

**Plus CD!**

Stimmen zu EclipseCon, JAX und Eclipse Forum Europe >> 10

4.09

Deutschland € 9,80  
Österreich € 10,80, Schweiz CHF 19,20



**eclipse**  
MAGAZIN

# eclipse

## MAGAZIN

[www.eclipse-magazin.de](http://www.eclipse-magazin.de)

### CD-INHALT

#### Exklusiver Buchauszug:

„Eclipse Web Tools Platform“  
von Kai Brüssau, Kapitel 3:  
Java Servlets **entwickler.press**



#### Chord Scale Generator

Eclipse-Tool für Saiteninstrumente

#### Eclipse-Tools

Java Workflow Tooling (JWT) 0.5.0,  
Bonita 4.1, FraSCAti 0.5

#### eDine

Innovatives Restaurant-  
Management

#### WTP

Dali JPA Tools, Ajax  
Toolkit Framework

# WEB TOOLS

# PLATTFORM

## Modellierung zur Laufzeit >> 31

CDO Model Repository im Einsatz

## Eclipse versus Maven >> 68

Kampf der Builder

## Eclipse RCP für Musiker >> 54

Musiktool Chord Scale Generator

## Java Workflow Tooling (JWT) >> 89

Vom abstrakten Geschäftsmodell zum lauffähigen Code

## RCP-Tests mit SWTBot >> 75

Eclipse-Magazin-Tutorial



0 4

4 196603 209806

D 68864

Datenträger enthält  
Info- und  
Lehrprogramme  
gemäß §14 JuSchG





## Teil 1: Funktionsweise des Frameworks

# Modellvergleich mit EMF Compare

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

---

Das Vergleichen und Zusammenführen EMF-basierter Modelle wurde bisher nur auf textueller Ebene im XML-Quelltext unterstützt. Bei der Entwicklung im Team führt das zu Problemen, da parallele Änderungen an Modellen so nur schwer zusammenzuführen sind. In diesem Artikel wird das EMF-Compare-Framework vorgestellt, das für den Vergleich die logische Struktur der Modelle heranzieht. Darauf aufbauend unterstützt ein Vergleichswerkzeug den Benutzer beim Zusammenführen unterschiedlicher Modelle

---

Wer modellbasiert Software entwickelt und sich im Eclipse-Umfeld bewegt, kommt um das Eclipse Modeling Framework (EMF) [1] nicht herum. Zur Serialisierung der mit EMF erstellten Modelle wird das Austauschformat XMI verwendet. Es ist nicht nur durch die Object Management Group (OMG) standardisiert, sondern bietet auch die bekannten Vorteile eines XML-Formats, wie verschiedene Möglichkeiten zur textuellen Bearbeitung. Zu diesen Möglichkeiten zählt auch der Vergleich zweier Dateien. Da die Modelle in einer textuellen Form serialisiert werden, können die üblichen textbasierten Vergleichswerkzeuge verwendet werden. Auch in Eclipse steht solch ein Vergleichswerkzeug zur Verfügung (Abb. 1). Ein Nachteil dieser Werkzeuge ist allerdings, dass sie meist weder die Syntax noch die Semantik der Texte kennen und sie nur zeilenweise vergleichen.

Diese Art des Vergleichs ist für Modelle jedoch ungeeignet. Werden zum Beispiel die Elemente zweier ansonsten identischer Modelle in unterschiedlicher Reihenfolge serialisiert, so zeigen diese Vergleichswerkzeuge irrtümlicherweise Unterschiede an. Außerdem sind XMI-Dateien, und damit auch die textuellen Vergleiche zwischen zwei XMI-Dateien, nicht geeignet, von Menschen gelesen zu werden. Das 2006 gestartete Projekt EMF Compare [2], [3], [4] berücksichtigt hingegen beim Vergleich die Elemente und die Strukturen der Modelle. Die Ergebnisse des Vergleichs werden mithilfe von Bäumen dargestellt, in denen geänderte, hinzugefügte oder gelöschte Modellelemente visualisiert werden. Durch die Integration in Eclipse wird sowohl beim manuellen Vergleich als auch beim Vergleich unterschiedlicher Versionen eines EMF-Modells aus einem Versionierungssys-

tem automatisch EMF Compare verwendet.

