gibt es neben „normalen" Methoden spezielle Suchme-

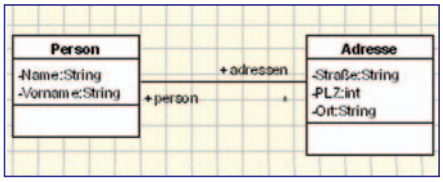

Abb. 1: Einfaches OOA-Beispiel

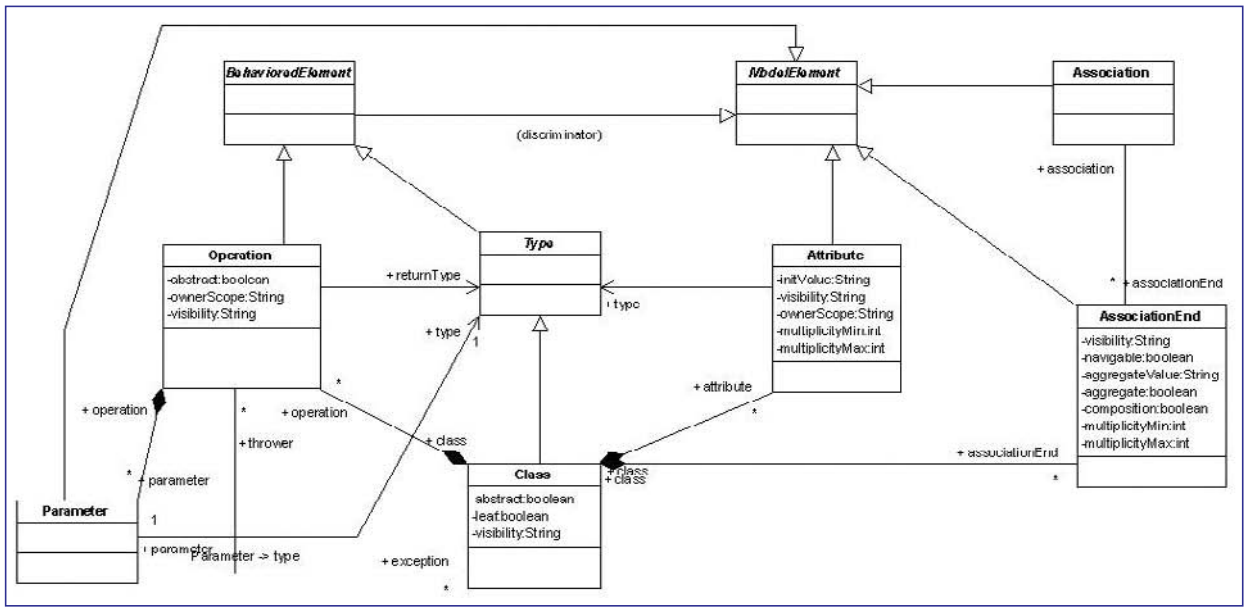

Abb. 2: Ausschnitt aus dem openArchitectureWare-Metamodell

Listing 1

Metaklassen-Geschäftsobjekt und Schlüssel

```
public class Geschaeftsobjekt extends Class {
  public ElementSet Suchmethode () {
    ElementSet result = new ElementSet();
    // alle Operationen des Typs Suchmethode suchen
    for (Iterator lIt=Operation().iterator();
                                    lIt.hasNext(); ) {
      Operation lOp = (Operation) lIt.next();
      if (lOp instanceof Suchmethode ) {
        result.add(lOp);
      }
    }
    return result;
  }

  public ElementSet Schluessel () {
    ElementSet result = new ElementSet();
    // alle Attribute des Typs Schluessel suchen
    for (Iterator lIt=Attribute().iterator();
                                    lIt.hasNext(); ) {
      Attribute lAttr = (Attribute) lIt.next();
      if (lAttr instanceof Schluessel) {
        result.add(lAttr);
      }
    }
    return result;
  }
}
////////////////////////////////////////////////
/////////////
public class Schluessel extends Attribute {
  // keine weitere Spezialisierung nötig
}
```

thoden. Zu guter Letzt werden Geschäftsobjekte in Geschäftsprozessen verwendet, welche die funktionalen Anforderungen an das System erbringen.

