Teil 2: Ein Praxisbeispiel

Modellvergleich mit EMF Compare

Quellcode auf CD!

>> ANDREAS MÜLDER, HOLGER SCHILL UND DR. LOTHAR WENDEHALS

Im zweiten Teil unserer Reihe zum Thema EMF Compare zeigen wir anhand eines Beispiels die praktische Anwendung des Frameworks, indem wir Schritt für Schritt einen Model-Change-Report-Generator erstellen.

Als Ergebnis eines Vergleichs zweier Modelle erzeugt das Framework EMF Compare ein Übereinstimmungs- und ein Differenzmodell. Zur automatisierten Weiterverarbeitung sind diese Modelle gut geeignet, da sie selbst wiederum EMF-Modelle sind. Im ersten Teil des Artikels wird beschrieben, wie man EMF Compare programmatisch verwendet, um ein Differenzmodell zu erzeugen. Zur Weiterverarbeitung von Modellen hat sich im Eclipse-Umfeld das Framework openArchitectureWare [1] etabliert. Mit diesem ist es leicht möglich, Modell-zu-Text(M2T)- bzw. Modell-zu-Modell-(M2M-)Transformationen durchzuführen. Im zweiten Teil des Artikels wird mittels einer Modell-zu-Text-Transformation gezeigt, wie die Lesbarkeit des Ergebnisses von EMF Compare gesteigert werden kann, indem aus dem Differenzmodell ein HTML-Report generiert wird.

Installation

Zum Ausführen unseres Praxisbeispiels wird Eclipse sowie openArchitectureWare und EMF Compare benötigt. Diese Features lassen sich bequem über die Update Site in eine bestehende Eclipse-Umgebung installieren. Alternativ steht eine Distribution unter [2] zum Download bereit, die alle nötigen Bestandteile enthält. Da im Artikel selbst der Sourcecode nur auszugsweise abgebildet ist, befindet sich das vollständige Projekt des Model-Change-Log-Generators auf der Heft-CD.

Das Beispielmodell

Als durchgängiges Beispiel dient ein einfaches Modell zur Verwaltung von Aufgaben. Abbildung 1 zeigt das Metamodell, das die Struktur der Modellinstanzen definiert.

Auch wenn der Nutzen des Modells durchaus infrage gestellt werden darf, ist es als Beispiel gut geeignet, denn es werden viele der gängigen Konzepte eines EMF-Modells angewendet.

Aufgaben, repräsentiert durch die Klasse *Task*, werden in einer *TaskList* zusammengefasst. Ein einzelner *Task* besteht aus verschiedenen Attributen (*name*, *priority*, *finished* und *lastActive*). Außerdem hat ein *Task* eine One-to-many-Referenz auf weitere Tasks (*subTasks*) sowie auf Ressourcen (*resources*). Was aus Abbildung 1 nicht ersichtlich ist, sind die beiden unterschiedlichen Referenztypen, die hier verwendet werden. Während *resources* als Komposition modelliert ist, handelt es sich bei *subTasks* um eine Assoziation. In EMF wird die Art der Referenz durch das Setzen des Attributs *containment* bestimmt. Der Wert *true* definiert eine Komposition, wohingegen *false* eine Assoziation beschreibt. Dieser Sachverhalt ist im Kontext der weiteren Betrachtung wichtig, da Änderungen abhängig vom Referenztyp unterschiedlich behandelt werden.

Programmatisches Erzeugen des Differenzmodells

Um ein Differenzmodell zu erzeugen, benötigen wir zunächst zwei Modellinstanzen aus dem zuvor beschriebenen Metamodell, die wir miteinander vergleichen wollen. Um diese zu erstellen, gibt es unterschiedliche Wege. Häufig wird zunächst mithilfe von EMF aus dem Metamodell der Modellcode generiert, um anschließend programmatisch Modelle zu erzeugen. Diese können dann im XMI-Format serialisiert werden. Alternativ kann auch ein grafischer Editor aus dem Metamodell generiert werden. In unserem Fall nutzen wir die dynamische Art der Modellinstanziierung. Das hat den Vorteil, dass das Generieren von statischem Quelltext nicht notwendig ist. Um Modellinstanzen aus einem Metamodell zu erzeugen, sind folgende Schritte nötig:

