Entwicklerunterstützung für einfache Erweiterungen von Eclipse GMF-basierten Editoren

Ralf Laue und Arian Storch
{laue|storch}@ebus.informatik.uni-leipzig.de
Universität Leipzig, Lehrstuhl für Angewandte Telematik und E-Business

Frank J. Rump
rump@informatik-emden.de
Hochschule Emden/Leer, Fachbereich Technik

Markus Nüttgens
{markus.nuettgens@wiso.uni-hamburg.de
Universität Hamburg/Forschungsschwerpunkt Wirtschaftsinformatik

Frank Hogrebe
frank.hogrebe@hfpv-hessen.de
Hessische Hochschule für Polizei und Verwaltung, Fachbereich Verwaltung

Abstract: Für Wissenschaft und Praxis bieten die quelloffenen Eclipse-Technologien EMF und GMF sehr gute Voraussetzungen, um graphische Editoren für vorhandene oder selbst definierte Modellierungssprachen zu erstellen. Im Sinne der Open Source-Philosophie ist es wünschenswert, dass in einen solchen Editor weitere vorhandene Werkzeuge (etwa zur Analyse von Modellen) möglichst einfach integriert werden können. Dadurch erhöhen sich die Einsatzmöglichkeiten des Editors.

Einer solchen einfachen Integration steht entgegen, dass die Verwendung von Eclipse-Technologien (wie EMF und GMF) einigen Einarbeitungsaufwand erfordert. In diesem Beitrag wird gezeigt, wie der Einbau zusätzlicher Abstraktionsschichten diesen Einarbeitungsaufwand deutlich reduziert.

1 Einleitung

Das Eclipse Graphical Modeling Project (GMP) stellt erprobte Werkzeuge für die Entwicklung graphischer Editoren zur Verfügung. Das Eclipse Modeling Framework (EMF) unterstützt die Definition von Modellen und das Graphical Editing Framework (GEF) die Entwicklung von graphischen Repräsentationen von Modellen [SBPM09, IBM04]. Durch das EMF kann das Metamodell einer graphischen Modellierungssprache definiert werden. Auf Basis von GEF kann zu einem Metamodell ein entsprechender graphischer Editor nach der Model-View-Controller-Architektur umgesetzt werden.

Da viele graphische Editoren eine große Ähnlichkeit und somit ähnlichen Programmcode aufweisen, wurde im Rahmen des Eclipse-Projektes weiterhin das Graphical Modeling

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4

Framework (GMF) entwickelt, das nach der Definition einer Abbildung von Metamodellobjekten auf deren graphische Repräsentation die automatische Generierung eines graphischen Editors vornimmt. Dieser kann danach an eigene Bedürfnisse angepasst werden.

2 Einstiegshürden beim Integrieren eigener Werkzeuge in den Editor

EMF, GEF und GMF stellen mächtige Konzepte bereit, um leistungsfähige Editoren zu entwickeln. Wichtige Bestandteile eines Editors - etwa die Serialisierung - erhält der Entwickler durch den Einsatz dieser Frameworks "geschenkt".

Aus der Sicht eines mit der Eclipse-Entwicklung nicht vertrauten Entwicklers, der ein bereits existierendes Programm in den Editor einbinden will, sieht die Situation allerdings weit weniger erfreulich aus als für erfahrene Eclipse-Programmierer. Er muss verstehen,

- wie auf das im internen Datenformat vorliegende Modell zugegriffen werden kann,
- wie die Modellinformationen in das vom eigenen Programm erwartete Dateiformat gebracht werden können,
- wie das einzubindende Programm aus der Eclipse-Arbeitsumgebung heraus gestartet wird,
- wie die vom eigenen Programm ermittelten Informationen zurück in die Eclipse-Ansichten gebracht werden.

In diesem Beitrag wird gezeigt, wie diese Einstiegshürden umgangen werden können. Hierzu stellen wir eine Sammlung von Eclipse-Plugins vor, das sich das Ziel gestellt hat, den Aufwand für Erweiterungen des Funktionsumfangs von GEF/GMF-basierten Editoren zu minimieren.

3 Export und Import

Für einen quelloffenen Diagrammeditor sind vielfältige Export- und Importmöglichkeiten ein wichtiges Qualitätsmerkmal. Eine einfache Schnittstelle soll Entwickler einladen, ihre eigenen Exporte und Importe zum Editor hinzuzufügen.

