Werkzeuge zur Erstellung und Verarbeitung von DSLs

Version 0.2
Markus Völter, itemis
voelter@acm.org

Abstract

Domänenspezifische Sprachen werden als Teil der modernen Softwareentwicklung immer wichtiger, die dazugehörige Toollandschaft entwickelt sich spürbar weiter. Die neuesten Kandidaten in diesem Umfeld sind Microsofts Oslo sowie MPS von Jetbrains. In diesem Artikel möchte ich auf den aktuellen State-of-the-Art eingehen und erläutern, welche Tools derzeit verfügbar sind, wie sie sich voneinander unterscheiden und wie reif sie für den Einsatz in der Praxis sind. Dem Leser soll damit der Überblick über das weite und unübersichtliche Feld erleichtert und ggfs. eine Entscheidung für oder gegen das eine oder andere Tool ermöglicht werden. In dem Artikel betrachte ich Eclipse Modeling und openArchitectureWare, Microsoft DSL Tools, MetaEdit+, Jetbrains' MPS sowie Microsoft Oslo.

Einleitung

Um mit domänenspezifischen Sprachen effizient arbeiten zu können, braucht man Werkzeuge. Zunächst sind das Werkzeuge zur Erstellung der domänenspezifischen Sprachen an sich und der dazugehörenden Editoren. Des Weiteren werden Plattformen und Tools benötigt, um Interpreter oder Codegeneratoren für die Sprachen zu erstellen. Je nach Werkzeug kommen dazu noch Laufzeitumgebungen für die Editoren sowie mehr oder weniger skalierbare Repositories zum Speichern von großen oder vielen Modellen.

Gerade was das Editieren von Programmen und Modellen angeht (ich verwende hier Programm und Modell synonym), sind wir durch moderne IDEs (für textuelle Sprachen) und UML-Werkzeuge (für grafische Sprachen) ziemlich verwöhnt, und wir wollen diesen Komfort natürlich auch für domänenspezifische Sprachen nutzen. Und natürlich will auch derjenige, der die Sprachen und Editoren baut, selbst diesen Komfort für seine Arbeit in Anspruch nehmen. Man braucht also gewissermaßen Sprach-IDEs. Martin Fowler hatte diese Art von System vor ein paar Jahren als Language Workbench bezeichnet [ref].

In diesem Artikel möchte ich auf einige der weit verbreiteten Language Workbenches eingehen und die Stärken und Schwächen dieser Werkzeuge aufzeigen. Die Auswahl der Werkzeuge wurde auch dadurch beeinflusst,

inwiefern man an dem betreffenden Werkzeug allgemein interessante Dinge zeigen kann. Einige der Werkzeuge sind altbewährt und für den rauen Alltagsgebrauch einsetzbar, andere sind noch ganz neue oder gerade erst im Entstehen.

Zunächst aber ein Überblick über die Bestandteile einer Language Workbench.

Bestandteile von DSLs

Wie bei Programmiersprachen auch muss man sich bei der Definition einer DSL mindestens um die folgenden drei Aspekte kümmern.

Für das verarbeitende Werkzeug (beispielsweise Codefgenerator) ist die abstrakte Syntax relevant (auch als Metamodell bezeichnet). Die abstrakte Syntax ist eine Datenstruktur, die den semantisch relevanten Inhalt eines mit der DSL beschriebenen Programms repräsentiert. Interpreter oder Codegeneratoren traversieren diese Datenstruktur.

Die konkrete Syntax ist die grafische oder textuelle Darstellung der abstrakten Syntax. Sie ist quasi das User Interface der Sprache. Anwender, die die Sprache nutzen, interagieren mit der konkreten Syntax. Es ist daher essenziell, dass diese für die entsprechende Zielgruppe angepasst ist. Oft werden Sprachen mit grafischer Syntax als Modellierungssprachen bezeichnet.

Der dritte Aspekt einer Sprache ist deren Semantik, also die Bedeutung dessen was man mittels der konkreten Syntax hin schreibt, und mittels der abstrakten Syntax dem Tool zur Verarbeitung zur Verfügung stellt. Es gibt verschiedene Arten der formalen Semantikdefinition, in der Praxis werden diese allerdings nicht verwendet. Semantik wird entweder durch eine Dokumentation der Sprache in Prosa definiert, oder mittels Interpretern und Codegeneratoren. Wenn ein Codegenerator eine DSL auf eine bekannte Sprache mit bekannter Semantik abbildet (beispielsweise C#), so lässt sich daraus im Umkehrschluss die Semantik der DSL ableiten.

