Werkzeuge einer MDSD-Entwicklungsumgebung für große Softwareproduktlinien

Steffen Stundzig und Alexander Nittka

itemis AG
Ludwig-Erhardt-Str. 51
04103 Leipzig
{Steffen.Stundzig|Alexander.Nittka}
@itemis.com

Steffen Skatulla, Martin Schmidt,
Detlef Hornbostel und Michael Hörseljau

IBYKUS AG
Herman-Hollerith-Str. 1
99099 Erfurt
{Steffen.Skatulla|Martin.Schmidt|
Detlef.Hornbostel|Michael.Hörseljau}
@ibykus.com

Abstract: Die modellbasierte Entwicklung großer, sich schnell erweiternder, Produktlinien erfordert eine angepasste Entwicklungsumgebung. Diese muss neben Mehrbenutzerfähigkeit viele spezifische Funktionalitäten hinsichtlich frühzeitiger Validierung, On-/Offlineentwicklung, Mehrsprachigkeit, Erweiterbarkeit, einfacher Benutzbarkeit, aufgabenbasiertem Arbeiten, Versionierung von Teilmodellen, verschiedener Sichten auf einzelne Modellaspekte und stark abgesichertes Ändern des Gesamtmodells haben. Wird außerdem das Ziel verfolgt, Fachexperten beim Endkunden zu erlauben, selbst an den Modellen und somit an der laufenden Software Änderungen vorzunehmen, muss auch die Entwicklungsumgebung sehr robust und intuitiv sein.

1 Motivation

In diesem Vortrag wird auf Erfahrungen verschiedener Kundenprojekte zurückgegriffen. Hauptsächlich geht es aber um ein Projekt bei der IBYKUS AG.

2 Umgebung

Die hauseigene Entwicklungsplattform der IBYKUS AG mit der Bezeichnung AP® sollte durch eine neu zu Entwickelnde ersetzt bzw. ausgebaut werden. Zielgruppe der Umgebung sind DV-Mitarbeiter mit Fachkenntnissen, die Fachanforderungen selbst als DV-Lösung umsetzen. Die IBYKUS AG begann Mitte der 90'er Jahre mit modellbasierter Entwicklung. Damals steckte die Methodik innerhalb der Wirtschaft noch in den Kinderschuhen, weshalb sich bei der IBYKUS AG eine eigene Begriffswelt für die verschiedenen Artefakte entwickelte. Ein Mapping dieser Begriffswelt auf Begriffe der mittlerweile recht großen Community war eine der ersten Herausforderungen.

Die Entwicklung geschieht bei der IBYKUS AG vollständig modellbasiert, momentan textuell und perspektivisch auch grafisch. Die Metamodelle orientieren sich an sogenannten Konfigurationsräumen. Ein Konfigurationsraum definiert ein Metamodell und eine konkrete Syntax für Typen und Attribute. Zusätzliche Validierungsregeln ergeben sich zum Teil aus den Attributen, z. B. erlaubte Attributtypen oder erlaubte Referenzzieltypen.

Ziel der Entwicklungsumgebung ist es nun, intuitive Editoren zu erstellen, die es erlauben, nur anhand der Definition eines Konfigurationsraumes, die Modellierung einer konkreten Anwendung zu unterstützen. Die Entwicklungsumgebung soll auf Basis der Eclipse Plattform als Eclipse-Plugins umgesetzt sein, um sich einfach in die vorhandene Umgebung zu integrieren. Das Eclipse Ecosystem, das besonderen Augenmerk auf die Entwicklung von Frameworks und Plattformen legt, ist dafür sehr gut geeignet, siehe [ECL].

Als Backend zum Speichern der Modelle wird ein relationales Datenbanksystem als Modellrepository, kurz DVR, eingesetzt. Vor dieses Repository ist eine spezifische Schnittstelle geschalten, die aufgabenbasiertes Arbeiten und Prozesse ähnlich einem Sourcecode-Configuration-Management-System unterstützt. Die Modelle selbst werden dabei als XML-Datenströme in das Repository transportiert und von dort geladen.