### Funktionsweise des Vergleichsalgorithmus

Der Vergleichsalgorithmus von EMF Compare ist generisch und kann dadurch beliebige EMF-Modelle miteinander verarbeiten. Der Algorithmus ist in zwei Phasen unterteilt (Abb. 2). Zunächst werden Übereinstimmungen unter Zuhilfenahme von verschiedenen Metriken gesucht und ein *Matchmodel* erstellt. Basierend auf diesem Übereinstimmungsmodell wird dann das Differenzmodell erstellt, das detaillierte Informationen über die Unterschiede der verglichenen Modelle enthält. Im Folgenden werden wir zunächst die Algorithmen zur Identifizierung der Übereinstimmungen und zum Bilden der Differenz erläutern. Des Weiteren werden wir zeigen, wie der Algorithmus zum Vergleich von drei Modellen funktioniert und wie die gefundenen Differenzen dazu verwendet werden, unterscheidende Modelle wieder zu einem einzigen Modell zusammenzuführen.

### Identifikation der Übereinstimmungen

In der ersten Phase werden die Elemente beider Modelle für die Suche nach Übereinstimmungen synchron durchlaufen. Die beiden Modelle werden „linkes Modell“ und „rechtes Modell“ genannt, das linke gilt als das Originalmodell, das

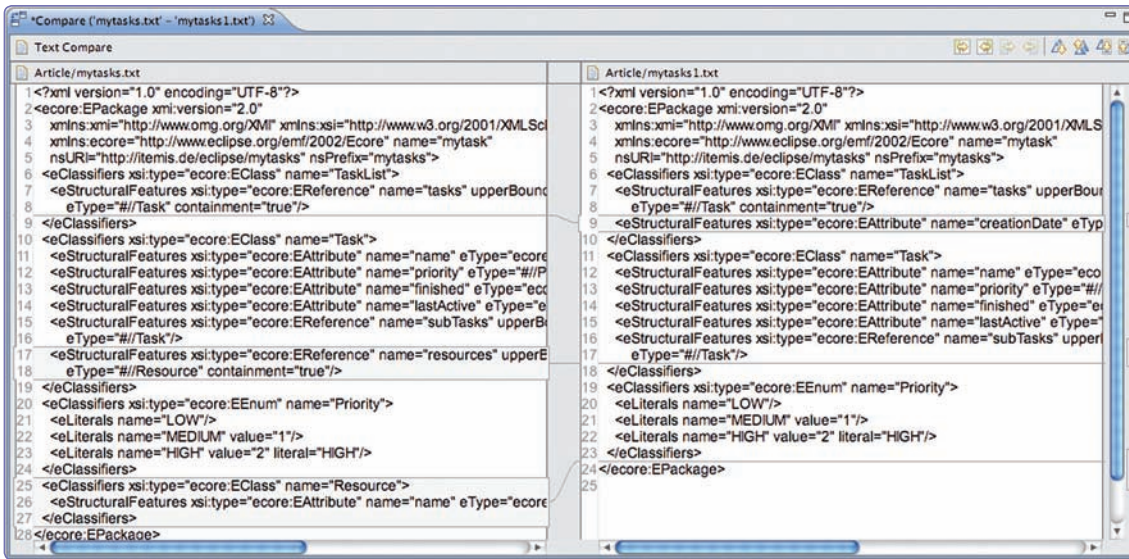
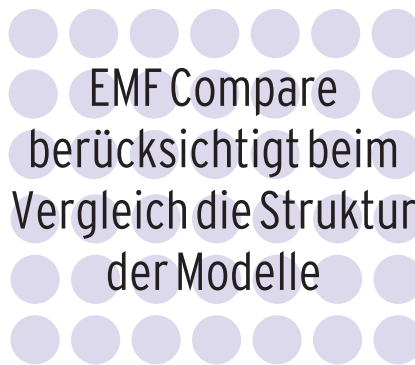


