

Plus CD!

Im Gespräch ▶ Wayne Beaton, Jeff McAffer, Mik Kersten, Roy Ganor, David Williams u.v.m.

5.09

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



eclipse  
MAGAZIN

# eclipse

## MAGAZIN

www.eclipse-magazin.de

**Net4j**  
The Signalling Platform

**EXKLUSIV AUF CD:**

**FSHARE-SERVER-  
PLUG-IN**

File-Sharing mit der Net4j Signalling Platform

**EMFText**

Komplette Beispiel-DSL aus unserem Tutorial  
„Sprechen Ihre Modelle Klartext?“

**Eclipse Classic 3.5**

Galileo-Basispaket: Eclipse-Plattform,  
Java Development Tools, Plug-in  
Development Environment u.v.m.

**Xtext 0.7**

Das neue Framework zur  
Entwicklung externer  
textueller DSLs

# Eclipse Galileo!

▶ **DSLs mit Xtext** >> 67

Domänenspezifische Sprachen  
selbst gemacht

▶ **My Mylyn** >> 46

Eigene Mylyn Connectors entwickeln

▶ **Vorschau auf Eclipse 4.0** >> 93

User Interface Styling mit CSS

**Großer Jahresrückblick: Was die  
Community im Galileo-Jahr bewegte** >> 10

**Was gibt's Neues in Galileo JDT?** >> 27

**Quantensprung für Equinox** >> 30

**PDT 2.1: Des Kaisers neue Kleider** >> 36

**Runde sechs für CDT** >> 32

**Modeling-Trends in Eclipse Galileo** >> 40

**„Wir werden e4 in Helios wiedersehen“** >> 43

Wayne Beaton über Gegenwart und Zukunft von Eclipse



D 68864

Datenträger enthält  
Info- und  
Lehrprogramme  
gemäß §14 JuSchG

GenGMF: Grafische Editoren für umfangreiche Metamodelle

# Kistenzauberei und Verwandlungskünstler

Quellcode  
auf CD!

>> ENRICO SCHNEPEL

Im ersten Teil unserer Artikelserie [1] wurde GenGMF verwendet, um anhand eines anschaulichen Beispiels einen grafischen Editor für ein komplexes Metamodell zu erstellen. Der Entwickler wurde dabei durch kontextsensitive Wizards unterstützt, um das Layout der grafischen Elemente in Templates zu definieren und diese mithilfe von Deskriptoren mit den Elementen des Metamodells zu verknüpfen. Mit GenGMF wurden anschließend die GMF-Modelle generiert, aus denen wiederum der eigentliche grafische Editor erzeugt wurde. In diesem Artikel soll nun das Modell erweitert werden, um Kompositionen im grafischen Editor adäquat darstellen zu können. Des Weiteren soll der Editor für jeden Knoten eine andere Hintergrundfarbe erhalten. Hierfür werden eigene Transformationen erstellt und im Modell hinterlegt.



Bisher haben wir uns mit relativ einfachen Elementen beschäftigt, die im grafischen Editor auf der Diagrammfläche platziert und mit Pfeilen verbunden werden konnten. Oft ist es jedoch gewünscht, inhaltliche Beziehungen von Elementen zueinander darzustellen. Hierzu wird das im ersten Artikel verwendete Beispiel eines Workflow-Editors um das Konzept der Prozesshierarchien erweitert.

Um Prozesshierarchien mit dem Metamodell semantisch abbilden zu können, wird dieses um den Knotentyp *SubProcess* ergänzt, der allein schon durch die Mehrfachableitung von den bestehenden Typ *IntermediateState* und *StateContainer* alle für ihn relevanten Funktionalitäten erbt. Das komplette Metamodell ist in Abbildung 1 darge-

stellt. Das auf der Heft-CD enthaltene Projekt entspricht dem fertigen Projekt aus dem ersten Artikel inklusive dem neuen Knotentypen.