Als Editoren in der vorhandenen Umgebung wurden bisher einfache Text- und XML-Editoren mit angepasstem rein strukturbasierten Syntaxhighlighting eingesetzt. In der neuen Umgebung sollen diese Editoren mit einem Modell- und Repositorynavigator direkt integriert werden, um somit ein Umschalten zwischen verschiedenen Applikationen zu vermeiden. Der Navigator selbst soll verschiedene Sichten auf die Modellstrukturen und Teilmodelle zulassen, ohne die tatsächliche interne Struktur des Gesamtmodells zu kennen. Diese Sichten bilden Aspekte wie Dokumentation, Quellcode, Komponentenschnitt u. ä. ab. Auch alle Funktionen zum Laden und Speichern im DVR sollen aus dem Kontextmenu des Editors erreichbar sein.

Die Modelle liegen im DVR mit Historie in verschiedenen parallelen Ländervarianten vor. Ein Modell enthält dabei durchschnittlich 20.000 Hauptmodellelemente wie Sachverhalte, Parameter, Methoden, Kommandos, Bearbeitungsabläufe, Prozessschritte usw. und ca. 200.000 abhängige Modellelemente wie Attribute an Parametern usw. Diese Modelle werden parallel von mehreren Entwicklern bearbeitet. Dabei werden Teilmodellversionen für die Bildung von Releases oder weiteren Entwicklungssträngen selektiv freigegeben.

Diese Modelle basieren auf verschiedenen Metamodellen, die durch die oben genannten Konfigurationsräume inklusiver Transformationsregeln beschrieben werden.

3 Anforderungen

Der Kunde selbst ist Softwarehersteller, der Endkunden mit seiner fachspezifischen Software beliefert. Daraus resultieren Anforderungen hinsichtlich der Paketierung der Komponenten, damit diese in das Gesamtsoftwarepaket konfiguriert werden können.

Dabei wird Software für verschiedene Endkunden und Branchen erstellt. Dies führt zur Anforderung mehrere verschiedene Metamodellvarianten, hier entsprechend Konfigurationsräume genannt, zu erlauben. Um die Dynamik in der fachlichen Softwareentwicklung nicht zu behindern, kommt die Anforderung hinzu, dass die Konfigurationsräume durch IBYKUS selbst ohne Programmieraufwand geändert werden können, was sich wiederum in den Modelleditoren niederschlagen muss.

Zur kundenspezifischen Erweiterbarkeit sollen definierte Erweiterungspunkte existieren, an denen die Infrastruktur kundenseitig erweitert und angepasst werden kann. Der Navigator selbst soll über verschiedene Konfigurationsoptionen direkt über das DVR konfigurierbar sein. Somit ist sichergestellt, dass IBYKUS selbst Einfluß auf Aussehen und Funtionalität der Werkzeuge nehmen kann, ohne eine Zeile Quellcode zu schreiben oder Programmieraufwände an Editor und Navigator zu haben.

Neben den textuellen Modellen sollen auch unstrukturierte Daten wie Microsoft-Word Dokumente, PDF Dateien oder Grafiken mit dem Editor transparent für den Nutzer bearbeitet werden können.

Weitere Anforderungen kommen bzgl. Refactoring von Modellen, Autoformatieren von Modellen, Erkennen von Änderungen an Modellen gegenüber der aus dem Repository geladenen Variante usw. hinzu.

4 Umsetzung

Um die Anforderungen umzusetzen, kamen die Eclipse Platform [ECL], das Eclipse Modeling Framework [EMF], das Programming Language Framework Xtext [XTEX] und die Modeling Workflow Engine [MWE] zum Einsatz.

Eclipse als Plattform bietet Konzepte wie bspw. Extension Points, um kundenspezifische Erweiterungen zu erlauben. So wurden die Erweiterungspunkte der eigenen Plattform definiert. Die Plattform selbst wurde ebenfalls über die Erweiterungspunkte und das OSGi Plugin Konzept aufgebaut, um Teile austauschbar zu halten und eine sinnvolle Komponentisierung zu erreichen. Für den Kunden gibt es dafür zum Beispiel einen Erweiterungspunkt für länderspezifische Textbausteine und länderabhängige Zeichensätze wie polnisch oder russisch. Somit ist IBYKUS in der Lage, die Editoren für Endkunden spezifisch zu paketieren und auszuliefern.

Wie oben beschrieben gibt es auch mehrere Konfigurationsräume die von Kundenseite änderbar sind. Zu jedem Konfigurationsraum wird ein entsprechend spezialisierter textueller Editor Unterstützung in Form von Syntaxhighlighting, Autovervollständigung, Modellvalidierung zur Editierzeit, Finden und Annavigieren von Referenzen auf Modellelemente, Kurzübersicht zur leichten Navigierbarbeit, Refactoring und Autoformatierung bieten.