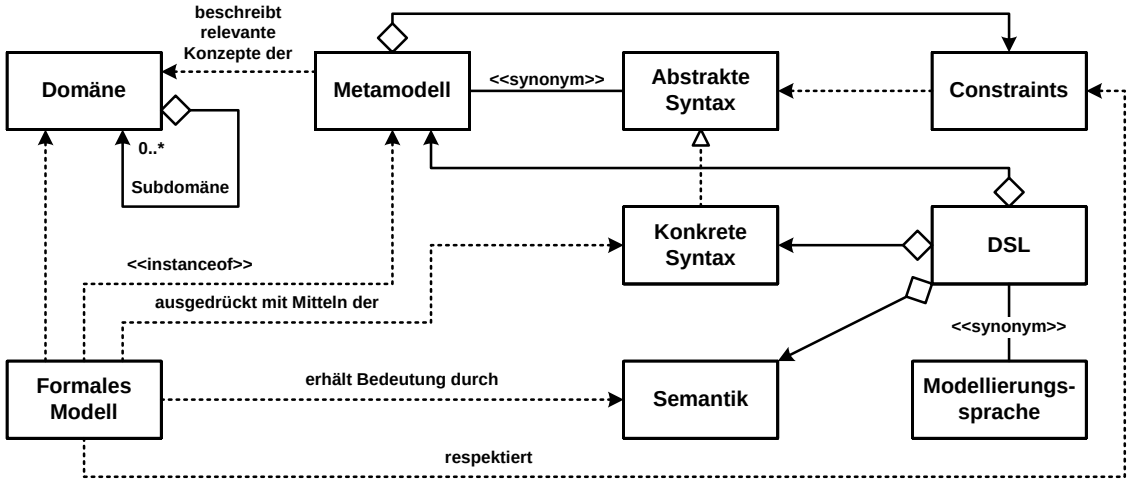

Abbildung 1: Zusammenhang zwischen Metamodell, konkreter Syntax, Semantik, Constraints, Modell und Domäne

In der Praxis werden noch zwei weitere Aspekte benötigt. Zum einen so genannte Constraints, und zum anderen natürlich die Modellverarbeiter, also Codegeneratoren oder Interpreter.

In aller Regel gibt es Korrektheitsregeln für Programme einer DSL, die sich mit gängigen Formalismen zur Beschreibung von abstrakter Syntax definieren lassen. Beispielsweise müssen die Attribute einer Datenstruktur eindeutige Namen haben. Gleiches gilt sinngemäß auch für die Zustände in einer Zustandsmaschine. Constraints dienen dazu, genau solche Dinge sicherzustellen. Constraints sind Boolsche Ausdrücke. Damit ein Modell als korrekt betrachtet wird, müssen alle Constrains zu true evaluieren.

Im hier betrachteten Zusammenhang dienen DSLs der Entwicklung von Software. In aller Regel bedeutet dies, dass die per DSL erstellten Programme/Modelle in existierende Laufzeitinfrastrukturen eingefügt werden müssen. Dafür gibt es prinzipiell zwei Vorgehensweisen (es existieren auch Mischformen).

Ein Interpreter ist ein Programm, welches den abstrakten Syntaxbaum eines Modells traversiert und der Semantik der Sprache entsprechende Seiteneffekte ausführt. Ein einfacher Interpreter für einen Taschenrechner würde für eine Datenstruktur, die die Addition zweier Zahlen repräsentiert, tatsächlich diese beiden Zahlen addieren.

Codegeneratoren sind spezielle Interpreter, die als Seiteneffekt der Modeltraverierung semantisch äquivalenten Quellcode in einer existierenden Programmiersprache ausgeben. Dieser wird dann nachfolgend interpretiert oder seinerseits kompiliert.

In der Praxis werden auch oft Modelltransformatoren benötigt, die entweder Modelle ergänzen (quasi als Prä- oder Postprozessor) oder Instanzen eines Metamodells in Instanzen eines anderen Metamodells überführen. Dies ist beispielsweise notwendig, wenn man Modelle abstrakterer DSLs (die beispielsweise Fachlichkeit beschreiben) auf Modelle von DSLs abbildet, die eher technischer Natur sind (beispielsweise Architektur-DSLs).