- Das Metamodell im *Package Explorer* selektieren
- Im Kontextmenü OPEN WITH SAMPLE REFLECTIVE ECORE MODEL EDITOR auswählen
- Das Root-Element (in unserem Fall *TaskList*) auswählen
- Im Kontextmenü CREATE DYNAMIC INSTANCE wählen

Der Editor nutzt das EMF-Reflection-API und erlaubt so das dynamische Erzeugen von Modellen. Abbildung 2 zeigt die beiden Beispielmodelle, die wir nachfolgend miteinander vergleichen werden. Um nun das Differenzmodell zu

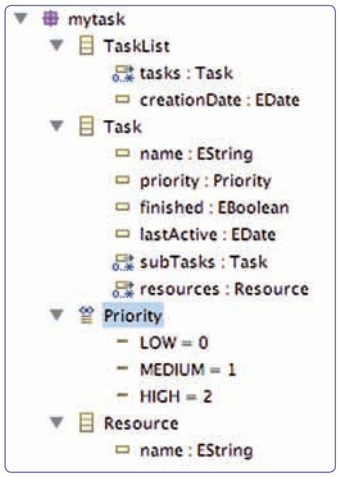

Abb. 1: Das Metamodell des Aufgabenbeispiels

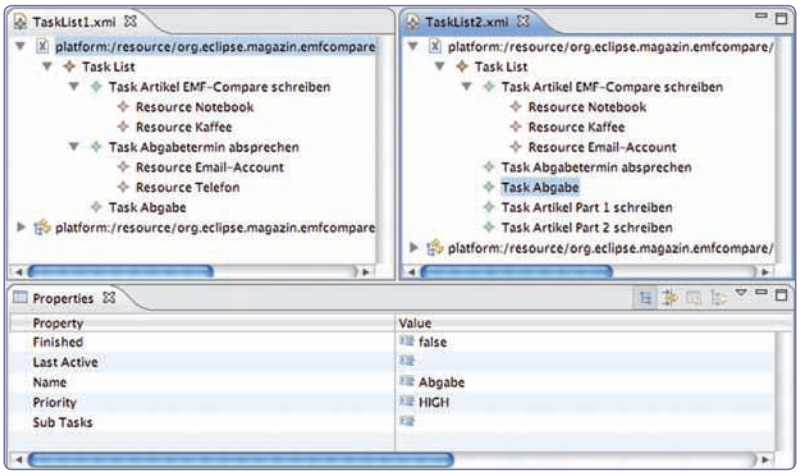

Abb. 2: Zwei Modellinstanzen im Sample-Reflective-Ecore-Model-Editor

erzeugen, verwenden wir den Code aus Listing 1.

Zunächst laden wir das Metamodell unseres Beispiels und fügen es der *PackageRegistry* des *ResourceSets* hinzu (Zeilen 1-5). In EMF ist ein *ResourceSet* eine Verwaltungseinheit von einer Menge von Ressourcen. Unter anderem können so Änderungen registriert und gegebenenfalls verfolgt werden. Des Weiteren werden die zu vergleichenden Modelle geladen und dem *ResourceSet* hinzugefügt. (Zeilen 7-11). Mittels *MatchOptions* ist es möglich, das Verhalten von EMF Compare zu beeinflussen. In unserem Fall setzen wir die *MatchOption* OPTION_DISTINCT_METAMODEL auf *false*. So informieren wir den *MatchService* darüber, dass beide Modelle auf demselben Metamodell aufbauen. Das spart einige Checks und erhöht so die Performanz. Tabelle 1 gibt einen Überblick über weitere *MatchOptions*.

Im nächsten Abschnitt wird das *MatchModel* erzeugt (Zeile 18). Wie bereits im ersten Artikel der Serie ausführlich erläutert [3], besteht das *MatchModel* aus einem Mapping von übereinstimmenden Elementen sowie einer Menge von Elementen ohne Übereinstimmung.

Dieses wird anschließend dazu verwendet, das *DiffModel* zu erstellen (Zeile 20), das die Details zu den Modellunterschieden beinhaltet. Da das *DiffModel* selbst ein EMF-Modell ist, können wir es abschließend im XMI-Format serialisieren (Zeile 22).

Den generischen Differenzalgorithmus anpassen

Öffnen wir nun das im vorherigen Abschnitt erzeugte Differenzmodell mit dem EMF-Editor unserer Wahl, bekommen wir einen vollständigen Überblick über alle Modelländerungen. Abbildung 3 zeigt das Ergebnis mit den beiden Beispielmodellen aus Abbildung 2.

