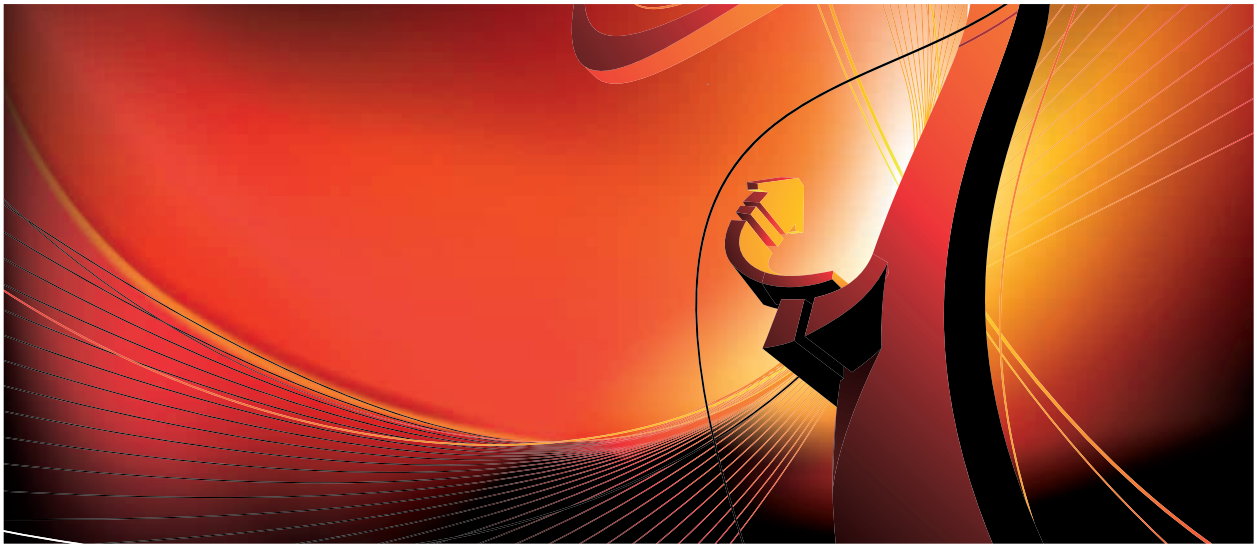

Anforderungsmanagement mit RMF und ProR

Requirements Modeling Framework

Im August 2011 hat das Requirements Modeling Framework (RMF) als ein neues Eclipse-Projekt das Licht der Welt erblickt [1]. RMF besteht aus einem Kern, der Daten im Requirements Exchange Format (RIF/ReqIf) verarbeiten kann, und einem GUI namens ProR, mit dem sich diese Daten komfortabel bearbeiten lassen. ProR stellt einen Extension Point zur Verfügung, der die Integration mit anderen Werkzeugen ermöglicht.

von Michael Jastram und Andreas Graf

In diesem Artikel beschreiben wir zunächst die Entwicklungen im Bereich Anforderungsmanagement und geben Hintergrundinformationen zum RIF/ReqIf-Format. Wir zeigen dann konkret, wie man mit ProR Anforderungen bearbeiten kann. Im letzten Teil beschreiben wir, wie man ProR mit so genannten Präsentationen programmatisch erweitert.

Technische Aspekte

Der wichtigste technische Grund für das Fehlen einer etablierten Open-Source-Umsetzung war bisher das Fehlen eines akzeptierten Standards für die Modellierung und den Austausch von Anforderungen. Eine Internetsuche nach „Open Source Requirements Management“ liefert zwar ein paar Ergebnisse, doch darunter finden sich auch Projekte, deren aktueller Status nicht klar ist. Requirements-Werkzeuge müssen sich auch in eine Werkzeuglandschaft integrieren. Ein Open-Source-Werkzeug per se ist nicht nützlich, wenn es ähnliche Integrationsprobleme verursacht wie kommerzielle Werkzeuge. Wir erwarten, dass sich die Situation durch die Veröffentlichung des ReqIF-Standards [2] stark verändert, da wir Parallelen zur Geschichte der UML sehen. Bevor die UML veröffentlicht wurde, gab es eine Reihe inkompatibler Modellierungsmethoden (Grady, Booch, ROOM). Nach der Veröffentlichung der UML verwendeten die meisten Bücher, Kurse und Werkzeuge den neuen Standard. Dadurch konnten die Projekte leichter qualifizierte Teams aufbauen, denn die gemeinsame Notation verkürzte Lernzeiten und die Menge an verfügbarer Information senkte die Eintrittsschwelle. Auf der Werkzeugseite eröffnete das gemeinsame Metamodell und die Notation den Markt für preisgünstige Tools und Open-Source-Werkzeuge. So werden beispielsweise