Abb. 1: Textueller Vergleich zweier serialisierter EMF-Modelle

rechte als das geänderte Modell. Übereinstimmende Paare von Elementen der beiden Modelle werden in einem Mapping abgelegt. Die Identifizierung der Paare beginnt zunächst bei den Wurzelementen der Modelle. Für jedes Wurzelement des linken Modells wird dazu aus einer vorbestimmten Menge der Wurzelemente des rechten Modells das ähnlichste Element herausgesucht. Die Größe der Menge wird durch den Parameter `OPTION_SEARCH_WINDOW` bestimmt, der auf 100 Elemente voreingestellt ist, aber geändert werden kann. Die Größe dieses Suchfensters beeinflusst die Laufzeit und die Präzision des Algorithmus.

Die übereinstimmenden Paare aus dem Mapping werden nun weiterverarbeitet. Für jedes Paar aus linkem und rechtem Modellelement wiederholt sich der Vorgang, der bereits für die Wurzelemente durchgeführt wurde. Dazu wird über die durch die Containment-Beziehung eines EMF-Modells gegebenen Kindelemente iteriert. Für jedes Kindelement des linken Modellelements wird wiederum das ähnlichste Kindelement des rechten Modellelements gesucht. Die Suche wird auch hier auf eine durch das Suchfenster bestimmte Menge beschränkt. Diese Schritte werden rekursiv fortgeführt, bis alle Modellelemente überprüft worden sind. Zur Messung der Ähnlichkeit zweier Modellelemente werden vier verschiedene Metriken verwendet. Die erste Metrik gibt an, wie ähnlich sich die Namen der beiden Modellelemente sind. Dazu wird jeweils das Attribut eines Modellelements gesucht,

dessen Bezeichnung dem String *name* am ähnlichsten ist. Der Wert dieses Attributs wird als Name des Elements interpretiert. Die zweite Metrik errechnet die Ähnlichkeit der Werte aller Attribute der Elemente. Die Referenzen der Elemente werden mithilfe der dritten Metrik miteinander verglichen. Die letzte Metrik berechnet die Ähnlichkeit der Elementtypen. Diese Metrik wird nur verwendet, wenn der Parameter `OPTION_DISTINCT_METAMODELS` auf *true* gesetzt wird. Er



gibt an, ob die Metamodelle der zu vergleichenden Modelle unterschiedlich sind. Als Voreinstellung ist *false* gesetzt, es wird also von gleichen Metamodellen ausgegangen. Alle Metriken liefern als Ergebnis Werte zwischen 0 und 1. Mithilfe der Metriken wird zunächst bestimmt, ob zwei Elemente grundsätzlich ähnlich sind. Stimmen z. B. die Namen überein – liefert also die Namensmetrik als Ergebnis 1 – gelten die Elemente als ähnlich. Liegt die Namensmetrik über einem vorgegebenen Schwellwert, werden die weiteren Metriken überprüft. Liegen auch diese über bestimmten Schwellwerten,

gelten die Elemente als ähnlich. Alle Metriken werden dann zu einem Gesamtergebnis zwischen 0 und 1 verrechnet, das die Ähnlichkeit der beiden Elemente ausdrückt und im Mapping der beiden Elemente gespeichert wird. Das Ergebnis dieses Algorithmus ist ein Matchmodell, das die beiden verglichenen Modelle enthält. Des Weiteren enthält es ein Mapping von übereinstimmenden Elementen und eine Menge von Elementen beider Modelle, für die keine Übereinstimmungen gefunden werden konnten.