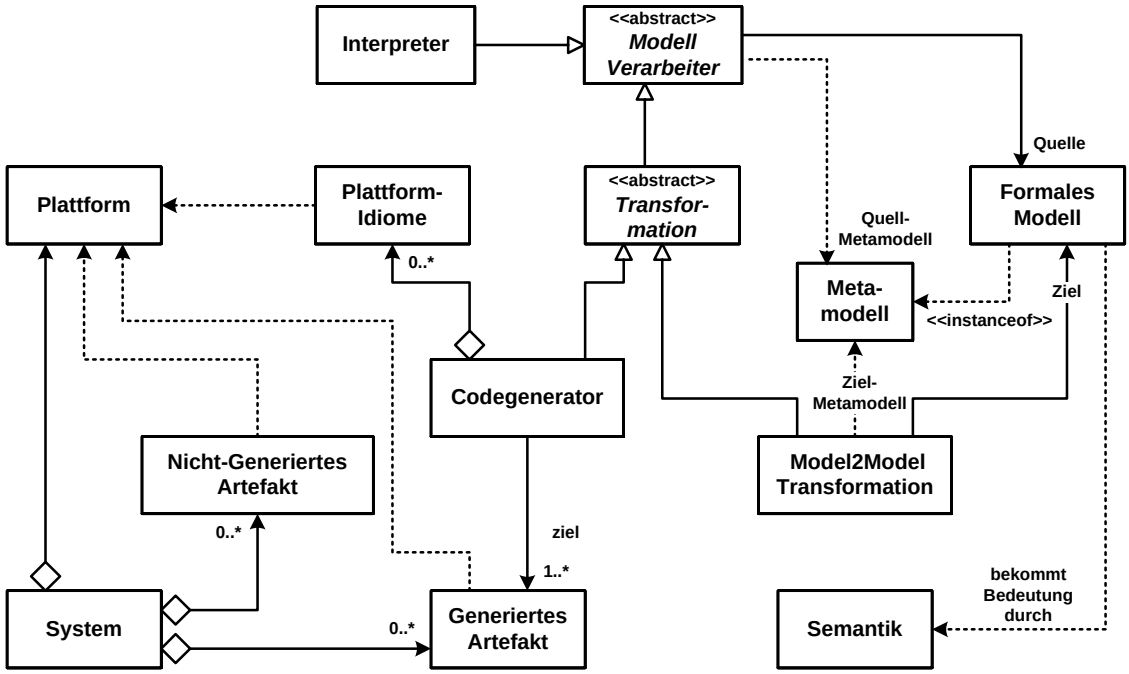

Abbildung 2: Zusammenhang zwischen dem Modell und den verarbeitenden Werkzeugen

Ein weiteres wichtiges Tool, das wir noch nicht erwähnt haben, ist der Editor. Dessen Aufgabe ist es, den Umgang mit der konkreten Syntax der DSL für den Anwender möglichst effizient zu gestalten. In aller Regel baut der Editor (mehr oder weniger direkt) einen der abstrakten Syntax entsprechenden Baum auf. Bei Sprachen mit grafische Syntax geschieht dies in aller Regel direkt: wenn man beispielsweise einen Zustand in ein Zustandsdiagramm einfügt, so wird im entsprechenden Syntaxbaum auch direkt das entsprechende Modellelement angelegt. Bei Sprachen mit textueller Syntax ist diese Abbildung in aller Regel nicht so einfach, sondern benötigt einen Parser. Die Aufgabe des Parsers ist, die textuelle konkrete Syntax eines Programms auf Korrektheit zu prüfen und den zur konkreten Syntax passenden abstrakten Syntaxbaum aufzubauen. Dazu kommt eine so genannte Grammatik zum Einsatz. Dies ist letztendlich eine formale Beschreibung, die Abbildungsregeln von der konkreten Syntax auf die abstrakte Syntax definiert.

Eclipse Modeling und openArchitectureWare

Im Rahmen des Eclipse-Projektes werden auch eine ganze Reihe Werkzeuge entwickelt, die mit Modellierung und modellgetriebener Entwicklung zu tun haben. Diese sind zusammengefasst im Eclipse Modeling Projekt, und seinen Unterprojekten und -komponenten. Bei Eclipse Modeling handelt es sich also um kein Produkt, sondern um eine Sammlung von Frameworks und Tools. Kern des Ganzen stellt das Eclipse Modeling Framework (EMF) dar. Es dient zur Beschreibung der abstrakten Syntax von Domänenspezifischen Sprachen (dem Metamodell). EMF enthält auch eine ganze Reihe von APIs und Bibliotheken zum Umgang mit EMF-Modellen, sowie eine XML-basierte Persistenzschicht. Mit CDO gibt es bei Eclipse auch eine Anbindung an relationale Datenbanken.

Neben EMF ist insbesondere GMF bekannt, das Graphical Modeling Framework. Mit diesem lassen sich grafische Editoren erstellen (Box-and-Line Diagramme). Als Gegenstück für textuelle Domänenspezifische Sprachen befindet sich derzeit TMF in der Entwicklung, das Textual Modeling Framework. Zur Validierung von Modellen existieren verschiedene Implementierungen von OCL. Zur Überführung von Modellen in andere Modelle gibt es verschiedene, teils auf OMG-Standards beruhende Modelltransformationssprachen, darunter QVT Operational, ATL und Xtend. Zur Codegenerierung kommt insbesondere Xpand zum Einsatz. Allerdings existieren auch hier verschiedene Alternativen.

Weil Eclipse Modeling kein fertiges Produkt darstellt, gibt es verschiedenste Distributionen und Zusammenstellungen, die dem Anwender den Umgang mit der großen Menge an Tools und Frameworks erleichtern. Neben kommerziellen Angeboten (beispielsweise Borland's Together) ist insbesondere das Open Source-Werkzeug openArchitectureWare weiterverbreitet.