Im gesamten Modell gibt es neun Änderungen. Vier davon beziehen sich auf den Task *Artikel EMF-Compare schreiben*, wovon zwei das Ändern eines Attributwerts angeben und zwei das Hinzufügen von Referenzen. Außerdem wurde die Ressource *Telefon* vom Task *Abgabetermin absprechen* entfernt. Zusätzlich wurden dem zweiten Modell zwei weitere Tasks hinzugefügt, nämlich *Artikel Part 1 schreiben* und *Artikel Part 2 schreiben*.

Der *DiffService* von EMF Compare verwendet standardmäßig den generischen Differenzalgorithmus, dessen Funktionsweise im vorherigen Artikel [3] beschrieben wurde. So kann jedes beliebige Modell unabhängig vom zugrunde liegenden Metamodell verarbeitet werden, und man erhält einen vollständigen Überblick über alle Modelländerungen. Das ist jedoch nicht immer zielführend, da die Semantik der verschiedenen Modellelemente nicht berücksichtigt wird. In unserem Beispiel hat das Element *Task* das Attribut *lastActive*. Dieses gibt an, wann zuletzt an dieser Aufgabe gearbeitet wurde. Betrachtet man das Differenzmodell in Abbildung 3, sieht man, dass die Änderung des Attributwerts *lastActive* als Modelländerung mit aufgeführt ist. Das ist auch korrekt, für unseren Model-Change-Report ist die Änderung dieses Werts jedoch irrelevant. Die *GenericDiffEngine*, die die Unterschiede berechnet, lässt sich jedoch leicht an solche Spezialfälle anpassen. Der nachfolgende Quelltext zeigt, wie das Attribut *lastActive* beim Aufbau des Differenzmodells ignoriert werden kann:

```
1.  public class TaskDiffEngine extends GenericDiffEngine {
2.
3.  @Override
4.  protected boolean shouldBeIgnored(EAttribute
                                          attribute) {
5.  if ("lastActive".equals(attribute.getName()))
6.  return true;
7.
8.  return super.shouldBeIgnored(attribute);
9.  }
10. }
```

Zunächst erstellen wir eine neue Klasse *TaskDiffEngine*, die von der Klasse *GenericDiffEngine* erbt. Diese bietet einige *protected*-Methoden an, die von der Subklasse überschrieben werden können. Für unser Beispiel überschreiben wir die Methode *shouldbeIgnored*, um so das Modellelement *lastActive* beim

MatchOption	Bedeutung
OPTION_DISTINCT_METAMODEL	Gibt an, ob die zu vergleichenden Modelle ein gemeinsames Metamodell haben. Das erhöht die Geschwindigkeit beim Aufbau des *MatchModels*. Default: *false*.
OPTION_IGNORE_ID	Gibt an, ob die *functional*-ID ignoriert werden soll.
OPTION_IGNORE_XMI_ID	Gibt an, ob XMI-IDs ignoriert werden sollen. Default: *false*.
OPTION_SEARCH_WINDOW	Bestimmt die Suchtiefe beim Modellelementvergleich. Ein hoher Wert erhöht die Genauigkeit des Vergleichs, niedrige Werte erhöhen die Performanz. Default: *100*.

Tabelle 1: Einige MatchOptions im Überblick

Modellvergleich ignorieren zu können (Zeile 4-9). Abschließend müssen wir den Quelltext aus Listing 1 so anpassen, dass unsere *TaskDiffEngine* anstelle der *GenericDiffEngine* verwendet wird. Das geschieht dadurch, dass wir Zeile 20 durch die nachfolgenden beiden Codezeilen ersetzen:

```
TaskDiffEngine diffEngine = new TaskDiffEngine();
DiffModel customdiff = diffEngine.doDiff(match);
```

Führen wir den Code nun erneut aus, sehen wir, dass die Änderung des Attributwerts von *lastActive* im erzeugten Differenzmodell nicht mehr berücksichtig wird.

Model-2-Text mit openArchitectureWare

Die Transformation von Modellen in Modelle oder Text ist mittels der in openArchitectureWare vereinten Bestandteile eine leicht zu bewältigende Aufgabe. Die statisch typisierte Template-Sprache Xpand bietet die Möglichkeit, in Dateien zu schreiben. Templates werden dabei polymorph aufgerufen. Die kompakte Syntax in Kombination mit der Einbettung der funktionalen Sprache Xtend ermöglicht einen einfachen Umgang mit Modellen. Durch die komponentenbasierte Workflow Engine wird der Generatorablauf gesteuert. Hierbei werden einzelne Komponenten definiert und durch Dependency Injection konfiguriert.