### Bildung der Differenz

Das aus der ersten Phase stammende Matchmodell wird in der zweiten Phase zur Bildung der Differenzen der beiden Modelle genutzt. Zunächst wird das Mapping durchlaufen. Da zwei Modellelemente bereits als ähnlich gelten, wenn sie nur zu einem bestimmten Teil übereinstimmen, können sie kleinere Unterschiede aufweisen. Für jedes Paar aus linkem und rechtem Modellelement werden also die Unterschiede des rechten zum linken Element berechnet und als Änderungen am linken Element interpretiert. Zu den Änderungen gehören die Manipulation von Attributwerten, das Hinzufügen oder Löschen von Referenzen oder das Verschieben des Modellelements. Für jede dieser Änderungen wird eine Differenzbeschreibung erzeugt und im Differenzmodell hinterlegt. Anschließend werden die Elemente durchlaufen, für die keine Übereinstimmungen gefunden werden konnten. Gehört ein solches Element zum linken Modell, hat es keine Übereinstimmung im rechten Modell

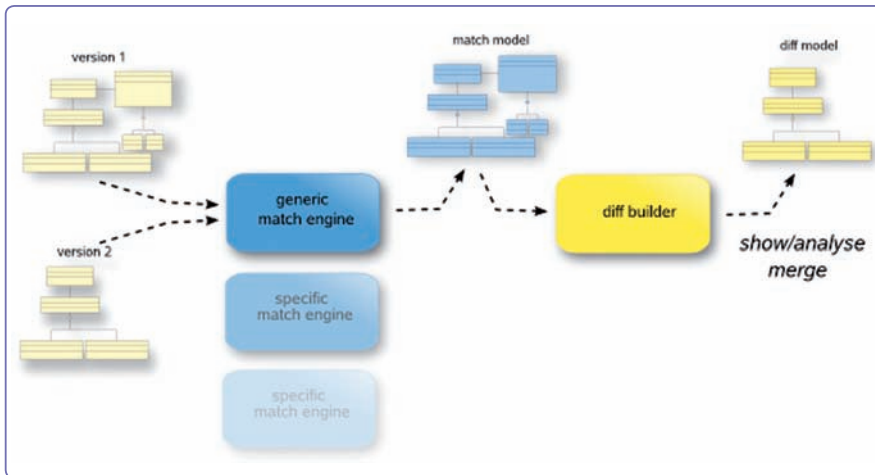


Abb. 2: Funktionsweise von EMF Compare (Quelle [4])

und gilt als gelöscht. Gehört ein solches Element allerdings zum rechten Modell, gilt es analog als hinzugefügt. Auch hierfür werden Differenzbeschreibungen angelegt und dem Differenzmodell hinzugefügt. Das Differenzmodell ist ebenfalls ein EMF-Modell und kann somit wie jedes andere EMF-Modell behandelt werden. Es enthält Referenzen auf die beiden verglichenen Modelle.

### Drei-Wege-Vergleich

Beim bisher beschriebenen Algorithmus wurden nur zwei Modelle miteinander verglichen. Das reicht aus, wenn man z. B. ein Modell mit seiner Vorgängerversion vergleichen will, um Änderungen nachzuvollziehen. Solch ein Vergleich wird auch „Zwei-Wege-Vergleich“ genannt. Will man den Algorithmus jedoch in einer Versionsverwaltung einsetzen, muss er in der Lage sein, drei Modelle miteinander zu vergleichen. Das erste der drei Modelle ist jenes, das der Benutzer initial ausgecheckt hat – es wird das „ursprüngliche Modell“ genannt. Das zweite Modell ist das vom Benutzer geänderte Modell – das ist in diesem Fall das rechte Modell. Das dritte Modell ist die aktuelle Version des Modells im Versions-Repository, das in der Zwischenzeit durch einen weiteren Benutzer geändert worden sein kann – das ist das linke Modell. Sowohl das linke als auch das rechte Modell stammen vom ursprünglichen Modell ab. Will der Benutzer nun seine geänderte Version des Modells einchecken, so müssen alle drei Modelle miteinander verglichen werden, um die Änderungen des linken Modells aus dem Repository mit den eigenen Änderungen