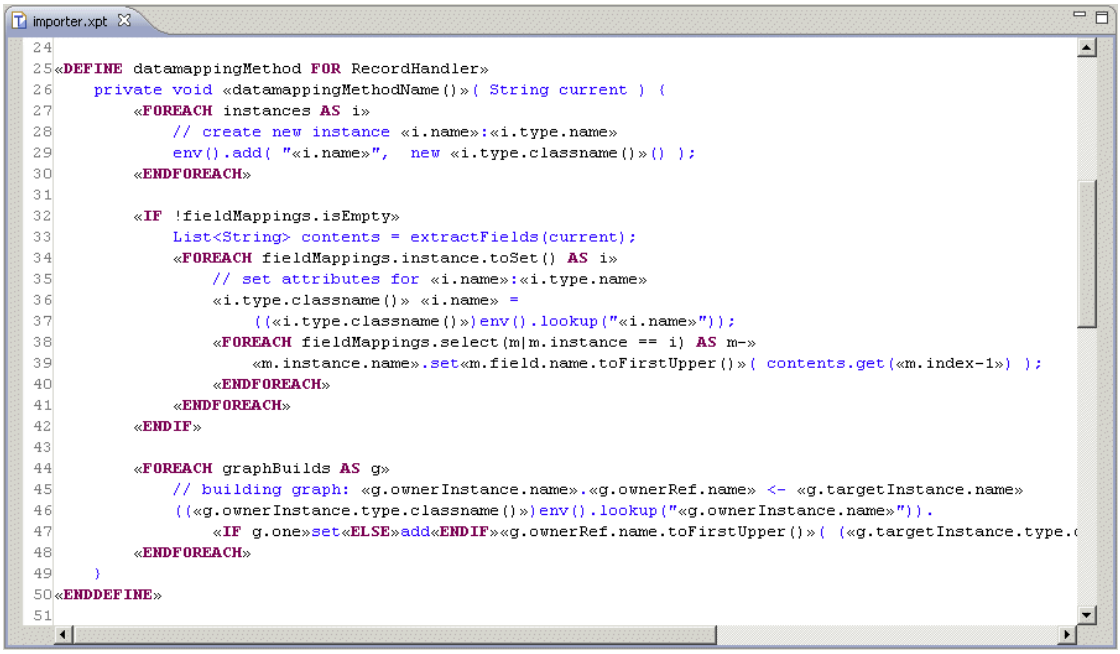

Abbildung 3: Codegenerierungstemplate von Eclipse oAW Xpand.

Eclipse Modeling im allgemeinen und openArchitectureWare im besonderen stellen eine bewährte und praxistaugliche Plattform zu Modellgetriebene Entwicklung dar. Der größte Vorteil ist die Flexibilität und Offenheit der Werkzeuge und Plattform. Viele Aspekte lassen sich - entsprechende Programmierkenntnisse vorausgesetzt - an die eigenen Anforderungen anpassen. Dies ist nicht zuletzt auch deshalb der Fall, weil alle Werkzeuge und Frameworks unter der Eclipse Public Licence als Open Source verfügbar sind, und in einigen Fällen OMG Standards implementieren.

Die Erstellung textueller domänenspezifischer Sprachen und ihrer Editoren wird von TMF (und seinem Vorläufer openArchitectureWare Xtext) sehr gut unterstützt. Aus einer EBNF-artigen Grammatik wird nicht nur einen Parser und ein Metamodell erstellt, sondern auch automatisch einen Eclipse-Editor

abgeleitet, der für die DSL Syntax-Highlighting, Code Completion, Go-To-Definition und andere aus modernen IDEs bekannten Features bietet.

Die Erstellung grafischer Editoren ist zwar mit GMF sehr flexibel möglich, bedarf aber einiges an Einarbeitung und Handarbeit, wenn man wirklich benutzbare und skalierbarer Editoren erstellen will. Im Gegensatz zu MetaEdit+ (siehe nächster Abschnitt) ist einiges an detaillierter Konfiguration und teils auch Programmierung notwendig.

MetaEdit+

MetaEdit+ wird schon seit 1995 von der finnischen Firma Metacase entwickelt und vertrieben, es ist ein kommerzielles Produkt. Der Fokus bei MetaEdit+ liegt auf grafischen DSLs und Codegenerierung. Zur Beschreibung von graphischer konkreter Syntax bringt MetaEdit+ einen schönen Editor mit. Mit diesem kann man in WYSIWYG Manier seine Symbole erstellen und sie mit den Abstraktionen des Metamodells verbinden. Der damit erstellter Editor lässt sich dann direkt in der entsprechenden Laufzeitumgebung zur Modellierung nutzen. Es ist keine Codegenerierung oder ein separater Deploymentschritt erforderlich, die Editoren sind vollkommen interpretativ implementiert. Dies hat den Vorteil eines schnellen Turnarounds, hat aber auch den Nachteil, dass alle Anwender der DSL die MetaEdit+ Laufzeitumgebung benötigen. Die Beschreibung von Codegeneratoren geschieht mittels einer Templatesprache. Dafür ist das Tooling zur Erstellung grafischer DSLs mit Abstand das beste auf dem Markt.