Da die Serialisierung in einem EMF-basierten Editor das auf XML-Basis beruhende Austauschformat XMI verwendet, bieten sich zur Realisierung von Ex- und Importschnittstellen XML-Standardtechniken an. Unser Ansatz sieht vor, dass ein Entwickler, der einen neuen Export/Import hinzufügen will, zu diesem Zweck ein oder mehrere XSLT-Skripts bereitstellt. Um dies zu realisieren, sind natürlich noch immer Kenntnisse über das interne (XMI-)Format des Editors notwendig. Diese Forderung entfällt jedoch, wenn bereits Konvertierungen von bzw. in ein anderes, einfacher zu verstehendes Austauschformat vorhanden sind.

Zur Illustration wollen wir darstellen, wie unsere Techniken in das Geschäftsprozess-Modellierungswerkzeug *bflow* Toolbox* eingebunden wurden. Dieses Werkzeug dient der Modellierung von Geschäftsprozessen in der Notation der Ereignisgesteuerten Prozessket-

ten. Für diese existiert bereits ein Vorschlag für ein offenes Austauschformat, das ebenfalls XML-basierte EPML [MN02]. Nachdem von uns nun eine Transformation des *bflow* Toolbox* -eigenen XMI-Formates in EPML zur Verfügung gestellt wurde, müssen Entwickler in der Folge lediglich ein XSLT-Skript bereitstellen, um das eigene Dateiformat in das EPK-Austauschformat EPML zu transformieren (oder umgekehrt). Die Einbindung eines neuen Exports/Imports erfolgt dann einfach, indem das XSLT-Skript zu den schon vorhandenen Transformationen hinzugefügt wird. Weiterhin muss der neue Export/Import in einer Konfigurationsdatei beschrieben werden. Vorteil dieser Lösung ist, dass zum Einbinden eines neuen Export/Importformates keinerlei Eclipse-spezifischen Entwicklerkenntnisse nötig sind.

Dem steht nicht entgegen, dass Entwickler mit tiefergehenden Eclipse-Kenntnissen oft die Benutzung einer Transformationssprache wie QVT oder oAW XTend für die Behandlung neuer Exportformate bevorzugen werden – unser Ziel ist es, zusätzliche einfache Schnittstellen anzubieten, die ganz ohne Compilieren des Editors auskommen.

Abb. 1 zeigt die aktuell realisierten Exporte für die *bflow* Toolbox* . Jeder Pfeil im Diagramm entspricht der Anwendung einer (oder mehrerer) XSLT-Transformationen. Durch den einfachen Mechanismus konnte die *bflow* Toolbox* um Export- und Importmöglichkeiten erweitert werden, die nach unserem Wissen derzeit kein vergleichbares Werkzeug (kommerziell oder frei) anbietet.

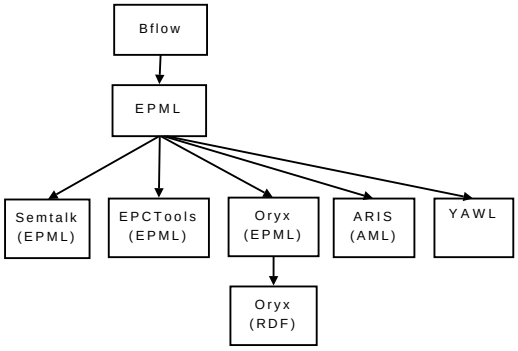

Abbildung 1: Exportformate (ein ähnliches Bild ergibt sich für die Importe.)

4 Hinzufügen von Attributen zu Modellelementen

Mitunter ist es wünschenswert, einem Modell oder einzelnen Modellelementen zusätzliche Informationen in Form selbst definierter Attribute hinzuzufügen. Beim Beispiel von Geschäftsprozessmodellen könnten dies etwa Informationen über Ausführungszeiten oder Angaben für die Prozesskostenrechnung sein. Zwar ist das Hinzufügen neuer Attribute auf Basis des EMF-Metamodells möglich; auch hier ist aber wiederum einen gewisser Einarbeitungsaufwand nötig. Um diesen zu umgehen, bieten unsere Plugins die Möglichkeit, zusätzliche Attribute durch Menüeingaben zum Modell hinzuzufügen. Diese eigenen At-