aus dem rechten Modell in einem gemeinsamen Modell zusammenzuführen. Dabei werden eventuelle Konflikte erkannt, bei denen an gleichen Stellen Änderungen vorgenommen worden sind. Ein solcher Vergleich wird „Drei-Wege-Vergleich“ genannt. Das Matchmodell verbindet beim Drei-Wege-Vergleich jeweils ein Element aller drei Modelle. Zunächst werden die Übereinstimmungen des linken und rechten Modells mit dem ursprünglichen Modell in Zwei-Wege-Vergleichen berechnet. Zu jedem Paar aus linkem und ursprünglichem Modellelement wird dann das passende Element aus dem rechten Modell ermittelt, das mit dem Element des ursprünglichen Modells übereinstimmt. Bei der Bildung des Differenzmodells werden dann die drei Elemente auf Änderungen hin überprüft. Unterscheiden sich z. B. das linke und rechte Element, so liegt ein Konflikt vor.

### Zusammenführen von Modellen

Wurde ein Differenzmodell berechnet, kann man die Differenzbeschreibungen dazu nutzen, die Änderungen von zwei Modellen in ein gemeinsames Modell zu übernehmen. Für jede Art von Änderung gibt es dazu sog. „Merger“, die jeweils eine Änderung an dem gemeinsamen Modell durchführen. Die Änderungen, die zu keinem Konflikt führen, können mithilfe des MergeService automatisch übernommen werden. Konflikte müssen dagegen manuell aufgelöst werden.

### Laufzeitverhalten

Die Modelle in vielen praktischen Anwendungen werden zunehmend größer.