Die Figur für den neuen Knoten soll die Kindelemente in einem im Elternknoten enthaltenen Rechteck darstellen. In GenGMF bedarf es hierfür eines *CompartmentTemplate*-Elements. Der passende Wizard wird über das Wurzelement *Model Workflow* aufgerufen. Als Name wird für dieses Beispiel *SubProcessState* eingetragen. Wie auch für die *SimpleState*-Figur aus dem ersten Artikel geschehen, soll auch dem neuen Knoten ein Label für den Klassennamen (via `ADD CLASS NAME LABEL`) sowie ein dynamisches *feature label* dem Template hinzugefügt werden. Zur Aufnahme der Kindelemente ist es nötig, anschließend noch ein Rechteck mit dem Wizard `ADD NEW COMPARTMENT RECTANGLE` zu definieren. Wie auch für die anderen Knoten muss nach der Fertigstellung des Templates ein Deskriptor mit

dem Wizard `CREATE NEW COMPARTMENT DESCRIPTORS USING TEMPLATE SUBPROCESSSTATE` angelegt werden. Als Containment-Referenz wird *Workflow.states* ausgewählt und anschließend ein passendes *SubProcess*-Toolelement erstellt.

Im Deskriptor wird nun definiert, welche Kindelemente der Knoten aufnehmen können soll. Dies geschieht mit dem Wizard *Add child descriptors...* auf dem *SubProcess*-Deskriptor. Durch den Wizard werden nach und nach verschiedene Kombinationen aus Containment-Referenz und dazu passenden Kindelementen abgefragt. Dabei ist bei den Kindelementen eine Mehrfachauswahl erlaubt. Für den *SubProcess*-Knoten sollen jeweils alle möglichen Elemente der Referenzen *startState*, *endState* sowie *states* ausgewählt werden.

Durch den Wizard ist nun für jeden potenziellen Kindknoten ein *CompartmentChildDesc*-Element angelegt und entsprechend mit einer Referenz auf den „echten“ Deskriptor des Kindes initialisiert worden. Für den *SubProcess*-Knoten sind damit keine weiteren Schritte in der Modellierung notwendig, denn die möglichen Kindelemente wurden bereits in den anderen Deskriptoren beschrieben.

Aus dem gespeicherten Modell können nun mit dem Kontextmenüeintrag `GENERATE GMF-GRAPH AND -MAP MODELS...` (im PROJECT EXPLORER) die GMF-Modelle erstellt werden. Der eigentliche grafische Editor wird wieder über das GMF-Generator-Zwischenmodell *workflow.gmfgen* generiert. Der fertige Editor ist mit einem Beispielmmodell in Abbildung 2 dargestellt.

## Verwandlungskünstler

In Abbildung 2 ist zu sehen, dass jeder Knoten im Prinzip dasselbe grafische Aus-



sehen hat. Ein guter grafischer Editor sollte jedoch aus Gründen der Benutzerfreundlichkeit die verschiedenen Modellierungselemente ausreichend differenzierbar darstellen. Im Rahmen des Workflow-Editors sollen den verschiedenen Knotentypen eigene Hintergrundfarben und Formen zugewiesen werden. Zu diesem Zweck werden in GenGMF Transformatoren verwendet. Transformatoren werden für die Generierung der GMF-Modelle genutzt, um das Modell zu modifizieren. Sie wirken immer nur auf den Deskriptor, dessen Kindelement sie sind. Grundsätzlich gibt es dabei zwei Arten von Modifikationen: Veränderung von Typen während sowie Modellverarbeitung nach der Generierung der GMF-Modelle.

Für den ersten Fall wird der *ElementTypeTransformer* genutzt, um Typen von einzelnen GMF-Modellelementen zu verändern. So ist es z. B. möglich, statt des bei GenGMF standardmäßig verwendeten Rechtecks mit abgerundeten Ecken (*RoundedRectangle*) eines mit normalen Ecken (*Rectangle*) darzustellen. Alle Attribute und Referenzen des ursprünglichen Elements werden dabei eins zu eins übernommen, wenn sie in einer gemeinsamen Basisklasse definiert wurden (was in der Regel der Fall ist).