ISBN 978-3-88579-286-4

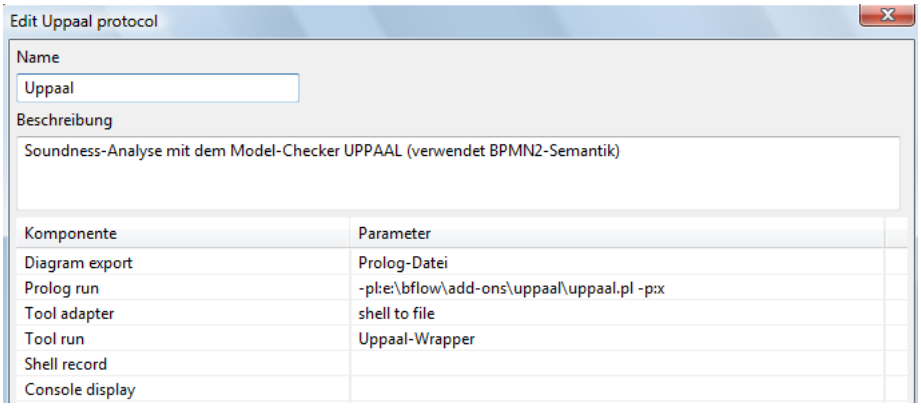

Abbildung 2: Konfiguration eines Protokolls

tribute werden nicht Bestandteil des Metamodells, werden aber dennoch beim Serialisieren des Modells in die gespeichere Datei übernommen. Sie sind somit für eine Weiterverarbeitung verfügbar. Automatisch wird dabei das Attribut „ist markiert“ für alle per Maus ausgewählten Modellelemente gesetzt, auch auf diese Information können externe Programme zurückgreifen.

5 Einbinden externer Werkzeuge als Add-ons

Ziel unseres Add-on-Mechanismus ist es, die in Abschn. 2 genannten Einstiegshürden zu vermeiden. Der Entwickler erhält Schnittstellen, um ein externes Programm (oder eine Kette von externen Programmen) aus der Editor-Umgebung heraus aufzurufen, ohne sich um Interna der Eclipse-Programmierung kümmern zu müssen.

Die Einbindung eines externen Programms erfolgt mittels eines sog. Protokolls. In einem solchen Protokoll wird konfiguriert, welche Arbeitsschritte nacheinander aufgerufen werden. Für typische Aufgaben stehen vordefinierte Bausteine bereit, die mit einem graphischen Werkzeug zu einem Protokoll zusammengefügt werden können. Dieses Zusammenfügen der vordefinierten Bausteine erfolgt im Preferences-Menü von Eclipse; ein Beispiel für ein konfiguriertes Protokoll zeigt Abb. 2. Definierte Protokolle können gespeichert und geladen werden, was es einem Anwender nochmals erleichtert, ein Protokoll in den eigenen Editor einzubinden.

Vergegenwärtigen wir uns an dieser Stelle noch einmal die in Abschn. 2 genannten Probleme, die bei der Einbindung eines externen Werkzeugs zu bewältigen sind: Zunächst müssen die Modelldaten des aktuellen Modells ausgelesen und dem externen Werkzeug in dessen Eingabeformat zur Verfügung gestellt werden. Danach ist das externe Werkzeug zu starten. Schließlich sind die vom externen Werkzeug gelieferten Ausgaben an die Editor-Oberfläche zurückzuliefern. Im Folgenden soll dargestellt werden, wie unser Add-on-Mechanismus diese Anforderungen unterstützt.

ISBN 978-3-88579-286-4

5.1 Bereitstellen der Modelldaten

Um die Daten des aktuell bearbeiteten Modells bereitzustellen, wird das in Abschnitt 3 beschriebene Verfahren zur Definition eigener Exporte genutzt. Ein Export in ein Format, für das eine XSLT-Transformation vorliegt, lässt sich als „Baustein“ in die Kette der aufzurufenden Schritte einfügen (vgl. erste Zeile in der unteren Tabelle von Abb. 2).

5.2 Starten des externen Werkzeugs

Das einzubindende Werkzeug kann ein beliebiges lokal installiertes Programm sein. In einem Preferences-Menü von Eclipse kann festgelegt werden, welche Programme für einen Aufruf zur Verfügung stehen und welche Eingabeparameter sie erwarten. Nachdem ein Programm dem Editor auf diese Weise bekannt gemacht wurde, kann es durch Aufruf des Bausteins „Tool run“ in einem Protokoll angesprochen werden. In Abb. 2 wird ein Protokoll gezeigt, das ein zuvor definiertes Programm namens *Uppaal-Wrapper* aufruft. Dieses bekommt (ähnlich dem Prinzip der UNIX-Pipes) die Ausgaben der vorherigen Arbeitsschritte als Eingabeparameter übergeben. Somit ist es auch möglich, mehrere externe Werkzeuge nacheinander aufzurufen und das Gesamtergebnis an den Editor zurückzuliefern.

Mit der Definition eines Protokolls wird ein Menüeintrag zum Starten des Protokolls automatisch generiert (vgl. Abb. 3). Somit sind keine Eclipse-Kenntnisse nötig, um den Start des externen Programms aus dem Editor heraus zu ermöglichen.