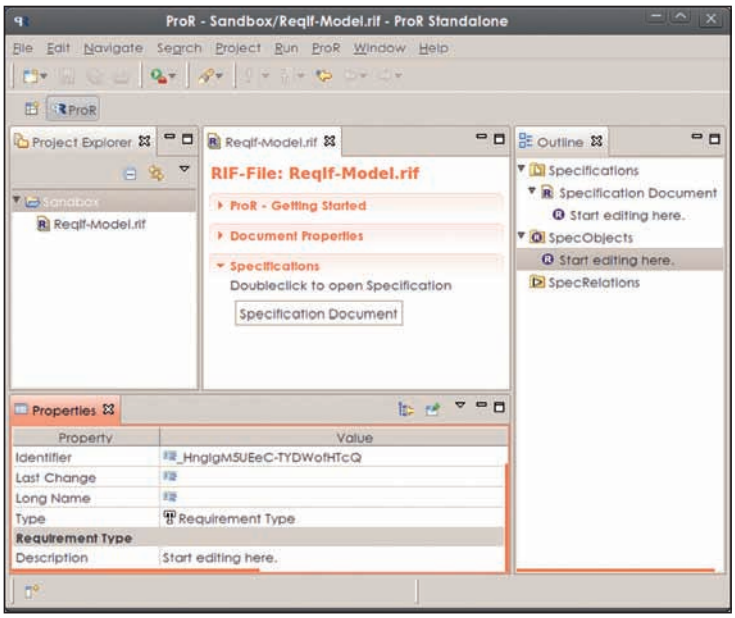

Abb. 1: ProR nach dem Erstellen einer neuen Datei über den ReqiF Wizard

Papyrus und Topcased in industriellen Projekten eingesetzt. Gegenwärtig ist das Angebot an preisgünstigen Tools und Open-Source-Werkzeugen für das Requirements Management nicht sehr umfangreich. Die Situation ist ähnlich wie vor der Einführung der UML. Zwar bietet zum Beispiel die SysML grundlegende Ansätze für das Requirements Engineering, allerdings sind diese dann eng an den Einsatz der SysML gebunden, was nicht für alle Projekte geeignet sein mag. Der ReqIF-Standard ist flexibel genug, um mehrere Ansätze des Requirements Engineerings zu unterstützen, dabei ist er konkret genug, um eine Werkzeugimplementierung zu ermöglichen. Sowohl hinter ReqIF als auch UML steht die OMG. Dadurch sind die Chancen für eine Akzeptanz in der Industrie recht hoch.

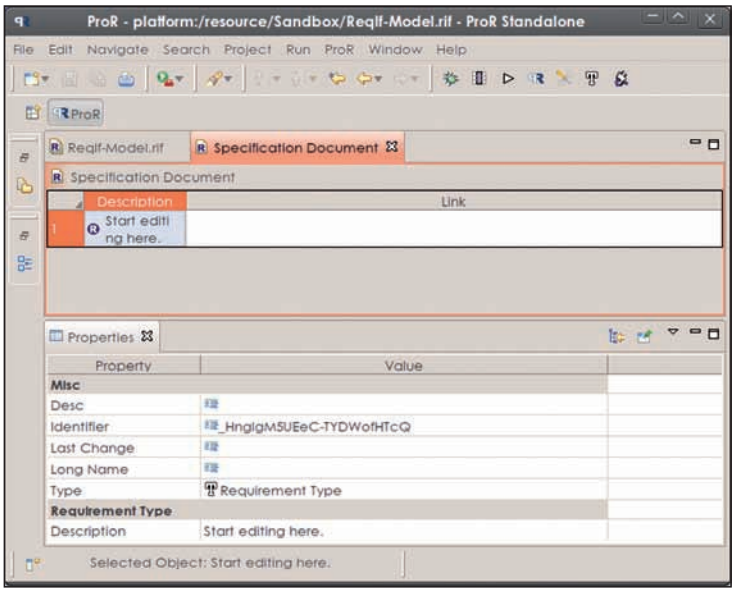

Abb. 2: Die Specification View, in der die Anforderungen tabellarisch bearbeitet werden