Der Workflow-Editor soll für die Start- und Endknoten Kreisdarstellungen erhalten. Bei den beiden Knoten können mit dem Wizard *TRANSFORM THE FIGURE LAYOUT TYPE...* Kindelemente vom Typ *ElementTypeTransformer* angelegt und auch schon vorinitialisiert werden. So wird z. B. angenommen, dass die Hauptfigur (*RoundedRectangle \_\_CLASSNAME\_\_StateNodeRect*) modifiziert werden soll, was jedoch leicht geändert werden kann. Es muss nur noch die EType-Referenz auf *EClass gmfgraph::Ellipse* gesetzt werden. Im Template ist zwar noch das Rechteck eingetragen, die beiden Knoten bekommen jedoch während der Generierung eine andere Form verpasst.

Um die angesprochene Hintergrundfarbe für die verschiedenen Knotentypen zu setzen, kann man einen *ColorTransformer* verwenden. Als Figur muss das mit einer Farbe zu belegende Element angegeben werden (hier *RoundedRectangle \_\_CLASSNAME\_\_StateNodeRect*). Die Farbe selbst kann für den Vorder- und Hintergrund in Form von Farbkonstanten (*TRANSFORM THE*

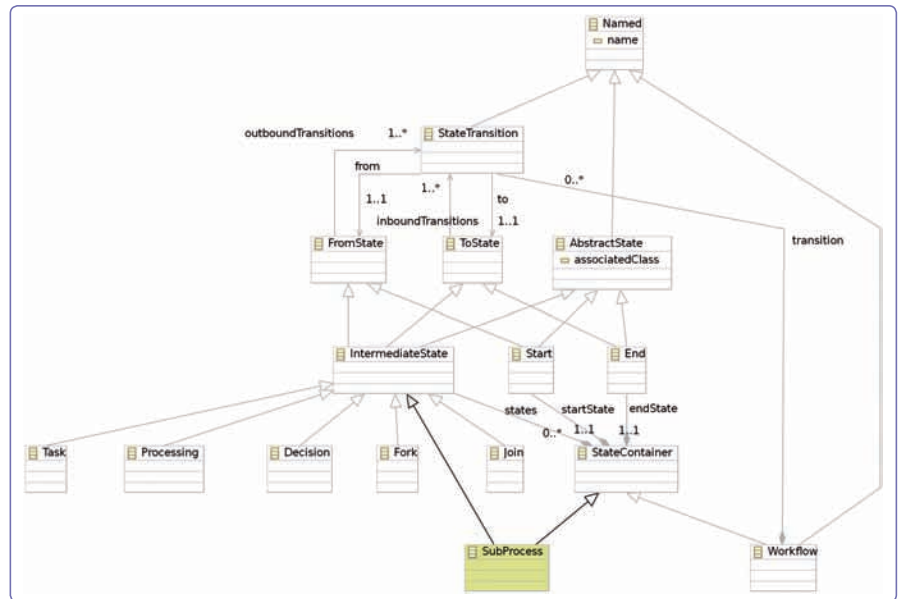


Abb. 1: Erweitertes Workflow-Metamodell

FIGURE BG OR FG COLOR USING CONSTANTS ...) bzw. über die RGB-Farbwerte (*TRANSFORM THE FIGURE BG OR FG COLOR USING RGB VALUES ...*) spezifiziert werden. Für die einzelnen Knotentypen verwenden wir in diesem Beispiel die in Tabelle 1 enthaltenen Vorder-/Hintergrundkonstanten. Aus dem fertigen Modell kann man dann wieder die GMF-Modelle sowie anschließend den grafischen Editor generieren.

Der fertige Editor mit geänderten Farben und Formen ist in Abbildung 3 dargestellt. Im Vergleich zum „farb- und

formlosen“ Editor aus Abbildung 2 ist der Wiedererkennungswert der einzelnen Elemente wesentlich höher.

### Meister der Zauberei

Sollte einmal etwas nicht mit den bis jetzt vorgestellten Transformatoren umsetzbar sein, bietet sich der Einsatz des *ScriptTransformators* an, der es ermöglicht, programmatisch Modellveränderungen durchzuführen. Für den Einsatz muss im Wurzelement das Attribut *Creation Transformer Ext* (für die Typtransformationen)

Knotentyp	Vordergrundfarbe	Hintergrundfarbe
Start	white	darkGrey
End	white	grey
Task	black	yellow
Process	black	orange
SubProcess	black	red
Decision	black	blue
Fork	black	cyan
Join	black	green