Im ersten Schritt unseres Beispiels gilt es, zunächst eine Xpand-Datei mit der Dateiendung *xpt* anzulegen. Das Root-Element des Differenzmodells, das EMF Compare erzeugt, ist das *DiffModel*. Ein *Define-Block* (Zeilen 5-12) – in etwa vergleichbar mit einer Funktion – nimmt dieses Modellelement als Parameter entgegen und bildet den Einstiegspunkt des Generators:

```
1.  «IMPORT diff»
2.  «IMPORT ecore»
3.  «EXTENSION helper::helper»
4.
5.  «DEFINE main FOR DiffModel-»
6.  «FILE 'myResult.html'»
7.  <html>
8.  «EXPAND header-»
9.  «EXPAND body-»
10. </html>
11. «ENDFILE»
12. «ENDDEFINE»
```

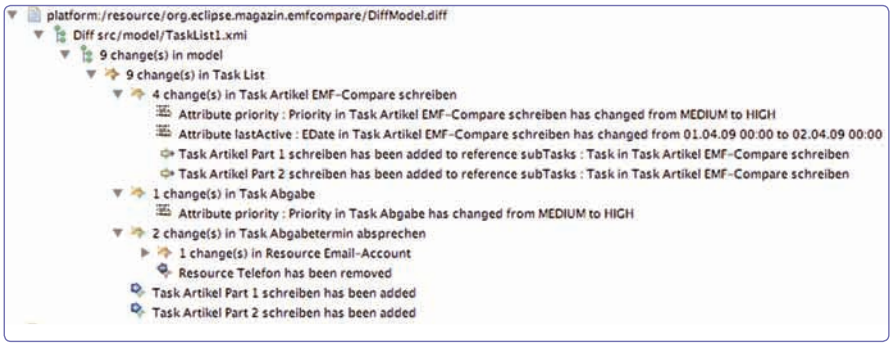

Abb. 3: Das erzeugte Differenzmodell

Durch das Importieren des Metamodells, des *DiffModels* und Ecore selbst (Zeilen 1–2) bietet der Editor automatische Codevervollständigung. Zwischen den Schlüsselwörtern *FILE* bzw. *ENDFILE*" ist das Schreiben in eine Datei möglich (Zeilen 6-11).

Da wir einen HTML-Report generieren wollen, hat die Datei die Endung *html* und Inhalte werden in HTML-Tags geschachtelt (Zeilen 7, 11). Xpand-Audrücke, umgeben von französischen Anführungszeichen, werden dabei vom Generator ausgewertet, während einfache Textbausteine wie <*html*> direkt in die Ausgabedatei übernommen werden. Um die Templates strukturieren zu kön-

Listing 1

Programmatisches Erzeugen des Diffmodels

```
1.  //Load the metamodel
2.  EPackage taskPackage = EcoreUtil2
3.  .getEPackage("src/metamodel/mytasks.ecore");
4.  ResourceSet resourceSet = new Resource
                                        SetImpl();
5.  resourceSet.getPackageRegistry().put(null, task
                                        Package);
6.
7.  // Load the two models to compare with each
                                        other
8.  EObject tasklist1 = ModelUtils.load(
9.  new File("src/model/tasklist1.xmi"),
                                        resourceSet);
10. EObject tasklist2 = ModelUtils.load(
11. new File("src/model/tasklist2.xmi"),
                                        resourceSet);
12.
13. // Set the compare options
14. Map<String, Object> options = new
                                HashMap<String, Object>();
15. options.put(MatchOptions.OPTION_DISTINCT_
                                METAMODELS, false);
16.
17. // build the matchmodel
18. MatchModel match = MatchService.
                        doMatch(tasklist1, tasklist2, options);
19. // build the diffmodell based on the matchmodel
20. DiffModel genDiff = DiffService.doDiff(match,
                                        false);
21. //Serialize to XMI
22. ModelUtils.save(genDiff, „DiffModel.diff");
```