Geschäftliche Aspekte

Anforderungswerkzeuge hatten bis jetzt einen Makel, der die Akzeptanz im Open-Source-Bereich erschwert: Sie sind oft das eher ungeliebte Kind im Entwicklungsprozess. Ein Entwickler, der sich in diesem Bereich engagiert, konnte wohl auf deutlich weniger Anerkennung hoffen, als ein Entwickler im Bereich Modellierung oder Entwicklungsumgebungen. Das ändert sich gegenwärtig, da immer mehr Unternehmen aus den Industrien, die mit klassischem Anforderungsmanagement arbeiten, auf Eclipse als Werkzeugplattform setzen und zunehmend die Vorteile von Open Source erkennen. So entwickelt die französische Luft- und Raumfahrtindustrie das Systems-Engineering-Werkzeug Topcased, und die Automobilindustrie koordiniert ihre Aktivitäten in der Eclipse Automotive Working Group. Gerade in internationalen Projekten, die sicherheitskritische Systeme entwickeln, ist klassisches Anforderungsmanagement erforderlich. Das große Interesse zeigt sich auch an den Firmen, die sich für das RMF-Projekt als interessierte Parteien gemeldet haben. Als Endanwender sind hier zum Beispiel Airbus oder Thales zu nennen.

Requirements Interchange Format

Die Geschichte von ReqIF beginnt 2004, als die HIS-Automotive, eine Arbeitsgruppe der deutschen Automobilindustrie, ein Austauschformat für Anforderungen unter dem Namen RIF veröffentlichte. Von da an wurde RIF bis zur Version 1.2 weiterentwickelt und von einigen kommerziellen Werkzeugen unterstützt. Ab 2010 wurde die Standardisierung unter dem Dach der OMG weitergeführt und kürzlich als ReqIF veröffentlicht. Obwohl der Standard als Austauschformat geplant war, geht er aus unserer Sicht wegen seiner Flexibilität deutlich darüber hinaus. Er erlaubt nicht nur ein fixes Datenmodell für die Anforderungen, im Austauschformat ist auch die Definition des jeweiligen Datenmodells enthalten. Der Standard hat damit die Mächtigkeit eines (Meta-)Modellierungswerkzeugs. Aus unserer Sicht ist daher die Bezeichnung „Interchange Format“ historisch korrekt, trifft aber den Umfang nicht ganz, denn es ist wichtig, das Metamodell von ReqIF und das konkrete Austauschformat zu unterscheiden. Zwei Parteien tauschen Anforderungen über das XML-basierte Format aus. Das Metamodell kann im Werkzeug aber auch in anderer Form, zum Beispiel in Java-Klassen, implementiert sein.

Ein ReqIf-Modell besteht aus *SpecObjects*, die den Anforderungen entsprechen. Sie haben einen Typ, der bestimmt, welche Attribute die Anforderung hat. Attribute können gängige Datentypen wie Text, Zahlen, Daten, Aufzählungen, Rich-Text haben, aber auch OLE-Objekte und Binärdaten werden unterstützt. *SpecObjects* werden in Specifications hierarchisch organisiert, und es gibt *SpecRelations*, über die *SpecObjects* verlinkt werden. Es gibt noch viele weitere Features, auf die wir hier nicht im Detail eingehen wollen.

Die Umsetzung des ReqIF-Kerns

Da der Standard in Form eines Enterprise-Architekt-Modells beziehungsweise als MOF-Modell vorliegt, war eine erste Umsetzung recht einfach. Wir importierten die Modelle in Eclipse und erzeugten daraus entsprechende EMF-Metamodelle. Der Teufel sitzt hier ein wenig im Detail, da Enterprise-Architekt und der EMF-Generator bei UML-Modellen unterschiedliche Annahmen machen, zum Beispiel über die Zugehörigkeit von Assoziationen. Da ReqIF und EMF an manchen Stellen unterschiedliche Konzepte zur Serialisierung in XML verfolgen, haben wir noch Anpassungen in der Serialisierung vorgenommen. ReqIF referenziert für die Beschreibungen der Anforderungen unter anderem Teile des XHTML-Standards. Auch diese Anteile reichen wir über EMF nach oben, sodass Anwendungen korrekt darauf zugreifen können. RMF unterstützt gegenwärtig RIF1.1a, RIF1.2 und ReqIF1.0.1. Tests mit Exporten aus kommerziellen Werkzeugen bestätigen die Kompatibilität.