Wenn man den hier beschriebenen Ausschnitt aus der Anwendungsarchitektur durch ein UML-Klassendiagramm ausdrückt, dann findet das in Abbildung 3 beschriebene Metamodell Anwendung: Geschäftsobjekte und -prozesse sind Spezialisierungen von Klassen. Schlüsselattribute dagegen sind eine spezielle Art von Attributen und Suchmethoden sind besondere Operationen. Das Basis-Metamodell wird um Elemente unserer Anwendungsarchitektur erweitert und bildet damit die so genannte domänenspezifische Sprache (Domain Specific Language – DSL) ab. In Abbildung 4 sehen Sie eine mögliche Anwendung der DSL, d.h. ein Modell, welches auf dem von uns erweiterten oAW-Metamodell basiert.

Erstellung von Metaklassen

Damit der openArchitectureWare Generator die DSL „versteht" und man auf die Elemente zugreifen kann, müssen für die abgeleiteten Elemente des Metamodells so genannte Metaklassen erstellt werden. Im einfachsten Fall erben diese lediglich von einer der Basis-oAW-Metaklassen. Man wird normalerweise spezielle Methoden hinzuprogrammieren, die vor allem den Zugriff auf referenzierte Instanzmengen erlauben, wie z.B. alle Bezie-hungen von Elementen des Typs *Geschaeftsprozess* auf Elemente des Typs *Geschaeftsobjekt*. Listing 1 zeigt die Implementierung der beiden Metaklassen *Geschaeftsobjekt* und *Schluessel*. Aus den Genera-

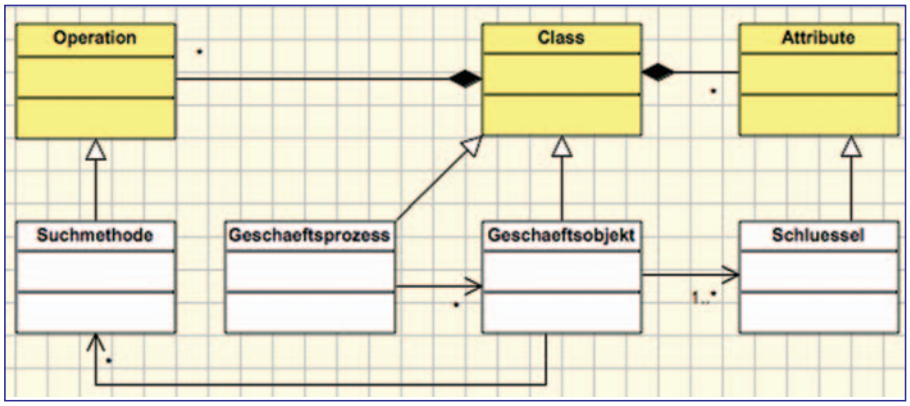

Abb. 3: Erweiterung des oAW-Basismetamodells

torschablonen heraus kann dann auf diese Hilfsmethoden zugegriffen werden. Beim openArchitectureWare Generator werden diese Schablonen in der Sprache Xpand erstellt [1]. Mit dem in Abbildung 5 gezeigten Xpand-Listing wird aus dem Beispielmodell eine Textdatei generiert (Listing 2).