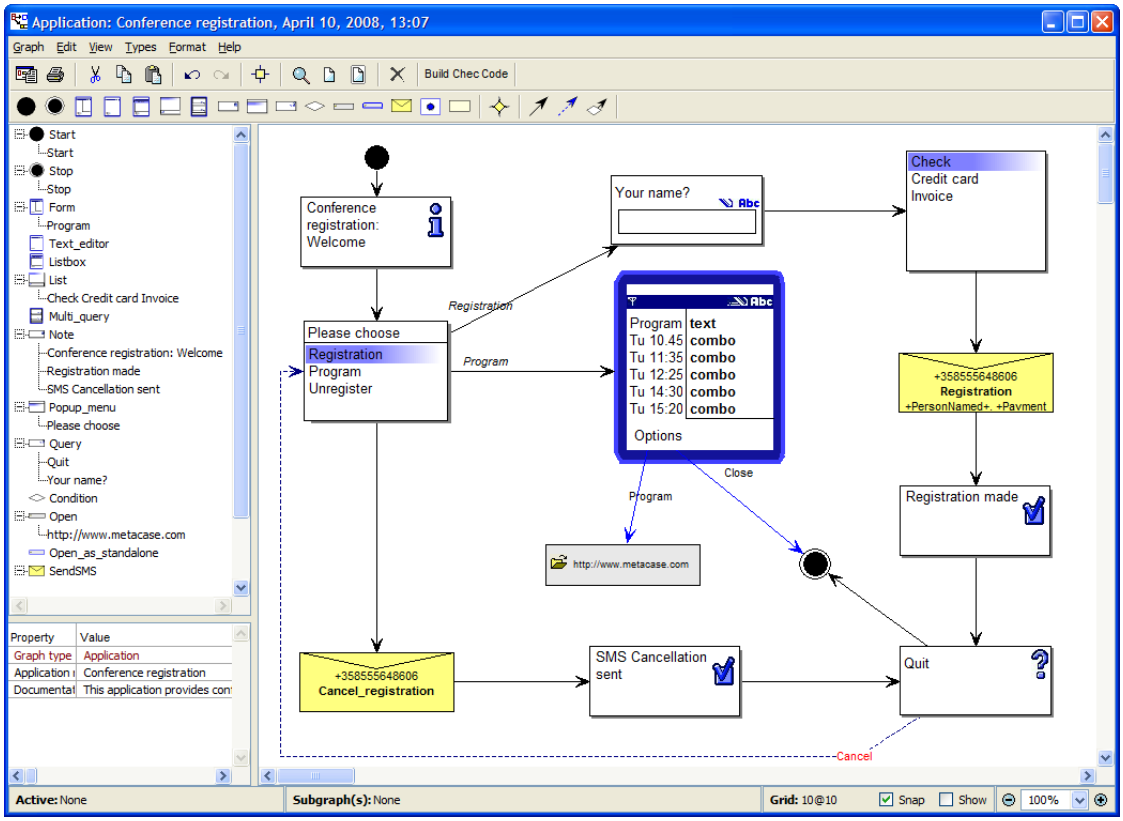

Abbildung 4: Ein mit MetaEdit+ erstellter Diagrammeditor

MetaEdit+ kommt auch mit einem Repository, so dass arbeiten im Team an einem gemeinsamen, im Repository abgelegten Modell problemlos möglich ist. Man merkt bei Metacase, dass sie schon lange in diesem Geschäft unterwegs sind. Aus der langjährigen Erfahrung ist Ihnen bewusst, dass die Migration von Modellen bei Änderungen der zu Grunde liegenden Sprache ein nicht triviales aber sehr wichtiges Problem darstellt. MetaEdit+ bringt daher entsprechendes Tooling mit, um solche Modelmigrationen möglichst schmerzfrei umzusetzen. Die Tatsache, dass die Editoren die Sprachdefinition interpretieren (und nicht wie bei Eclipse GMF auf aus der Sprache generiertem Code beruhen) macht den toleranten Umgang mit alten Versionen von Modellen leichter.

MetaEdit+ liefert übrigens keinen direkten Support für Modell-zu-Modell Transformationen. Man muss dies dadurch emulieren, dass man mittels des Codegenerators die serialisierte Form des Zielmodells erstellt.

JetBrains MPS

JetBrains, bekannt von Werkzeugen wie IntelliJ IDEA oder Resharper, hat Ende 2008 die öffentliche Betaphase ihres Werkzeuges MPS eingeläutet. MPS steht für Metaprogramming System und ist ein Werkzeug zum Bau und Verwendung von (derzeit) textuellen DSLs. Im Gegensatz zur klassischen Implementierung von textuellen Sprachen mittels Parsern kommt MPS vollständig ohne Parser aus. Als Folge davon lassen sich sehr leicht Sprachen kombinieren; dazu gleich mehr. MPS verwendet so genannte strukturelle Editoren, in denen die textuelle Syntax eine Projektion des abstrakten Syntaxbaums darstellt. Der Benutzer editiert direkt den Baum, wobei sich MPS redlich Mühe gibt, dass es sich so anfühlt, als würde man Text editieren. Bei der Definition einer neuen Sprache definiert man zunächst das Metamodell, und dann, in einer Art Formularsprache, die Abbildung der Metamodellbestandteile auf eine textuelle Syntax.