Als Basis für diese Editoren eignet sich das Framework Xtext, siehe [XTEX], sehr gut. Mit diesem Framework können, passend zu einer kundenspezifischen Grammatik und einem vorgegeben Metamodell, funktionale Texteditoren mit den gewünschten Funktionen erstellt werden. Darauf aufbauend wurde ein generischer Editor realisiert, in den die konfigurationsraumspezifischen Eigenschaften wieder über Erweiterungspunkte injiziert werden.

Die Konfigurationsraumdefinition selbst wird nun aus dem DVR als XML-Datenstrom ausgelesen und mittels Model-Text-Transformation in Implementierungen genau dieser Erweiterungspunkte übersetzt. Durch diesen Mechanismus ist es möglich, für jeden Konfigurationsraum einen spezifischen Editor zu haben, der die grundlegenden allgemeingültigen Funktionen aller Editoren erbt. Durch die automatisierte Transformation geschieht dies ohne eine Zeile Quellcode zu schreiben.

Zur Editierzeit der Modelle wird anhand von Metainformationen im Modell erkannt, welchem Konfigurationsraum dieses Modell angehört und automatisch der passende Editor mit den spezifischen Funktionen und Validierungsregeln verwendet.

Als Alternative wurde zu Beginn des Projektes überlegt, je Konfigurationsraum einen spezifischen Editor zu generieren. Dadurch würde aber die Komplexität beim Ausliefern der Editoren erheblich steigen und ein generischer Editoranteil wäre trotzdem vorhanden. Somit war auch die Forderung nach Anpassbarkeit der Editoren erfüllt.

Auf dem selben Wege konnte auch der Wunsch, dass Endkunden selbst Änderungen an Teilmodellen vornehmen können, erfüllt werden. Somit kann der Endkunde selbst die gelieferte Software der IBYKUS noch stärker auf seine eigenen Bedürfnisse anpassen. Für diesen Zweck wird entsprechend ein abgespeckter Konfigurationsraum geschaffen, der die erlaubten Modellierungsmöglichkeiten und auch die Funktionen des Editors einschränkt. Zusätzlich ist in der Lieferung von IBYKUS ein abgespecktes DVR enthalten, dass die Modelle des Endkunden direkt interpretiert und die ausgeführte Software live modifiziert.

Das Arbeiten mit Modellen geschieht für den Modellierer nun in mehreren Teilschritten, die zum Großteil transparent ausgeführt werden. Er meldet sich am Repository mit seinen Nutzerdaten und seiner aktuellen Aufgabe an und führt lediglich die Funktion laden oder auschecken des Modells im Modellnavigator durch. Intern passiert allerdings folgendes:

- Suchen des Teilmodells und Laden des XML-Datenstroms
- Lokales Speichern des XML-Datenstroms
- Transformieren der XML-Daten mit Hilfe von Xpand, siehe [XPAN], in das textuelle Editorformat
- Auswerten der Modellmetainformation zum Finden des korrekten Editors
- Validieren des Modells
- Öffnen des Editorfensters und Anzeige des textuellen Modells inkl. Validierungsergebnissen

Die Anforderung, unstrukturierte Daten innerhalb der Modelle zu bearbeiten, wurde durch einen Ansatz verfolgt, der den CDATA Elementen in XML, siehe [CDAT], ähnelt. An den Stellen im Modell, an denen ein entsprechender Binärdatenblock enthalten ist, wird beim Laden des Modells aus dem DVR im Transformationsschritt dieser Block ausgeschnitten, in eine lokale Datei mit korrekter Dateiendung gespeichert und ein eindeutiger Verweis auf diese Datei zurück in das Modell geschrieben.

Navigiert der Modellierer nun auf diese Verweise, wird automatisch ein dazu passender Editor, beispielweise Microsoft Word oder OpenOffice Writer, geöffnet. Beim Zurückschreiben in das DVR geschieht der Prozess genau umgekehrt. Der Verweis auf die Datei wird ersetzt durch den Inhalt der externen Datei und entsprechend in einer CData-Sektion in das DVR transportiert. Somit unterliegen diese unstrukturierten Daten auch automatisch der Versionierung, allerdings mit dem Nachteil, dass keine Unterschiede zu Vorgängerversionen automatisch erstellt werden können.