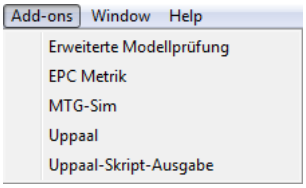

Abbildung 3: Automatisch generierte Menüeinträge für definierte Add-Ons

5.3 Zurückführen der Ergebnisse

Hat nun das aus dem Editor heraus gestartete externe Programm seine Arbeit verrichtet, sollen in der Regel Informationen an die Benutzeroberfläche des Editors zurückgeliefert werden. Hierfür werden verschiedene Bausteine angeboten:

Die *Ausgabekonsole* stellt die einfachste Form der Rückgabe dar: Die Ausgaben des externen Programms werden im Konsole-Fenster des Editors angezeigt.

Der Baustein *Problems View Display* ermöglicht die Ausgabe von Meldungen in der Problems-View von Eclipse sowie gleichzeitig das Markieren eines Modellelements. Die an-

gesprochene Werkzeugkette muss hierfür eine Textausgabe in einem vorgegebenen Format auf die Standardausgabe ausgeben. Dieses wiederum von UNIX-Pipes motivierte Verfahren stellt die denkbar einfachste Schnittstelle zwischen externem Programm und der Anzeige im Editor dar:

```
addon*[INFO][_CPxEEC2ZEd6FEP6IOyuujA][doppelter Knotenname!]#FS#
```

Im zweiten Klammerpaar muss hierbei die eindeutige ID des zu kennzeichnenden Elements stehen. Diese kann z.B. beim Export in die exportierte Datei eingefügt werden und steht folglich dem einzubindenden Werkzeug zur Verfügung.

Der Baustein *Attribute Adjust* gestattet es schließlich, Attribute von Modellelementen (z.B. Knotennamen) zu ändern. Dies ist sowohl für die im Metamodell hinterlegten Attribute möglich wie auch für die selbst hinzugefügten Attribute (vgl. Abschn. 4). Unterstützt werden weiterhin verschiedene graphische Attribute eines Modellelements. Es ist somit z.B. möglich, die Lage eines Modellelements sowie die zur Beschriftung verwendete Schriftart zu modifizieren. Die Informationen über die vorzunehmenden Attributänderungen werden wiederum über Schreiben auf die Standardausgabe an den Editor zurückgeliefert.

6 Anpassbare Validierungen

Validierung von Modellen zur Laufzeit ist ein weiterer Schwerpunkt, der von unserer Lösung unterstützt wird.

So kann beispielsweise festgelegt werden, welche Konstrukte als syntaktisch korrekt zugelassen sind; Fehler werden angezeigt. Bei komplexeren Modellierungssprachen ist es häufig der Fall, dass syntaktische Vorschriften über die Einschränkungen hinausgehen, die das EMF-Metamodell liefert. So sind oft Zyklen zwischen verschiedenen Modellelementen verboten (man denke an zyklische Vererbungsbeziehungen in einem UML-Klassendiagramm).

Wir haben einen Mechanismus geschaffen, in dem zusätzliche Eigenschaften des Modells zur Modellierungszeit überprüft werden können. Abb. 4 zeigt die Oberflächen zweier Editoren, die mit Hilfe unserer Plugins um solche Prüfungen erweitert wurden.

Im Falle des Geschäftsprozess-Modellierungswerkzeugs *bflow* Toolbox* werden durch diese Analyse auch inhaltliche Modellierungsprobleme erkannt [GLKK09]. Eigene Validierungsregeln (etwa zur Prüfung von Modellierungskonventionen) können leicht hinzugefügt werden. Zum Einsatz kommen die Validierungssprachen Check und Epsilon. Weiterhin sind nahezu beliebige Analysen des Modells mittels Prolog möglich, auch Beschriftung und Modell-Layout können in diese Analysen einbezogen werden [GL10, GL11]. Der Modellierer erhält eine unmittelbare Rückmeldung über Probleme im Modell und kann diese Probleme sofort beheben. Eine experimentelle Untersuchung belegt, dass eine solche Modellvalidierung im Hintergrund die Qualität der Modelle positiv beeinflussen kann [LKG09].