Standardmäßig kommt MPS mit einer Implementierung von Java, genannt BaseLanguage. Diese erfüllt zwei Aufgaben. Zum einen werden domänenspezifische Sprachen oft auf BaseLanguage abgebildet, so dass die Programme dann mit dem mitgelieferten Compiler in normalen Java Byte Code überführt werden können (natürlich kann man auch XML oder anderen beliebigen Text generieren). Zum anderen lässt sich aber BaseLanguage auch erweitern! Beispielsweise lassen sich mit sehr wenig Aufwand Zustandsmaschinen in Javaklassen einbauen, neue Statements einführen, oder Interfaces mit Vor- und Nachbedingungen erweitern. Die resultierende Sprache fühlt sich im Editor ganz genau so an wie das mitgelieferte Java. Dies liegt daran, dass eine Erweiterung der Sprache auch immer automatisch eine Erweiterung des Editors mit sich führt. Konsequenterweise werden in MPS auch keine Codegenerierungstemplates verwendet. Die Überführung von DSL Code in beispielsweise Java Code ist tatsächlich eine Modelltransformation, da ja auch Java als “Modell“ abgelegt wird. Das Schöne ist allerdings, dass solche Modelltransformationen die konkrete Syntax der Zielsprache verwenden, weswegen sie aussehen wie Codegenerierungstemplates.

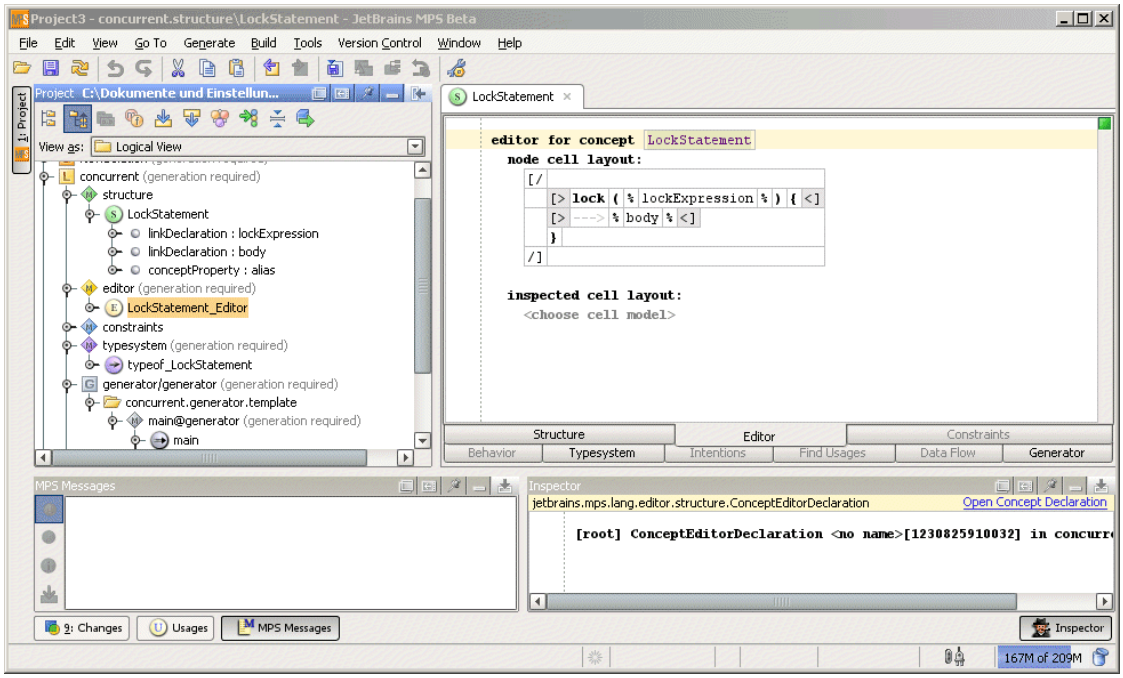

Abbildung 5: Definition der Syntax für ein *lock(...) {...}* Statement in Java

Mit MPS eröffnen sich für den DSL-Implementierer völlig neue Möglichkeiten, da er nicht nur separate, externe DSLs entwickeln kann, sondern existierende Sprachen (im Falle von MPS derzeit leider nur Java) transparent erweitern kann. Projekte können beispielsweise mit "normalem" Java beginnen, und wenn sich domänenspezifische Abstraktionen herauskristallisieren, kann man diese in Sprache und Tools integrieren. Der Haken ist natürlich, dass man zur Verwendung dieser Sprachen MPS einsetzen muss, ein normaler Texteditor reicht nicht aus, da eben kein Text editiert wird. Außerdem merkt man durchaus, dass man direkt auf einer textuell aussehenden Baumstruktur arbeitet. Meiner Erfahrung nach kann man sich daran aber im Laufe von ein paar Tagen gewöhnen.