Anforderungsmanagement mit ProR

ProR ist der Name des RMF GUI. ProR kann entweder als eigenständige Anwendung heruntergeladen oder über eine Updatesite installiert werden. ReqIf-Dateien können in jedem beliebigen Projekt abgelegt werden und müssen in *.reqif* enden. Wir können ein neues Modell über den ReqIf Wizard anlegen, woraufhin wir den in **Abbildung 1** gezeigten Editor bekommen. Dieser Editor besteht aus den typischen Eclipse Views: links der Projekt-Explorer, rechts die Outline, dazwischen die Editoren und unten die Properties. Im Outline sehen wir bereits die Struktur, die vom Wizard angelegt wurde: Es gibt ein Specification Document, das eine Anforderung enthält („Start editing here"). Da Anforderungen aus der Spezifikation lediglich referenziert werden, gibt es einen eigenen Ordner für Anforderungen (*SpecObjects*). Weiterhin sehen wir einen Ordner für Links (*SpecRelations*).

Um nun mit den Anforderungen zu arbeiten, machen wir die einzige Spezifikation auf, indem wir in dem ReqIf-Editor doppelt darauf klicken. Dadurch öffnet sich ein neuer Editor, der die Spezifikation in einer tabellarischen Sicht zeigt (**Abb. 2**). Jede Zeile stellt eine Anforderung dar, und jede Anforderung kann eine beliebige Anzahl von Attributen enthalten. Welche Attribute eine Anforderung nun wirklich hat, hängt von deren Typ ab. In der Properties View sehen wir, dass die ausgewählte Anforderung vom Typ *Requirements Type* ist, die genau über ein Attribut namens *Description* verfügt. Um nun das Beispiel etwas interessanter zu machen, werden wir die folgenden Schritte durchführen:

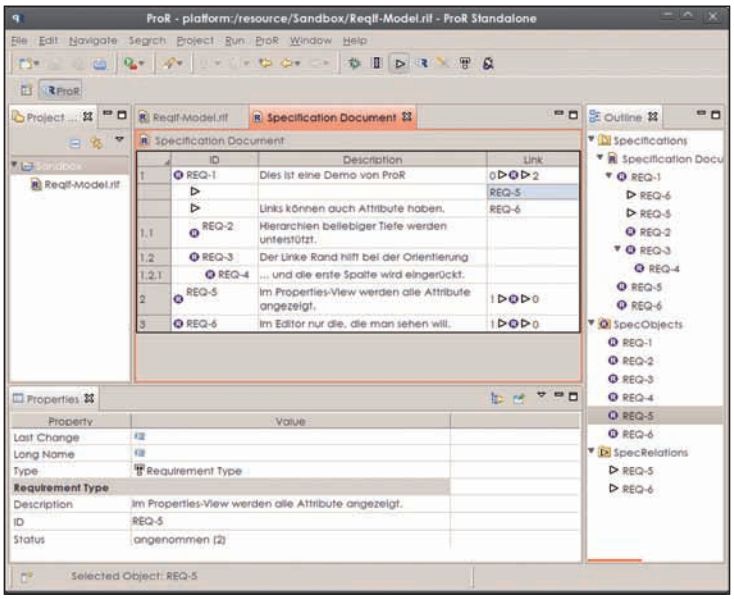

Abb. 3: Ein Beispieldokument mit verschachtelten Anforderungen und Verlinkungen

1. Hinzufügen weiterer Attribute zum *Requirements Type*
2. Anzeigen einer weiteren Spalte im Editor
3. Freischalten der ID-Präsentation, einem Mechanismus zur Vergabe von leserlichen IDs
4. Hinzufügen weiterer Anforderungen
5. Verlinkung von Anforderungen

Das Resultat nach Ausführung dieser Schritte ist in **Abbildung 3** zu sehen.