Anzeige

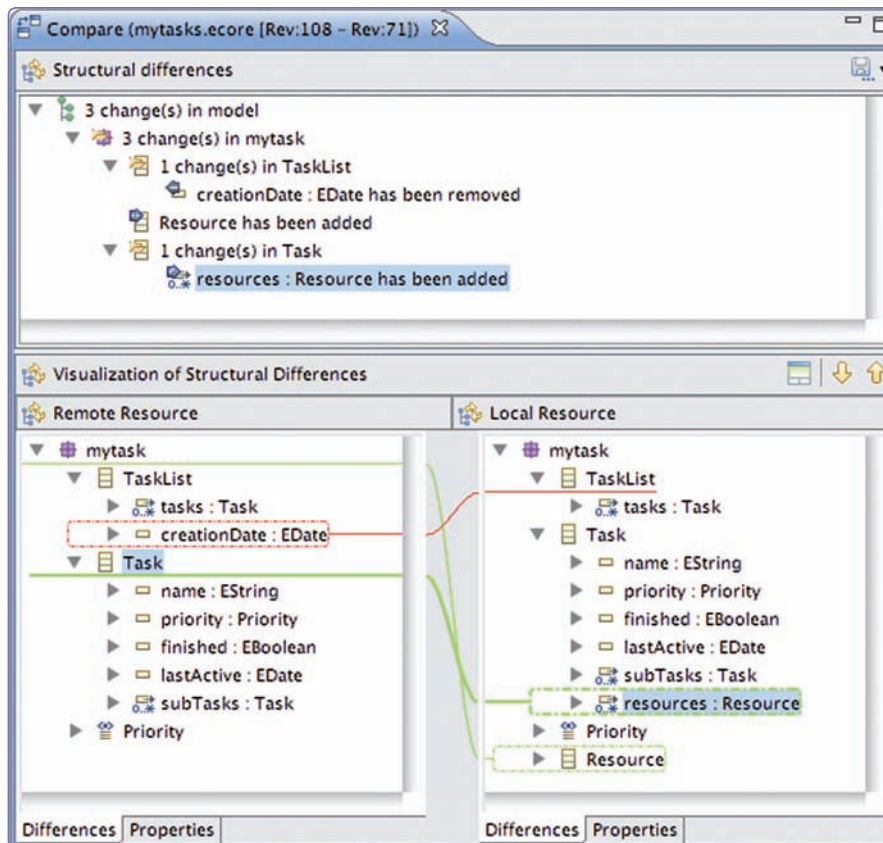


Abb. 3: Das grafische Vergleichswerkzeug von EMF Compare

Modelle von elektronischen Steuergeräten aus dem Automobilbau erreichen z. B. schon heute eine Größe von mehr als 100 000 Elementen. Die Performanz des Vergleichsalgorithmus spielt also eine große Rolle. Wir haben deshalb untersucht, wie sich der Vergleichsalgorithmus im Hinblick auf die Größe der Modelle und die Anzahl der Änderungen an den Modellen verhält. Für diesen Versuch haben wir ein einfaches hierarchisches Metamodell nach dem Composite-Design-Pattern erstellt. Neben der hierarchischen Beziehung existiert

eine weitere bidirektionale Beziehung zwischen den Klassen des Metamodells, die mehrere Modellelemente miteinander verbindet. Des Weiteren besitzen die Klassen unterschiedliche Attribute. Auf Basis dieses Metamodells erzeugen wir zufällige Modelle mit bis zu 100 000 Elementen. Für den Vergleich wird jeweils ein solches Modell geklont und eine vorbestimmte Menge ebenfalls zufälliger Änderungen am Klon durchgeführt. Die Änderungen umfassen Attribute, hinzugefügte, gelöschte und verschobene Elemente sowie veränderte Referenzen

zwischen den Elementen. Bei dem Benchmark haben wir uns auf den Zwei-Wege-Vergleich zwischen den so erzeugten Modellen und ihren veränderten Klonen konzentriert. Der Drei-Wege-Vergleich unterscheidet sich im Prinzip nicht sehr vom Zwei-Wege-Vergleich, sodass die Ergebnisse tendenziell vergleichbar sind. Jeder der Benchmarks wurde jeweils fünf Mal durchgeführt und die Ergebnisse gemittelt, um den Einfluss von Streuungen zu verringern. Als Vergleichsparameter wurde die voreingestellte Suchfenstergröße von 100 gewählt sowie der Parameter für gleiche Metamodelle auf *true* gesetzt. Die Quelltexte zu diesem Benchmark finden sich auf der Heft-CD.

Die Tabellen 1 und 2 enthalten die Ergebnisse des Benchmarks (ausgeführt auf einem Intel Core2 Duo T9500 mit 4 GB RAM unter Windows XP). Jeder Wert in den Tabellen ist ein Durchschnitt aus fünf Durchläufen. Die erste Tabelle zeigt die Zeiten zur Berechnung eines Matchmodels bei unterschiedlich großen Modellen und unterschiedlich vielen Änderungen. Wie zu sehen ist, wird das Berechnen des Matchmodels von der Anzahl der Änderungen kaum beeinflusst. In der zweiten Tabelle ist jeweils die Dauer zur Berechnung der Differenzen angegeben. Gerade hierbei fiel eine erhebliche Streuung unter den Durchläufen auf. Dies erklärt auch die höheren Werte von kleineren gegenüber größeren Modellen. Auffällig sind auch die Zeiten für 1 000 Änderungen bei der Berechnung der Differenzen. Die Werte zeigen, dass EMF Compare bei einem solchen Szenario nicht mehr praxistauglich ist. Eine solche Zahl von Änderungen ist in der Praxis aber durchaus vorstellbar, z. B. bei automatischem Refactoring. Liegen dagegen nur wenige Änderungen vor, wovon z. B. bei rein manuellen Änderungen auszugehen ist, so ist EMF Compare auch bei 100 000 Modellelementen noch praxistauglich. Auch die Art der Änderungen hat unterschiedlichen Einfluss auf die Dauer. Wir konnten beobachten, dass vor allem Verschiebungen von Elementen viel Zeit bei der Berechnung der Differenzen kosten.