MPS ist derzeit in Beta und ist unter der Apache 2.0 Lizenz verfügbar. Spätestens im zweiten Quartal des Jahres sollte eine Version 1.0 verfügbar sein.

Microsoft DSL Tools

Die DSL Tools sind Microsofts erster Versuch im Bereich der domänenspezifischen Sprachen (Oslo, siehe unten, der zweite). Von der Struktur her sind die DSL Tools stark an Eclipse Modeling angelehnt. Mit dem MDF, dem Meta Data Framework lässt sich die abstrakte Syntax (das Metamodell) von domänenspezifischen Sprachen definieren. MDF ist von seiner Ausdrucksmächtigkeit her mit EMF vergleichbar. Im Folgenden kann man dann für seine DSL eine grafische Box-and-Line Syntax definieren. Die Werkzeuge sind hier etwas einfacher zu benutzen als die von GMF. Die erstellten Editoren lassen sich in Visual Studio deployen. Zur Entwicklung von Codegeneratoren existieren verschiedene Templateengines: T3 und T4. Diese sind von ihrer Mächtigkeit der allerdings nicht mit Eclipse Xpand vergleichbar.

Der Vorteil der DSL Tools liegt insbesondere in ihrer Integration mit den Microsoft Entwicklungstools sowie ihrer Konsistenz. Im Gegensatz zu Eclipse Modeling gibt es in der Microsoft Welt keinen Zoo von mehr oder weniger zusammenpassenden Komponenten aus denen der Benutzer sich ein passendes Subset auswählen muss. Stattdessen sind die Bestandteile der DSL Tools aufeinander abgestimmt und funktionieren out-of-the-box. Natürlich fehlt auf der anderen Seite die Flexibilität, die man in der Eclipse vorfindet.

Microsoft Oslo

Oslo ist Microsofts zweiter Versuch im Bereich der domänenspezifischen Sprachen. Es wurde im Oktober 2008 auf der PDC vorgestellt. Im Gegensatz zu den DSL Tools habe ich den Eindruck, dass Oslo ein viel Microsoft strategisches Vorhaben ist (bei den DSL Tools hatte ich immer den Eindruck dass es außer ein paar Entwicklern in Cambridge bei Microsoft keinen so recht interessiert). Microsoft Oslo besteht aus verschiedenen Komponenten und Sprachen. Quadrant ist das primäre Werkzeug zum Umgang mit Modellen, textuell oder grafisch. Zur Speicherung von Modellen kommt ein auf dem MS SQL Server basierendes Repository zum Einsatz. Zur Definition von abstraktereren konkreter Syntax sowie zum Zugriff auf Modelle aus Quellcode kommt die M Sprachfamilie zur Verwendung. M Besteht aus drei Teilsprachen:

- MSchema dient zur Definition von Datenstrukturen, ganz ähnlich wie XML Schema, verwendet aber eine angenehmere Syntax. Anknüpfend an obige Erläuterungen dient MSchema also zur Definition der abstrakten Syntax von Sprachen.
- MGrammar ist eine Sprache zur Grammatikdefinition. Diese dient also letztendlich zur Festlegung textueller konkreter Syntax und deren Abbildung auf MSchema.
- MGraph dient letztendlich zur Beschreibung der tatsächlichen Nutzdaten von Modellen, und entspricht damit mehr oder weniger XML.

Ein kleiner Hinweis am Rande: die Unterscheidung in diese drei Teilesprachen wird vermutlich nicht so klar bleiben wie es hier scheint. In einem Interview mit Software Engineering Radio hat Doug Purdy erläutert, dass diese drei sprachen gerade mehr oder weniger in einen Formalismus überführt werden.

Zusätzlich zu Quadrant als primären Werkzeug zum Umgang mit Modellen existiert zur Definition und zum Testen von abstrakter Syntax und Grammatiken außerdem ein Werkzeug namens Intellipad. Mit diesem lassen sich sehr schön interaktiv Grammatiken definieren. Dies wird mittelfristig vermutlich in Quadrant integriert. Diverse Compiler und Kommandozeilen Tools runden das Paket ab.