Listing 2

Ein Workflow zum Ausführen des Report-Generators

```
1.  <workflow>
2.  <property name="src-gen" value="src-gen" />
3.  <property name="leftModelFile" value="src/model/
                                        TaskList1.xmi" />
4.  <property name="rightModelFile" value="src/
                                        model/TaskList2.xmi" />
5.  <property name="customDiffEngine" value="main.
                                        TaskDiffEngine" />
6.
7.  <!-- set up EMF for standalone execution -->
8.  <bean class="org.eclipse.mwe.emf.Standalone
                                        Setup">
9.  <registerGeneratedEPackage
10. value="org.eclipse.emf.compare.diff.metamodel.
                                        DiffPackage"/>
11. <registerEcoreFile value="src/metamodel/mytasks.
                                        ecore" />
12. <platformUri value=".." />
13. </bean>
14.
15. <component class="main.DiffmodelCreator">
16. <leftModelFile value="${leftModelFile}" />
17. <rightModelFile value="${rightModelFile}" />
18. <customDiffEngine value="${customDiffEngine}"
                                        />
19. <modelSlot value="model" />
20. </component>
21.
22. <!-- generate text from model -->
23. <component class="org.openarchitectureware.
                                        xpand2.Generator">
24.  <metaModel id='mm' class='org.eclipse.m2t.type.
                                        emf.EmfRegistryMetaModel'/>
25. <expand
26.  value="template::Main::main FOR model" />
27. <outlet path="${src-gen}" >
28. </outlet>
29. </component>
30. </workflow>
```

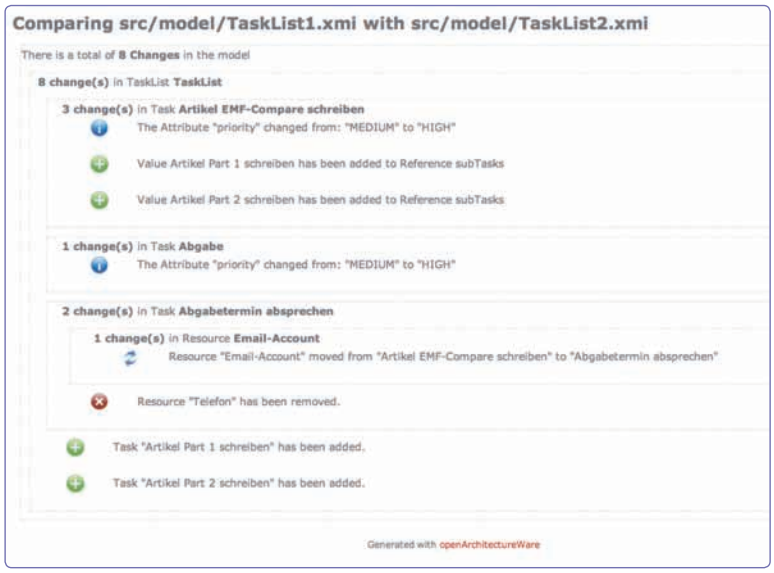

Abb. 4: HTML-basierter Report als Ergebnis des Generators

nen, kann man aus einem Define-Block heraus weitere Define-Blöcke aufrufen. Der Define-Block *header* (Zeile 8) generiert eine Zusammenfassung in die Datei, während *body* die einzelnen Modelländerungen verarbeitet (Zeile 9). Zusätzlich besteht die Möglichkeit, komplexe oder wiederkehrende Operationen in Extensions auszulagern, um so die Templates übersichtlich zu gestalten. Hierzu bietet openArchitectureWare die ausdrucksstarke Sprache Xtend, die durch Konzepte, z. B. *Closures* das Navigieren durch komplexe Baumstrukturen erheblich vereinfacht. Der Model-Change-Log-Generator, der der Heft-CD beiliegt, verwendet diese Sprache, um generisch die Werte von Attributen auszulesen.