Metamodell Generator

Der Aufbau von Metaklassen ist weitgehend schematisch. Meist wird per API über Instanzen von Metaklassen navigiert oder Mengen von Metaklassen werden aggregiert bzw. gefiltert, um diese in den Templates verwenden zu können. Was liegt also näher, als diesen schematischen Code ebenfalls zu generieren? Zu diesem Zweck gibt es das openArchitectureWare-Unterprojekt Metamodell Generator. Dieser generiert aus einem wie in Abbildung 3 beschriebenen Metamodell die zu erzeugenden Metaklassen. Was einem direkt abgenommen wird, ist die Überprüfung grundlegender Constraints wie Kardinalitätsprüfung bei Assoziation oder erforderliche Tagged Values. Diese Constraints werden direkt in die *CheckConstraints()*-Methode mit hineingeneriert. Da es dem Anwender einiges an einfachem und schematischem Implementierungsaufwand abnimmt, empfehlen wir, sich näher mit dem Metamodell Generator zu beschäftigen.

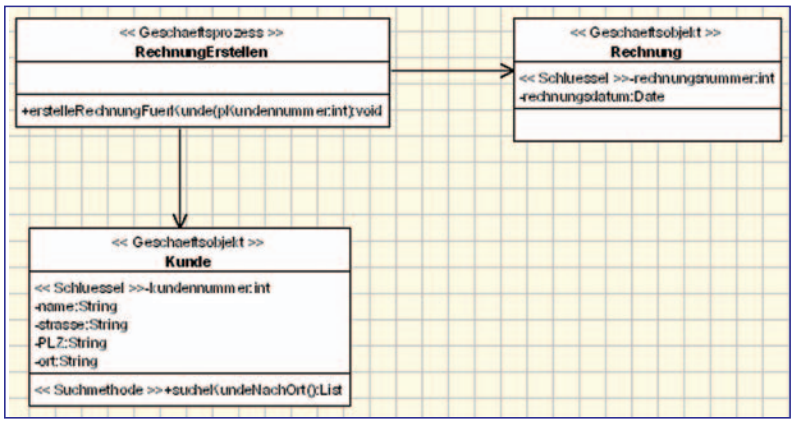

Abb. 4: Beispielmodell unter Anwendung der DSL

Eclipse-Integration

Wie einleitend erwähnt, geben wir nun einen Überblick über die Integration von openArchitectureWare in Eclipse über das eigene Plug-in. Das Plug-in kann von der oAW-SourceForge-Seite [3] heruntergeladen werden und bietet folgende Unterstützung bei der Verwendung von openArchitectureWare:

- Xpand Template Editor
- Content Outline Page
- Analyse von Generatorausgaben

Xpand Template Editor

Mit dem Template Editor könen Xpand-Template-Dateien einfacher editiert werden. Die Dateierweiterung *.tpl* ist durch