ISBN 978-3-88579-286-4

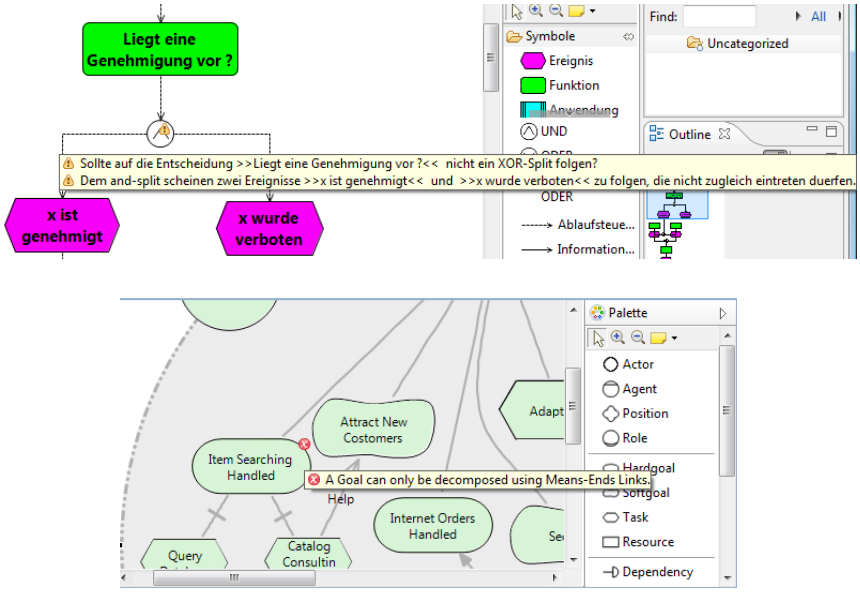

Abbildung 4: Validierung in den Editoren *bflow* Toolbox* und *openOME*

Die Einbindung des Prolog-Interpreters ist ein Beispiel für die bisher vorgestellten Mechanismen: Die Übersetzung des aktuellen Modells in Prolog-Regeln ist als Export wie in Abschn. 3 beschrieben per XSLT-Skript implementiert; der Prolog-Interpreter ist selbst als externes Werkzeug nach dem beschriebenen Verfahren eingebunden, und das aufgerufene Prolog-Programm ist als Add-On realisiert.

7 Fazit

Die in diesem Artikel vorgestellten Mechanismen reduzieren deutlich den Aufwand, Erweiterungen zu einem EMF/GMF-basierten graphischen Editor hinzuzufügen. Hervorzuheben ist, dass für keine der beschriebenen Erweiterungen der Editor neu compiliert werden muss. Wir folgen somit der Idee des End-User-Programming [Jon95]: Es werden einfache Schnittstellen geliefert, um „kleine“ Anpassungen unkompliziert zu erledigen. Dieser Ansatz steht nicht in Konkurrenz zu modernen Programmiermethoden, sondern ergänzt diese. Unser Ziel ist es nicht, Werkzeuge für Eclipse-Entwickler Verfügung zu stellen; unser Ansatz hat diejenigen im Blick, die wegen der steilen Lernkurve im Normalfall gar nicht entwickeln würden.

Unser Ansatz (technisch: die Eclipse-Plugins) wurde bisher integriert in das Geschäftsprozess-Modellierungswerkzeug *bflow* Toolbox* sowie in den Editor *openOME*, in dem Goal-Modelle in der Sprache i^* erstellt werden können.

Interessierte Entwickler sind zur Nutzung, Mitarbeit und Weiterentwicklung eingeladen;

Quelltexte und Dokumentation finden sich auf auf den Projektseiten der *bflow* Toolbox* , *www.bflow.org*.

Literatur

[GL10] Volker Gruhn und Ralf Laue. A Heuristic Method for Detecting Problems in Business Process Models. *Business Process Management Journal*, 16(4), 2010.

[GL11] Volker Gruhn und Ralf Laue. Detecting Common Errors in Event-Driven Process Chains by Label Analysis. *Enterprise Modelling and Information Systems Architecture*, 1(1):3–15, 2011.

[GLKK09] Volker Gruhn, Ralf Laue, Stefan Kühne und Heiko Kern. A Business Process Modelling Tool with Continuous Validation Support. *Enterprise Modelling and Information Systems Architecture*, 4(2), Dec 2009.

[IBM04] IBM Redbooks. *Eclipse Development Using the Graphical Editing Framework And the Eclipse Modeling Framework*. IBM, 2004.

[Jon95] Capers Jones. End-User Programming. *Computer*, 28:68–70, 1995.

[LKG09] Ralf Laue, Stefan Kühne und Andreas Gadatsch. Evaluating the Effect of Feedback on Syntactic Errors for Novice Modellers. In *EPK 2009, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, CEUR Workshop Proceedings, 2009.

[MN02] Jan Mendling und Markus Nüttgens. Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen zur Definition eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK). In *EPK 2002, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, Seiten 87–93, 2002.

[SBPM09] David Steinberg, Frank Budinsky, Marcelo Paternostro und Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2nd. Auflage, 2009.