Auch die weiteren Anforderungen wie Modelrefactoring, Autoformatierung und Mischen mit nutzerspezifischen Formatierungen, Makros in Modellen, Aspekte im Modellnavigator, verschiedene synchrone und asynchrone Möglichkeiten der Modellvalidierung und auch das Navigieren von referenzierten Modellelementen im lokalen Modell und im DVR wurden auf dieser technischen Basis gelöst.

Ausführlichere Darstellungen dazu und auch Screenshots werden aus Platzgründen allerdings nur im elektronischen Tagungsband erscheinen.

5 Ausbaustufen

Da am Anfang der Entwicklung der APx-Plattform noch nicht alle technischen Details bekannt sein konnten, sondern nur grobe Vorstellungen über die Funktionalität bestanden, wurde eine agile Vorgehensweise mit mehreren Ausbaustufen entsprechend dem Konzept Agiler Festpreis gewählt. Die einzelnen Ausbaustufen hatten Festpreise und entsprechende Prognosen der Preise für die nächsten Stufen. Diese Prognosen wurden iterativ verfeinert. Nur so war es möglich, den ungefähren Investitionsbedarf abzuschätzen, trotzdem aber genügend Freiraum für den *Appetit, der beim Essen kommt,* zu haben.

Die einzelnen Ausbaustufen waren:

textueller Editor mit grundlegenden Funktionen zur Schaffung der grundlegenden Akzeptanz und zur konkreteren Abschätzung des gewünschten Bedienverhaltens. Auf dieser Basis entstand der Editorprototyp, der die verschiedenen Grammatikvarianten und die grundlegenden Bedienkonzepte zeigte.

DVR-Anbindung mit entfernter Ablage der Modelle im Repository. Zusätzlich wurden hier die notwendigen Transformationsschritte beim Laden und Speichern und auch das Generieren der spezifischen Editorfunktionen aus der Konfigurationsraumbeschreibung ergänzt. In dieser Stufe wurde auch die Basis für den Modellnavigator realisiert.

Modellnavigator, genauer gesagt Repositorynavigator, da nicht nur Teile des lokalen Modells sondern auch die virtuellen Strukturen des Gesamtmodells angezeigt wurden. In diesem Navigator wurden die notwendigen Nutzeraktionen weiter verfeinert und ergänzt.

Erweiterte Funktionen wurden in der aktuell letzten Ausbaustufe der textuellen Variante ergänzt. Dies waren vor allem die oben beschriebene Integration der unstrukturierten Daten, die Formatierungsfunktionen, das Refactoring, die Komponentisierung für die Endkundenauslieferung und die Internationalisierung.

Herausfordernd bei dieser Vorgehensweise war das permanente Refactoring der einzelnen Plugins, da ja bspw. die Schnittstelle zum DVR permanent verfeinert wurde. Dies war immer eine Änderung an einer Basisfunktionalität, die sich durch mehrere Komponenten zog. Interessant war auch das bereits genannte Symptom *Der Appetit kommt beim Essen*, was nicht selten dazu führte, dass selbst die grundlegende generische Grammatik des Editors im Projekt erweitert wurde, genauso wie die immer bessere Nutzerführung. Der Editor wurde ab Ausbaustufe 2 einer größeren Testergruppe zur Verfügung gestellt, die wiederum weitere Verbesserungsvorschläge einbrachte.

6 Zusammenfassung und Ausblick

Die recht komplexen Anforderungen an Werkzeuge zur Realisierung großer Softwareproduktfamilien konnten durch die konsequente Nutzung modellbasierter Verfahren in konkreten Tools umgesetzt werden. Bereits in einer frühen Ausbaustufe wurde die neue Umgebung der altbewährten vorgezogen.

Perspektivisch werden noch grafische Editoren ergänzt, um bspw. Formularlayouts grafisch zu modellieren oder sogenannte Fachklassendiagramme zusammen mit Fachexperten zu erstellen.

7 Referenzen

[CDAT] http://www.w3.org/TR/REC-xml/#sec-cdata-sect
[ECK] http://eclipse.org
[EMF] http://www.eclipse.org/modeling/emf/
[MWE] http://wiki.eclipse.org/Modeling_Workflow_Engine_%28MWE%29
[XTEX] http://www.eclipse.org/Xtext/