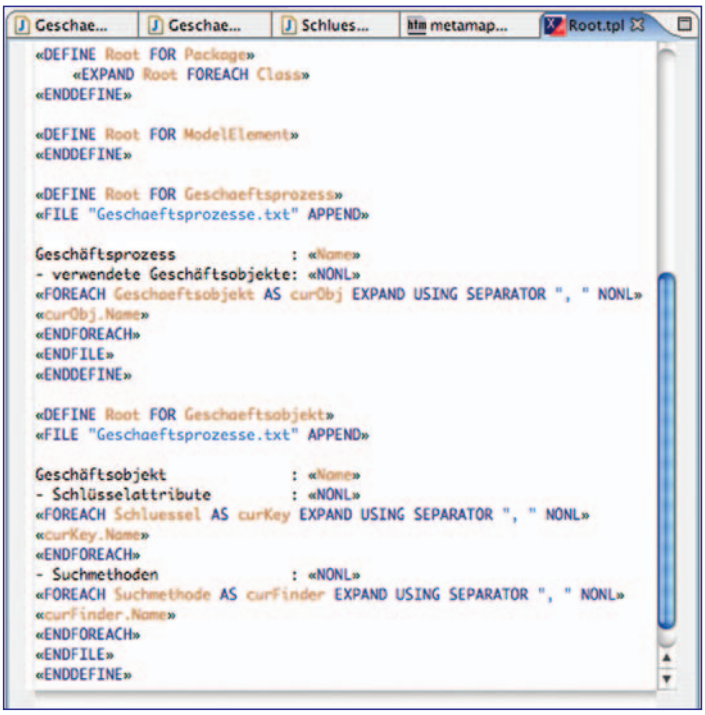

Abb. 5: Xpand Template Editor

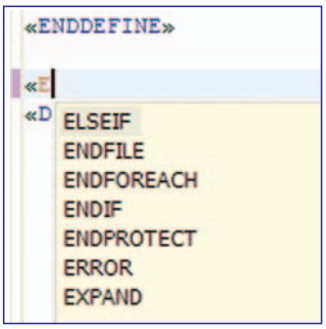

Abb. 6: Code Completion für Xpand

das Plug-in mit dem Template Editor verknüpft, sodass Dateien mit dieser Endung automatisch mit diesem Editor geöffnet werden.

Der Template Editor bietet Syntax-Highlighting und Code Completion für die Xpand-Sprache. Durch das Syntax-Highlighting lassen sich Xpand Templates einfacher lesen. Xpand-Code und zu generierender Code sind klar zu unterscheiden (Abb. 5). Code Completion für Xpand Schlüsselwörter erhält man wie gewohnt über die Tastenkombination Strg + Space (Abb. 6).

Eine wichtige Funktion des Template Editor sind festgelegte Tastenkombinationen für die französischen Anführungszeichen « und », welche als Klammerungszeichen verwendet werden. Diese Zeichen sind auf einer Standardtastatur (bis auf die Eingabe des Ascii-Codes über Alt oder auf Macs) nicht verfügbar. Die beiden Zeichen lassen sich einfach über die Tastenkombination Strg + < und Strg + > eingeben.

Content Outline Page

Das oAW-Eclipse-Plug-in enthält eine *ContentOutlinePage*-Implementierung, die in der Eclipse-Standard-View *Outline* die Struktur von Xpand-Dateien anzeigt. Für Xpand-Dateien werden in der Outline View alle *DEFINE*-Blöcke aufgelistet. Durch einen Klick auf einen der Einträge springt der Template-Editor an den Beginn der Definition. Gerade für längere Xpand-Dateien kann dies ein recht nützliches Feature sein.

Analyse von Generator-Ausgaben

Über den Ant-Task <*generate*> kann während der Generierung ein so genannter Dump File erzeugt werden. In dieser Datei werden während des Generierungslaufs Informationen, z.B. über die instanziierten Metaklassen und die verwendeten Templates, gespeichert. Hierzu wird für den Ant-Task das Attribut *dumpfile* mit dem Namen des zu erzeugenden Files gesetzt. Um die erzeugte Datei anschließend zu analysieren, selektiert man die Datei in der Navigator View und wählt aus dem Kontextmenü openArchitectureWare | Browse Dump File. Anschließend können die Informationen durch die im Folgenden besprochenen Views analysiert werden. Die Views können über Window | Show View | Other geöffnet werden. In dem Show View-Dialog sind sie unter der Kategorie openArchitectureWare eingruppiert. Die gezeigten Abbildungen sind dem Beispielprojekt *ExampleWorkspace* [6] entnommen.

View	Beschreibung
Generator Messages	Während des Generatorlaufs erzeugte Meldungen, wie z.B. Constraint-Verletzungen
File Access	Zuordnung von Template-Definitionen zu den Files, zu deren Erzeugung sie beigetragen haben
Model Structure	Zeigt die Struktur des instanziierten Modells sowie Eigenschaften der Modellelemente
Property Access	Zeigt den Zugriff auf Properties von Metaklassen durch Templates
Template Call Graph	Übersicht über die aufgerufenen Template-Definitionen mit den Namen der Elemente, für die sie expandiert wurden
Template Browser	Anzeige der Struktur der Template-Dateien und -Definitionen