Neben dem HTML-Code, der für die Schachtelung der Hierarchietiefe nötig ist, ruft der Define-Block *body* für jedes *DiffElement* einen Define-Block namens *owned* auf. Das Metamodell des Diff-Models definiert die Art der Modelländerung über eine Vererbungshierarchie. Das ermöglicht uns die Verwendung von Polymorphie, indem wir die Define-Blöcke überladen, sodass zur Laufzeit des Generators abhängig vom Typ des *ModelElements* der zutreffende Block ausgewählt wird. Der nachfolgende Block zeigt einen Ausschnitt für die beiden *DiffElements AddModelElement* und *UpdateAttribute*.

```
1. «DEFINE owned FOR AddModelElement»
2. «this.rightElement.metaName()» "«this.
   rightElement.getNameValue()»"
3. has been added.
4. «ENDDEFINE»
5.
6. «DEFINE owned FOR UpdateAttribute»
7. The Attribute "«this.attribute.name»" changed from:
8. «this.leftElement.getValue(this.attribute.name)»" to
9. «this.rightElement.getValue(this.attribute.name)»"
10. «ENDDEFINE»
```

Der erste Block (Zeilen 1-4) behandelt den Fall, dass ein Modellelement hinzugefügt wurde. Der zweite Block wird aufgerufen, wenn sich der Wert eines Attributs geändert hat. In EMF Compare wird zu diesem Zweck die Betrachtung in zwei Seiten geteilt. Die linke zeigt das Ausgangsmodell, die rechte hingegen das geänderte. Aus diesem Grund existieren in Elementen des *DiffModel* ebenfalls zwei Seiten bzw. Referenzen auf Elemente der jeweiligen Modelle, auf die über *rightElement* und *leftElement* zugegriffen werden kann (Zeilen 2, 8-9).

Um das Template ausführen zu können, definieren wir zunächst einen Workflow (Listing 2). Es gibt dabei unterschiedliche Möglichkeiten, das *DiffModel* zu erzeugen bzw. zu lesen. Zum einen ist es möglich, ein zuvor persistiertes *DiffModel* einzulesen und dann weiter zu verarbeiten. Zum anderen ist durch die Implementierung einer eigenen Workflow-Komponente die Erzeugung des *DiffModel* ohne Persistierung möglich. Dieses Ergebnis ist im Workflow nutzbar.

Im Workflow definieren wir zunächst die Eingangsmodelle sowie den Ausgabeordner und unsere *DiffEngine*, die das Attribut *lastActive* ignoriert (Zeilen 2-5). Im Weiteren werden die notwendigen Metamodelle registriert (Zeilen 7-13). Die Workflow-Komponente *main.DiffmodelCreator* erhält als Eingabe zwei Modelle sowie optional die angegebene *DiffEngine*. Das Ergebnis ist ein *DiffModel*, das wie oben beschrieben erstellt wird. In diesem Fall wird jedoch das entstandene Modell nicht persistiert wie in Listing 1 (Zeile 22), sondern direkt mittels eines Execution Contexts (*ctx*) in den Workflow zurückgegeben:

```
1. DiffModel customdiff = customDiffEngine.
   doDiff(match, false);
2. ctx.set(getModelSlot(),customdiff);
```

Die Generatorkomponente (Zeilen 23-30 des Workflow-Beispiels in Listing 2) kann sich dessen bedienen und an unser Template übergeben (Zeile 26). Durch diese Art der Erzeugung des *DiffModels* ist dieses stets aktuell und erfordert keinen vorherigen, manuellen Schritt.

Das Endergebnis des Generators ist eine hierarchisch aufgebaute Webseite, die die Unterschiede zwischen zwei Modellen übersichtlich und lesbar darstellt. Abbildung 4 zeigt schließlich den generierten HTML-Report.

Andreas Mülder arbeitet als Softwarearchitekt bei der itemis AG in Lünen. Seine Schwerpunkte liegen in der modellgetriebenen Softwareentwicklung und Werkzeugketten mit Eclipse. Kontakt: Andreas.Muelder@itemis.de.

Holger Schill ist Softwarearchitekt im Standort Kiel der itemis AG. Seine Schwerpunkte liegen dabei in der modellgetriebenen Softwareentwicklung und der Erstellung EMF-basierter, textueller und grafischer Editoren für individuelle DSLs. Kontakt: Holger.Schill@itemis.de.

Dr. Lothar Wendehals ist Berater und Softwarearchitekt bei der itemis AG. Er hat langjährige Erfahrungen im Bereich Reverse Engineering und modellbasierte Entwicklungsumgebungen. Kontakt: Lothar.Wendehals@itemis.de.

>> Links & Literatur

[1] http://www.openarchitectureware.de

[2] http://oaw.itemis.com/openarchitectureware/language=en/2837/downloads

[3] Mülder, Andreas; Schill, Holger; Dr. Wendehals, Lothar: „Modellvergleich mit EMF Compare. Funktionsweise des Frameworks", in Eclipse Magazin 4.09, S. 43