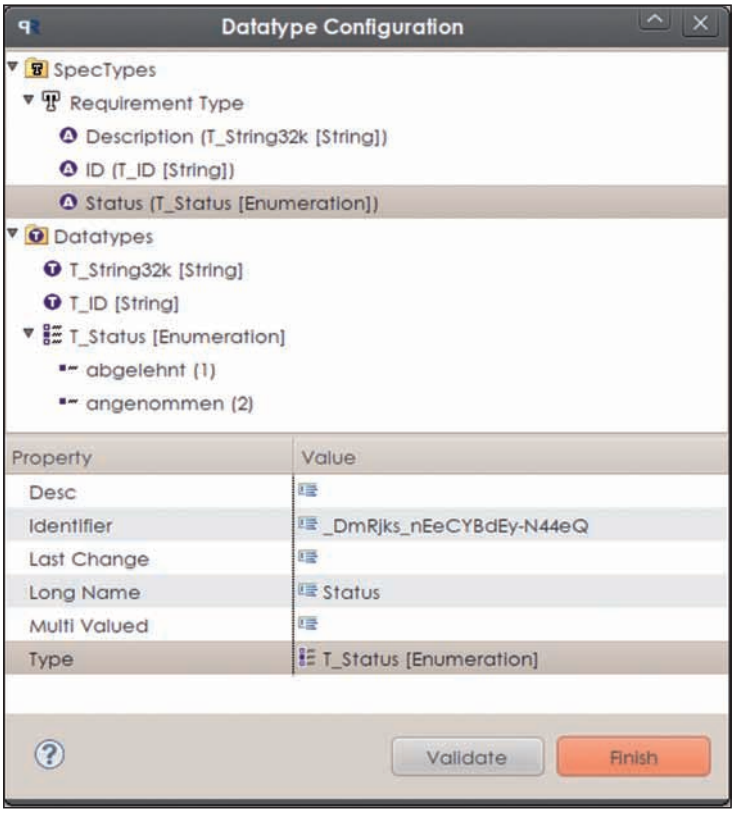

Abb. 4: Der „Datatype Configuration"-Editor

Neue Attribute

Über ProR | Datatype Configuration... öffnen wir den Editor für die Datentypen. Im oberen Teil des Editors sehen wir unsere Datenstrukturen, im unteren werden die Eigenschaften zum Editieren angezeigt. Neue Elemente werden über Kontextmenüs eingefügt. Wir wollen zwei Attribute hinzufügen: ein Feld für eine lesbare ID und ein Feld für Kommentare. Ein Bild sagt mehr als tausend Worte: Nachdem wir alle Attribute und Datentypen angelegt haben, sieht der Datentype Configuration-Editor aus wie in **Abbildung 4**. Der Typ *Requirement Type* hat jetzt zwei neue Attribute. Für ID haben wir einen neuen Datentypen *T_ID* angelegt, wie unter Datatypes zu sehen ist. Für das Attribut *Status* haben wir eine Enumerierung vom Typ *T_Status* erzeugt. Für das selektierte Element im oberen Bereich sind die Eigenschaften im unteren Bereich zu sehen, wo sie auch editiert werden können.

Spalten im Editor anpassen

Wenn wir nun den Dialog schließen und eine Anforderung auswählen, sind in der Properties View drei Attribute zu sehen, die dort auch editiert werden können. Der Editor zeigt allerdings nach wie vor nur eine Spalte an. Über ProR | Column Configuration können neue Spalten hinzugefügt werden. Wir öffnen den Dialog und fügen die Spalte ID hinzu. Mit Drag and Drop kann in diesem Dialog auch die Reihenfolge der Spalten geändert werden, und wir setzten die ID-Spalte an die erste Stelle (**Abb. 3**). Durch diesen Ansatz sehen wir im Editor nur das, worauf wir uns konzentrieren wollen, während die Property View immer alle Attribute anzeigt.

IDs generieren

Nun sehen wir zwar die ID-Spalte, sie enthält aber keine Inhalte. Von Hand wollen wir diese natürlich nicht generieren. ProR kann selbst keine IDs erzeugen, wohl aber eine Präsentation. Präsentationen sind ProR-spezifische Plug-ins, die sowohl die Darstellung als auch den Inhalt von Daten beeinflussen können. Später schauen wir uns an, wie das Ganze programmatisch aussieht, hier wollen wir es lediglich anwenden. Dazu gehen wir auf ProR | Presentation Configuration. Unter Select Action... finden wir ID Presentation. Ein neuer ID-Generator wird angelegt. In den Eigenschaften können wir den Präfix und den Zählerstand der ID anpassen. Am wichtigsten ist aber der Datentyp, mit dem die Präsentation assoziiert werden soll. Dafür wählen wir *T_ID* aus; das ist der Grund, warum wir für die ID eigens einen eigenen Datentyp angelegt haben. Das Ergebnis ist in **Abbildung 5** zu sehen. Wenn wir den Dialog schließen, wird unserer Anforderung eine ID zugewiesen.