Tabelle 1: Views des openArchitectureWare-Eclipse-Plug-in

Listing 2

Generierungsergebnis

```
Geschäftsobjekt       : Kunde
- Schlüsselattribute  : kundennummer
- Suchmethoden        : sucheKundeNachOrt

Geschäftsobjekt       : Rechnung
- Schlüsselattribute  : rechnungsnummer
- Suchmethoden        :

Geschäftsprozess      : RechnungErstellen
- verwendete Geschäftsobjekte: Rechnung, Kunde
```

Mit der Model Structure View (Abb. 7) kann man durch das instanziierte Modell browsen. In einer Baumansicht sind die Instanzen der Metaklassen aufgeführt. Im rechten Teil der View werden die Namen und Wertepaare des selektierten Eintrags tabellarisch aufgelistet. Man hat hier die Möglichkeit, das Modell zu betrachten, das dem Generator als Input dient. Die Template Call Graph View (Abb. 8) zeigt, welche Template-Definitionen mit entsprechenden Modellelementen expandiert wurden. Die Knoten der Baumansicht haben dabei das Format

<Templatedatei-Pfad>::<Definitionsname>
WITH <VollQualifizierterElementname>

Man sieht, dass Definitionen wiederum andere Definitionen aufrufen sowie bestimmte Expandierungen für Elementmengen wiederholt werden. Zusammen mit der Model Structure View und den Templates lässt sich nachvollziehen, was der Generator mit welchen Modellelementen ausführt. Das Plug-in enthält noch weitere Views, auf die wir im Rahmen dieses Artikels nicht im Detail eingehen. Wir haben sie in Tabelle 1 kurz zusammengefasst.

Weitere Integrationsmöglichkeiten

Die Einbettung von openArchitectureWare beschränkt sich nicht nur auf Eclipse. Es gibt bereits Projekte, die z.B. eine Integration in UML-Werkzeuge vornehmen. Im einfachsten Fall ermöglichen sie den Aufruf des Generators aus dem Tool heraus. Weitergehende Integrationen wie z.B. itemis CM3 [5] erzeugen aus der Beschreibung des Metamodells Plug-ins für UML-Werkzeuge. Diese Plug-ins erweitern die Tools z.B. um spezifisch auf die verwendeten Stereotypen zugeschnittene Property Sheets. Außerdem werden Verletzungen der Modellintegrität direkt in das UML-Tool zurückgemeldet.

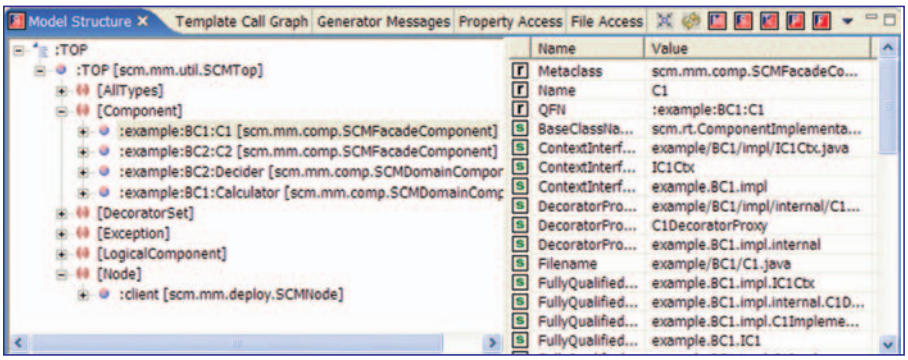

Abb. 7: Model Structure Browser

Zusammenfassung

Durch das Konzept der Metamodellierung und der Metaklassen zur Unterstützung von spezifischen Domänen lässt sich der openArchitectureWare Generator flexibel an die individuellen Bedürfnisse jedes Projekts anpassen. Eine Architekturbeschreibung sollte in jedem Projekt vorliegen, und wenn diese bereits als UML-Modell erstellt wurde, ist der Aufwand, das Metamodell für die DSL zu erstellen, sehr gering. Weitere Vorteile und Geschwindigkeit gewinnt man durch den Einsatz des openArchitectureWare-Unterprojekts Metamodell Generator, mit dem die Erstellung der notwendigen Metaklassen für die Template-Erstellung unterstützt wird.