Elemente Änderungen	1 000	4 000	7 000	10 000	40 000	70 000	100 000
10	334 ms	1,48 s	2,55 s	3,94 s	21,73 s	47,84 s	80,74 s
100	483 ms	1,69 s	2,82 s	4,02 s	22,08 s	47,92 s	80,25 s
1 000	-	-	-	14,86 s	38,41 s	66,69 s	103,19 s

Tabelle 1: Berechnung der Übereinstimmung

Elemente Änderungen	1 000	4 000	7 000	10 000	40 000	70 000	100 000
10	75 ms	2,92 s	1,27 s	1,63 s	14,50 s	42,97 s	87,88 s
100	1237 ms	5,48 s	12,92 s	5,89 s	23,19 s	62,57 s	119,11 s
1 000	-	-	-	2,7 min	12,6 min	11,7 min	71,2 min

Tabelle 2: Berechnung der Differenzen

### Das Vergleichswerkzeug

EMF Compare gibt dem Anwender ein grafisches Werkzeug an die Hand, das das Vergleichen von Modellen aus Eclipse heraus ermöglicht. Um diesen



Editor zu öffnen, genügt es, zwei bzw. drei EMF-Modelle im Workspace zu selektieren und im Kontextmenü die Aktion *Compare with/Each Other* auszuwählen. Wird ein Versionskontrollsystem verwendet, das über das Eclipse-Team-API in die IDE integriert ist, können auch Modelle aus verschiedenen Versionen miteinander verglichen werden. Das ist besonders beim parallelen Arbeiten an Modellen hilfreich, da der Editor auch das Zusammenführen von Modellen unterstützt. Standardmäßig ist EMF Compare auf die typischen Dateierweiterungen serialisierter (Meta-)Modelle registriert, z. B. *ecore*, *xmi*, oder auch *\*.uml*. Unter *Preferences/General/Content Types/EMF Compare* können weitere Dateierweiterungen hinzugefügt werden.

Abbildung 3 zeigt den Editor bei einem Zwei-Wege-Vergleich zweier verschiedener Versionen des Metamodells *mytask.ecore*. Dieses Metamodell verwenden wir später im Praxisartikel (in der nächsten Ausgabe des Eclipse Magazins) als Beispiel, wo wir auch ausführlich auf seinen Aufbau eingehen werden. Bei einem Zwei-Wege-Vergleich wie in Abbildung 3 unterteilt sich der grafische Editor in zwei Bereiche. Im Bereich *Structural differences* (oben) wird eine Zusammenfassung aller Unterschiede in Textform gegeben. Im Bereich *Visualization of Structural Differences* (unten) sind die beiden zu vergleichenden Modelle gegenübergestellt. Die einzelnen Modelle werden in einer Baumstruktur dargestellt, die die Containment-Hierarchie der Elemente wiedergibt. Farbige Linien, so genannte *difference markers*, verdeutlichen die Art der Modelländerungen:

- grün: ein Modellelement wurde hinzugefügt,
- rot: ein Modellelement wurde entfernt,
- blau: ein Modellelement hat sich geändert.

Die Action *export Diffmodel* (rechts oben) erlaubt es, den aktuellen Vergleich als Modell mit der Dateierweiterung *.emfdiff* zu serialisieren, dem Dateiformat von EMF Compare. Auf dieser Dateierweiterung ist der grafische Editor ebenfalls registriert. Diese Datei ist selbst wiederum ein persistiertes EMF-Modell und kann so per Modelltransformation weiterver-

arbeitet werden. Weitere Actions unterstützen das Zusammenführen von Modelländerungen, wie man es bereits von den textuellen Vergleichswerkzeugen kennt.