Weitere Anforderungen hinzufügen

Nun sind wir soweit, dass wir endlich Daten hinzufügen können. Das geschieht über die Tastatur oder per Kontextmenü. Wenn wir für eine Anforderung das Kontextmenü öffnen, können weitere Elemente über New

Sibling und eingeschobene Elemente über New Child hinzugefügt werden. Eine Spezifikation ist eine Baumstruktur beliebiger Tiefe, und im linken Rand wird über eine entsprechende Nummerierung gezeigt, wo man sich befindet. Außerdem ist die linke Spalte der Hierarchie entsprechend eingerückt.

Über das Kontextmenü können direkt typisierte Anforderungen eingefügt werden, was einem viel Klickerei erspart. Es gibt auch die Möglichkeit, untypisierte Anforderungen oder sogar leere Hierarchieplatzhalter einzufügen. Das kann sinnvoll sein, wenn man eine bestehende Anforderung referenzieren möchte. Dieselbe Anforderung darf also mehrfach auftauchen. Um in der Praxis schnell arbeiten zu können, lassen sich neue Elemente auch per Tastatur, über Strg-Enter, einfügen, und zwar vom selben Typ, auf dem wir uns aktuell befinden. Bleibt nur noch die Anordnung von Anforderungen. Anforderungen können per Drag and Drop verschoben werden, und Copy and Paste funktioniert selbstverständlich auch.

Verlinkung

Die Verlinkung von Anforderungen wird auch über Drag and Drop realisiert, allerdings in Kombination mit der Tastatur, denn einfaches Drag and Drop verschiebt lediglich Anforderungen. Das Tastaturkürzel ist abhängig vom Betriebssystem und ist dasselbe, das auch zum Erstellen von Dateilinks beziehungsweise Shortcuts benutzt wird. Sobald eine Verlinkung angelegt wurde, wird in der letzten Spalte angezeigt, wie viele Links zu der Anforderung hinführen, und wie viele davon ausgehen (**Abb. 3**). Über ProR | SpecRelations können die Verlinkungsobjekte angezeigt werden, und zwar unterhalb der Anforderungen, von denen sie ausgehen. Wenn sie angezeigt werden, zeigt die letzte Spalte den Namen des Zielobjekts, und beim Auswählen dieser Zelle werden die Attribute des Zielobjekts in der Properties View angezeigt. Das ist ebenfalls in **Abbildung 3** zu sehen. Verlinkungsobjekte können übrigens ebenfalls einen Typ haben, wodurch sie Attribute bekommen, die im Editor angezeigt sind, falls es eine entsprechende Spalte gibt (in der Properties View sind natürlich immer alle Attribute zu sehen).

ProR integrieren

Es ist schön und gut, Anforderungen strukturiert zu erfassen, aber der eigentliche Mehrwert kommt durch die Integration von ProR in bestehende Werkzeugketten. Das ist bis zu einem gewissen Grad durch den ReqIf-Standard gegeben. Aber dank Eclipse können wir noch ein gutes Stück weitergehen.

ProR bietet einen Extension Point an, über den so genannte Präsentationen in das GUI und das Modell eingeklinkt werden. Eine Präsentation besteht aus einer Serviceklasse und aus einem Datenelement, das sich in das Modell einklinkt. Wie das in der Praxis aussieht, haben wir ja eben schon bei der ID-Generierung gesehen. Die Serviceklasse bietet viele Möglichkeiten, das GUI