Tabelle 1: Verwendete Farbkombinationen für die einzelnen Knotentypen

Deskriptortyp	Beschreibung
<i>NodeDesc</i>	Knoten ohne Kindelemente
<i>CompartmentDesc</i>	Knoten mit Kindelementen
<i>EdgeDesc</i>	Elementbasierte Verbindungen
<i>ReferenceEdgeDesc</i>	Referenzbasierte Verbindungen
Superklassen für Deskriptoren	verwendet für
<i>AbstractNodeDesc</i>	<i>NodeDesc</i> und <i>CompartmentDesc</i>
<i>AbstractEdgeDesc</i>	<i>EdgeDesc</i> und <i>ReferenceEdgeDesc</i>
<i>ClassFigureDesc</i>	<i>NodeDesc</i> , <i>CompartmentDesc</i> und <i>EdgeDesc</i>
<i>FigureDesc</i>	alle Deskriptoren

Tabelle 2: Verwendete Deskriptortypen für das Skriptinterface

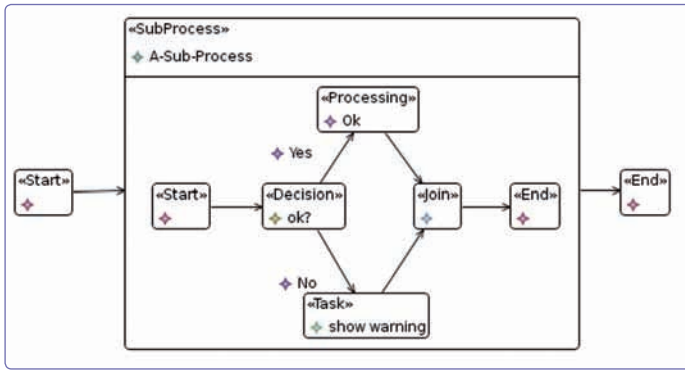


Abb. 2: Erweiterter grafischer Editor

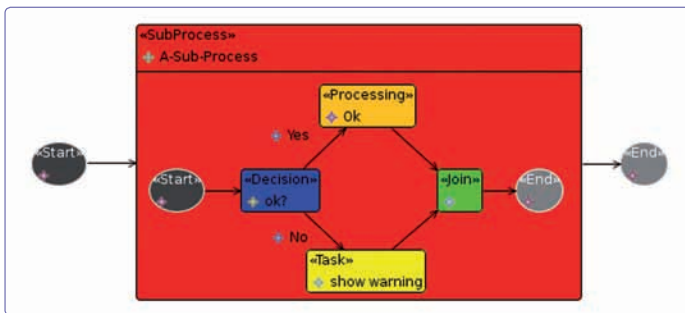


Abb. 3: Fertiger grafischer Editor mit farbigen Knoten und geänderten Formen

bzw. *Post Proc Transformator Ext* (für Nachverarbeitung) mit einem Verweis auf ein XTend-Script (Kasten: „openArchitectureWare“) gefüllt werden. Das Script muss im Klassenpfad desselben Projekts liegen. Die Angabe erfolgt dabei als Ressourcenpfad, jedoch mit zwei Doppelpunkten als Pfadtrenner (*some::package::MyXTendScript*) und

ohne Nennung der Dateiendung *.ext*.

Die Funktionen müssen dabei der Prototypdefinition [*ElementTyp*] [*Funktionsname*]([*ElementTyp*] *element*, [*DescriptorTyp*] *desc*) entsprechen. Der [*Funktionsname*] ist frei wählbar. Als [*DescriptorTyp*] ist einer der in Tabelle 2 beschrieben zu verwenden. Bei Typumwandlungen kann für den [*ElementTyp*] jeder in einem Template verwendete Typ oder eine Superklasse davon angegeben werden. Die Funktion muss ein neues Element des neuen Typs zurückgeben. Der [*ElementTyp*] muss für die Nachverarbeitung ein direktes Kindelement des Templates (*FigureDescriptor*, *Node*, *Label*, *Compartment* oder *Connection*) sein. Welcher Knoten letztendlich verändert wird, hängt von der implementierten Logik ab. Die „Wahlfreiheit“ ermöglicht die Verwendung einer Funktion für mehrere Deskriptoren. Um eine erstellte Funktion in einem Deskriptor zu verwenden, muss dem Deskriptor ein *ScriptTransformator* hinzugefügt werden. In diesem wird im Attribut *Function Name* der gewählte