### Anpassen und Erweitern

EMF Compare bietet einige Einstellungsmöglichkeiten, um das Verhalten des generischen Vergleichsalgorithmus, der vom grafischen Editor verwendet wird, an spezielle Bedürfnisse anzupassen. Eine Preference Page erlaubt beispielsweise das Setzen des *Search-Window*-Parameters, der wie bereits erwähnt die Größe der Suchmenge angibt. Eine weitere sehr nützliche Option ist *Ignore XMI ID for*

## EMF Compare gibt ein grafisches Werkzeug zum Vergleichen von Modellen an die Hand

*model comparison*. Normalerweise sieht die XMI-Spezifikation vor, dass jedes Modellelement durch eine eindeutige ID identifiziert wird. Manche Werkzeuge berechnen diese ID jedoch unterschiedlich, was dazu führen kann, dass EMF Compare gleiche Modellelemente als unterschiedlich ansieht. Durch das Aktivieren dieser Option kann man dieses Problem umgehen.

Weiterhin definiert das EMF Compare API einige *extension points*, die es erlauben, die Funktionsweise des Vergleichsalgorithmus direkt zu beeinflussen. Durch die Erweiterung von *org.eclipse.emf.compare.match.engine* kann die *Match Engine* angepasst werden, die für die Identifikation von Übereinstimmungen verantwortlich ist. Dies macht oft Sinn, wenn man die Bedeutung spezifischer Modellelemente berücksichtigen möchte. Anwendungsfälle hierfür sind beispielsweise das Herausfiltern von Modellelementen, die für die zweite Metrik (Abschnitt Funktionsweise) nicht berücksichtigt werden sollen, oder aber die direkte Implementierung der Methode zur Berechnung der Namens-

ähnlichkeit (Metrik 1). Zwar verliert der Algorithmus dadurch seinen generischen Charakter und kann so nicht mehr für jedes beliebige Metamodell verwendet werden, dafür kann man so die Geschwindigkeit des Algorithmus und die Präzision des Resultats erhöhen.

Ähnliches gilt für den Extension Point *org.eclipse.emf.compare.diff.engine*. Hier kann man die Art und Weise, wie Unterschiede interpretiert werden, anpassen. Ein konkretes Beispiel, wie die generische *DiffEngine* angepasst wird, um ein Attribut zu ignorieren, zeigen wir im zweiten Teil dieser Serie. Der grafische Editor stellt den Extension Point *org.eclipse.emf.compare.ui.export* bereit. Hier können Handler hinzugefügt werden, die den aktuellen Modellvergleich in einem eigenen Format serialisieren, z. B. als HTML Report.



**Andreas Müller** arbeitet als Softwarearchitekt und Entwickler bei der itemis AG in Lünen. Seine Schwerpunkte liegen in der modellgetriebenen Softwareentwicklung und Werkzeugketten mit Eclipse.

**Kontakt:** [Andreas.Muelder@itemis.de](mailto: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 DSIs.

**Kontakt:** [Holger.Schill@itemis.de](mailto: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](mailto:Lothar.Wendehals@itemis.de)

### >> Links & Literatur

- [1] Eclipse Modeling Framework Project: <http://www.eclipse.org/modeling/emf/>
- [2] EMF Compare: <http://www.eclipse.org/modeling/emft/?project=compare>
- [3] EMF Compare FAQ: [http://wiki.eclipse.org/EMF\\_Compare\\_FAQ](http://wiki.eclipse.org/EMF_Compare_FAQ)
- [4] [http://wiki.eclipse.org/index.php/EMF\\_Compare](http://wiki.eclipse.org/index.php/EMF_Compare)