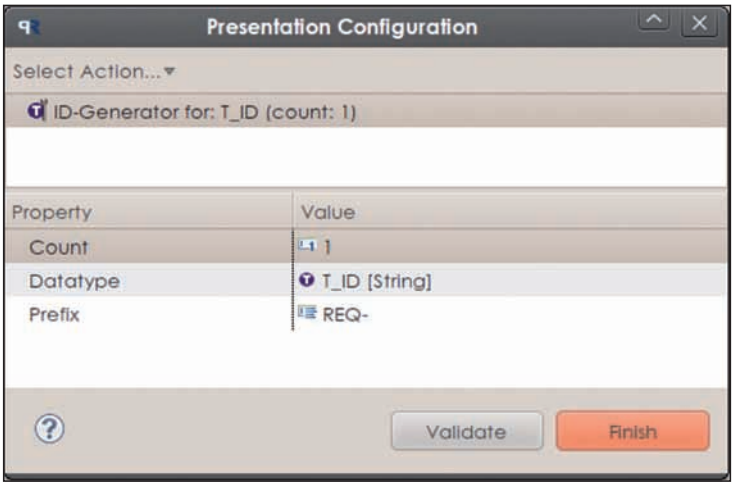

Abb. 5: Die fertig konfigurierte ID-Präsentation

anzupassen und mit anderen Eclipse-basierten Werkzeugen zu integrieren. Die Serviceklasse muss das Interface *PresentationService* implementieren. Oft bietet es sich an, die abstrakte Klasse *AbstractPresentationService* zu überschreiben, die für alle Methoden eine Default-Implementierung zur Verfügung stellt. Das Datenelement erlaubt es der Präsentation, eigene Daten abzulegen. ReqIf hat eigens das Vater-Element *ReqIfToolExtension*, das Werkzeuge beliebig nutzen darf. Präsentationen müssen immer mit einer *DatatypeDefinition* assoziiert sein. Warum das so ist und wie das funktioniert, werden wir gleich sehen. Was kann man mit Präsentationen überhaupt machen? Wir stellen im Folgenden drei einfache Präsentationen vor und beschreiben anschließend, wie sie in ProR realisiert wurden.

HeadlinePresentation: Da im Moment noch kein formatierter Text unterstützt wird, haben wir die *HeadlinePresentation* entwickelt. Dieses erlaubt es, bestimmte Attribute in einer größeren Schrift darzustellen, ohne auf Rich Text zugreifen zu müssen. Dabei wird die Präsentation mit einer *DatatypeDefinition* vom Typ *String* assoziiert. Alle Attribute, die diese *DatatypeDefinition* nutzen, werden dann groß und fett dargestellt.

IDPresentation: Die von ReqIf vergebenen IDs sind für Menschen nicht gut lesbar, für uns ist eine ID wie REQ-27 wesentlich einfacher zu handhaben. Diese Präsentation hatten wir ja bereits vorgestellt.

RodinPresentation: Rodin [3] ist ein Eclipse-basiertes Werkzeug für die formale Modellierung. Wir wol-

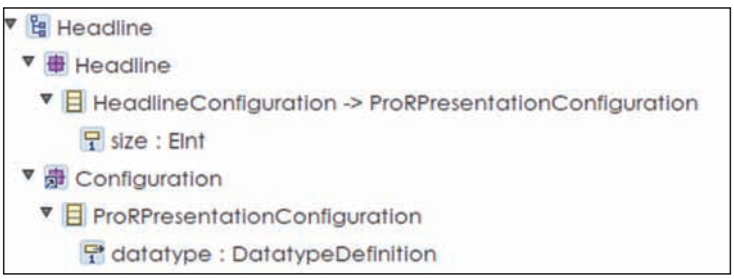

Abb. 6: Das EMF-Modell für die HeadlinePresentation

len auf das Werkzeug nicht im Detail eingehen; wichtig zu wissen ist jedoch, dass Systemelemente mit Variablen und anderen Konstrukten modelliert werden können. Wir wollen bestimmte Konstrukte des Modells mit Anforderungen assoziieren (z. B. eine Variable im Modell mit deren textueller Definition im Anforderungsdokument). Unsere Präsentation erlaubt es, per Drag and Drop solche Verbindungen herzustellen und die Modellelemente direkt im Anforderungsdokument anzuzeigen.

Wie können nun diese Präsentationen mit ProR realisiert werden? Für eine Präsentation wird zunächst ein neues Plug-in angelegt, das den Extension Point *org.eclipse.rmf.pror.reqif10.presentation* implementiert und ein eigenes EMF-Modell hat, das *ProrPresentationConfiguration* erweitert. Fast alles Weitere wird durch die Implementierung der Serviceklasse geregelt, die das Interface *PresentationService* implementiert. Schauen wir uns einmal einige ausgewählte Methoden dieses Interface an: *IProrCellRenderer getCellRenderer(AttributeValue av);*. Diese Methode erlaubt es, einen alternativen *CellRenderer* zu implementieren (um den Default Renderer zu nutzen, wird *null* zurückgegeben). Wir überschreiben diese Methode für die *HeadlinePresentation* und stellen einen Renderer zu Verfügung, der die Schrift anpasst. Das System garantiert, dass der übergebene *AttributeValue* die korrekte *DatatypeDefinition* hat, die über das Modellelement *HeadlineConfiguration* konfiguriert wurde. Schauen wir uns nun die *HeadlineConfiguration* genauer an. **Abbildung 6** zeigt den entsprechenden Teil des Modells. Wir sehen, dass *HeadlineConfiguration* ein neues Attribut aufweist (*size*). Da *HeadlineConfiguration* von *ProRPresenationConfiguration* erbt, hat es ebenfalls das Attribut *datatype*, über das wir die relevanten *AttributeValues* identifizieren. Nun zu weiteren Methoden von *PresentationService*:

```
CellEditor getCellEditor(AgileGrid agileGrid, EditingDomain editingDomain,
                                                    AttributeValue av);
```