Funktionsname eingetragen. Letztendlich lassen sich die Funktionalitäten der oben angesprochenen Transformatoren auch mit einem *ScriptTransformator* umsetzen. Für den *ElementTypeTransformator* sieht das passende Script folgendermaßen aus (Im auf der CD enthaltenen Demoprojekt wird das Script für den Endknoten verwendet):

```

import gengmf::desc;
import gmfigraph;
Ellipse CircleNode(RoundedRectangle fd, NodeDesc nd) :
    new Ellipse;
  
```

### Ausblick

Es hat sich gezeigt, dass mit GenGMF schnell Resultate in der Entwicklung grafischer Editoren erzielt werden können. Aufbauend auf einem EMF-basierten Metamodell werden mit der Unterstützung durch interaktive Wizards GenGMF-Modelle iterativ aufgebaut. Verwendet werden kann GenGMF sowohl für das „Master-Modell“, bei dem die GMF-Modelle jedes mal nach dem Editieren des Masters neu generiert werden, als auch als „Scaffolding“-Ansatz, also zum initialen Aufsetzen einer Infrastruktur für den grafischen Editor.

Für GenGMF werden in Zukunft weitere Wizards zu Verfügung gestellt, um die Entwicklung von grafischen Notationen für domänenspezifische Sprachen noch weiter zu vereinfachen. Aktuell nur über das Skriptinterface unterstützte Funktionen, wie *Initializer* und *Constraint* aus dem GMF-Map-Modell, sollen direkt verwendet werden können und sind für das kommende Release 2.1 geplant. Außerdem wird angestrebt, GenGMF als eigenständiges Eclipse-Projekt zu etablieren.



**Enrico Schnepel** arbeitet bei der b+m Informatik AG [4] als Softwarearchitekt im Bereich der modellgetriebenen Softwareentwicklung und domänenspezifischen Sprachen. GenGMF ist als Teil seiner Diplomarbeit entstanden und wird inzwischen in verschiedenen Projekten eingesetzt.

### » Links & Literatur

- [1] Schnepel, Enrico: „GenGMF - Jetzt wird gezaubert...“, in Eclipse Magazin 4. 09, S. 48
- [2] Schnepel, Enrico: „GenGMF - a GMF Model Generator“: <http://gengmf.randomice.net/>
- [3] Stahl, Thomas, et. al.: „Modellgetriebene Softwareentwicklung“, 2. Edition (2007)
- [4] b+m Informatik AG: <http://engineering.bmiag.de/>

### openArchitectureWare

GenGMF baut auf openArchitectureWare (oAW) auf, einem ausgereiften Framework für die modellgetriebene Softwareentwicklung. Hier ist XTend eine funktionale Sprache für Modelltransformationen, die in GenGMF für die Generierung der GMF-Modelle und für die Skriptunterstützung verwendet wird.

### GenGMF – Short Facts

Mit GenGMF können grafische Editoren für ein bestehendes EMF-basiertes Metamodell innerhalb weniger Minuten erstellt werden. Die Entwicklung des Editors erfolgt dabei iterativ, wobei der Entwickler durch Wizards unterstützt wird. Durch die GenGMF Wizards wird die im GMF enthaltene Komplexität auf ein überschaubares Minimum reduziert.

Für jede grafisch ähnlich aussehende Gruppe von Elementen wird zuerst ein Template angelegt, das alle für das Layout wichtigen Definitionen enthält. Anschließend wird für jedes Element des Editors ein Deskriptor erstellt, der das zum Element passende Template mit den Elementen des Metamodells verknüpft. Bei jedem Schritt wird der Entwickler durch spezialisierte Wizards unterstützt. Für den weiteren Verlauf werden wichtige Informationen dabei interaktiv abgefragt. Anschließend werden aus dem fertigen GenGMF-Modell automatisch die GMF-Modelle generiert. Der eigentliche grafische Editor wird aus den GMF-Modellen erzeugt, ohne die GMF-Modelle noch einmal verändern zu müssen.