In der zweiten Hälfte des Artikels haben wir die Eclipse-Integration unter die Lupe genommen. openArchitectureWare lässt sich einfach in jede IDE integrieren. Für Eclipse existiert ein eigenes Plug-in, welches zusätzliche Vorteile bringt. Auch in Zukunft wird Eclipse die bevorzugte IDE bleiben und die Entwicklung des Plug-ins aktiv vorangetrieben. Natürlich darf sich jeder beteiligen, um eine Integration in eine andere IDE beizutragen. Schließlich ist openArchitectureWare ein Open-Source-Projekt.

Der abschließende Teil der openArchitectureWare-Artikelserie wird zeigen, in welche Richtung sich openArchitectureWare bewegen wird. Derzeit gibt es einige interessante Vorhaben für das nächste Major Release, die wir nicht vorenthalten wollen. openArchitectureWare ist deutlich mehr als nur ein Generator!

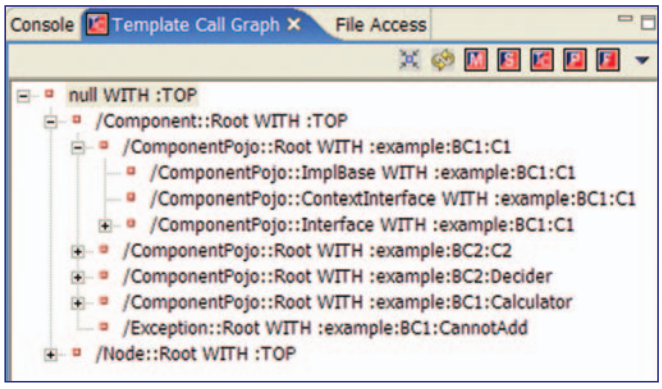

Abb. 8: Template Call Graph View

Markus Guske ist Senior-Berater bei einem IT-Dienstleister. Der Schwerpunkt seiner Arbeit liegt seit Jahren im Bereich der qualitätsgesicherten Anforderungsaufnahme und der modellgetriebenen Softwareentwicklung.

Karsten Thoms ist bei itemis als Senior-Berater tätig. Er beschäftigt sich schwerpunktmäßig mit modellgetriebenen Entwicklungsverfahren, der J2EE-Plattform und im Detail mit bekannten Java-Open-Source-Frameworks. Karsten Thoms ist aktiv an der Entwicklung des openArchitectureWare-Projekts beteiligt.

Links & Literatur

[1] Karsten Thoms, Boris Holzer: The Next Generation: Codegenerierung mit dem openArchitectureWare Generator 3.0, in *Java Magazin* 7.2005

[2] www.openarchitectureware.org

[3] sourceforge.net/projects/architecturware/

[4] Markus Völter, Thomas Stahl: Modellgetriebene Softwareentwicklung, dpunkt, 2005

[5] www.itemis.de

[6] Example Workspace: sourceforge.net/project/showfiles.php?group_id=88344

[7] www.omg.org/mda/

[8] MOF: www.omg.org/technology/documents/formal/mof.htm

[9] Tim Weilkiens, Bernd Oestereich, Thomas Stal: Vom Modell zum Code: Vom Geschäftsprozess zum Code – ein kurzer Weg mit MDA, in *Java Magazin* 9.2003

[10] Martin Schepe, Wolfgang Neuhaus, Peter Roßbach: Model Driven Architecture: Grundlegende Konzepte und Einordnung der Model Driven Architecture (MDA), in *Java Magazin* 09.2003