Ähnlich wie beim *CellRenderer* kann man auch einen *CellEditor* einklinken. Wir können beispielsweise einen Xtext-basierten Editor einbinden, sodass beim Editieren Syntax Highlighting, Autocompletion und ähnliche Features zur Verfügung stehen. Wie beim Renderer kann der Default-Editor benutzt werden, indem *null* zurückgegeben wird. Die Methode *public void openReqIf(ReqIf reqif)* erlaubt einer Präsentation, sich direkt beim Öffnen des ReqIf-Modells in dieses einzuklinken. Wir nutzen das bei unserer *IdPresentation*, um bei neu eingefügten Elementen eine neue ID zu vergeben. Sobald die Präsentation das ReqIf-Objekt erhalten hat, kann man die normalen EMF-Mechanismen benutzen (*Notifier*, *Adpater* etc.):

```
public Command handleDragAndDrop(Collection<?> source, Object target,
                                  EditingDomain editingDomain, int operation)
```

Um eine wirklich intuitive Integration in der Eclipse-Umgebung zu realisieren, ermöglichen wir es Präsentationen, Drag-and-Drop-Operationen zu bedienen. Der zurückgegebene Wert ist ein EMF *Command*, der dann von ProR ausgeführt wird. Wir manipulieren unser Modell über Commands, damit *Undo*, *Redo* und ähnliche Funktionen wie gewohnt funktionieren. Wir benutzen diesen Mechanismus zum Beispiel für die Rodin-Integration. Modellelemente werden in einer Baumstruktur dargestellt und können direkt in den ReqIf-Editor gezogen werden, wo sie dann entweder visualisiert oder verlinkt werden. Um diese Elemente überhaupt darstellen zu können, enthält das ReqIf-Attribut lediglich die ID des referenzierten Elements. Der Renderer der Präsentation findet dann den entsprechenden Inhalt und stellt ihn grafisch dar.

Fazit

Wir hoffen, mit RMF einen wertvollen Beitrag im Bereich Open-Source-Werkzeuge zum Anforderungsmanagement geleistet haben; das bisherige Interesse bestätigt das. Das Projekt ist zurzeit mächtig in Bewegung. Wir haben gerade die Eclipse-Proposal-Phase beendet (August 2011) und sind nun dabei, die Migration von Code und Entwicklungsinfrastruktur zur Eclipse Foundation durchzuführen. Parallel entwickeln wir das Werkzeug weiter. Daher ist es gut möglich, dass zum Zeitpunkt der Veröffentlichung sich bereits einiges geändert hat: Namen werden angepasst, Code wird umstrukturiert, neue Features implementiert. Aktuelle Informationen gibt es auf unserer Eclipse-Seite [1]. Und wir freuen uns auf viele Rückmeldungen im Eclipse-Forum [4].

Michael Jastram ist Gründer und Geschäftsführer der Formal Mind GmbH, einem Spin-Off der Universität Düsseldorf, wo er zurzeit auch promoviert. Sein Forschungsschwerpunkt ist die Integration und Nachverfolgbarkeit zwischen Anforderungen und formalen Spezifikationen. Er ist Gründer und Vorsitzender der Düsseldorfer Java User Group (rheinjug e.V.) und Mitverfasser des Buchs „Eclipse Rich Client Platform“ (2008). Bevor er zwei Jahre für die HOOD Group als Abteilungsleiter der Software Division tätig war, hat er zehn Jahre lang als Softwareentwickler im US-amerikanischen Umfeld Erfahrung gesammelt. Er machte 1997 einen Doppelabschluss mit einem Dipl.-Ing. von der Universität Hamburg und einen Master of Science vom Massachusetts Institute of Technology in Cambridge, USA.

Andreas Graf ist Business Development Manager Automotive bei itemis.

Links & Literatur

[1] http://eclipse.org/rmf/

[2] http://www.omg.org/spec/ReqIF/

[3] http://sourceforge.net/projects/rodin-b-sharp/

[4] http://eclipse.org/forums/eclipse.rmf