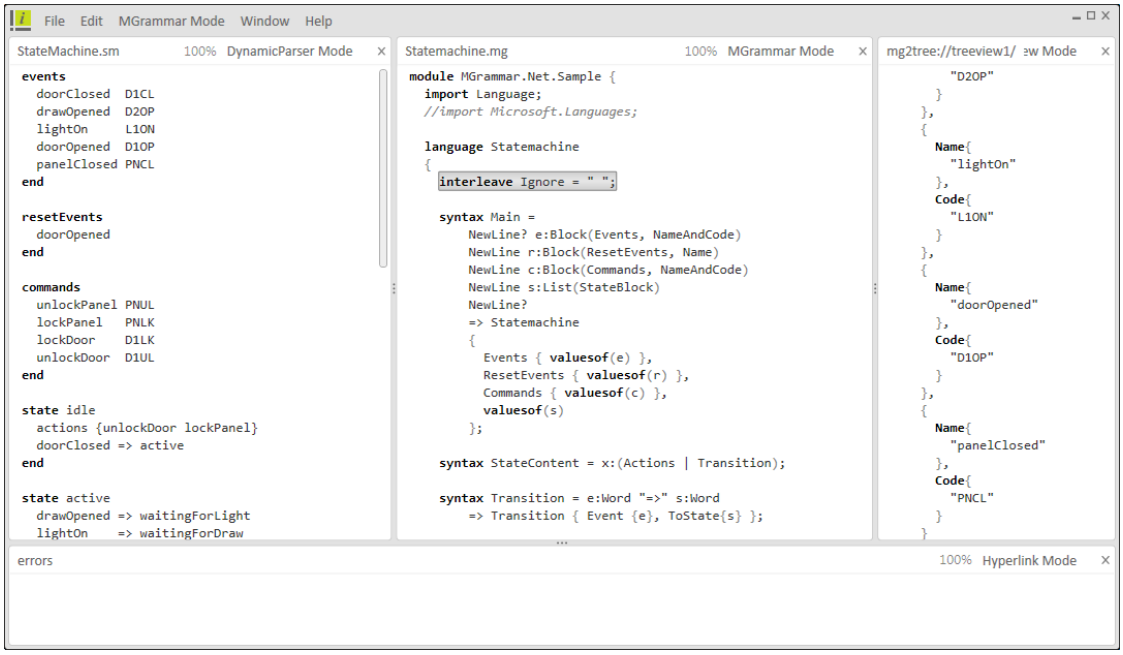

Abbildung 6: Grammarikdefinition MGrammar in Intellipad

In obiger Diskussion der Tools fällt vermutlich auf, das nicht von Werkzeugen zum Bau von Codegeneratoren die Rede ist. Das liegt daran, dass Microsoft derzeit vor allem Interpreter im Hinterkopf hat. Dafür gibt es aus unserer Sicht zwei Gründe:

- zum einen hat Microsoft in der Vergangenheit gute Erfahrungen gemacht mit Plattformen, die XML Konfigurationsdateien interpretiert haben. WCF und WF sind Beispiele dafür. Das Problem bei all diesen Plattformen war eben vor allem die schlechte Syntax von XML, und nicht der interpretative Ansatz. Microsoft will in erster Linie diesen Ansatz auf eine solide Basis stellen, indem anpaßbares Syntax und ein vernünftiges Repository zur Verfügung gestellt werden.
- Zum anderen existieren ja immer noch die DSL Tools. Diese sind für grafische DSLs optimiert, und kommen mit Werkzeugen zur Codegenerierung. Es zwängt sich der Eindruck auf, dass sich Microsoft nicht selbst Konkurrenz machen will. In oben erwähntem SE Radio Interview sagte Doug auch, dass es mittelfristig eine Integration der beiden Welten geben wird, so dass man beispielsweise für mit MSchema definierte abstrakte Syntax graphischer Editoren definieren kann, oder mittels der Generator Werkzeuge der DSL Tools aus MGraph Code generieren kann.

Unsere persönliche Interpretation dieser Aussage ist, dass die DSL Tools mittelfristig in Oslo aufgehen werden.

Der aktuelle Stand von Microsoft Oslo ist der eines Technology Preview. Es gibt kein Release Datum, kein wohldefiniertes Produkt, und wir sind auch noch nicht in der Betaphase. Im Laufe der nächsten Monate oder Jahre will das Oslo Team in regelmäßigen und recht kurzen Abständen Drops der Werkzeugkette

veröffentlichen, insbesondere in Abstimmung mit Konferenzen oder anderen relevanten Events.

Zusammenfassung

Anhand der obigen Schilderungen wird klar, dass alle diese Werkzeuge ihre Vor- und Nachteile haben. Und das gilt ganz abgesehen davon, dass einige noch nicht Produktreife erreicht haben. Die Eclipse Werkzeuge eignen sich sehr gut zum Bau umfassender, offener, domänenspezifischer Plattformen. Untertützt werden alle Facetten der modellgetriebenen Entwicklung. MetaEdit+ ist klarer Klassensieger bei grafischen Editoren, lässt sich aber aufgrund seiner proprietären Architektur nicht ohne weiteres in größere Workbenches einbauen. MPS hat seine Stärke ganz klar bei der Erweiterung existierender Sprachen, derzeit leider nur Java. Die DSL Tools und vor allem Microsoft Oslo spielen ihr volles Potenzial vor allem dann aus, wenn die Laufzeitumgebungen auf Microsoft Plattformen laufen.

Allerdings wollte ich mit diesem Artikel nicht nur die einzelnen Werkzeuge vorstellen und voneinander abgrenzen. Mir ist wichtig, klarzumachen, dass Domänenspezifische Sprachen und die damit zusammenhängenden Themen und Tools der Kinderstube entwachsen sind. Aufgrund der Vielfältigkeit der Werkzeuge, und der Tatsache, dass Mainstream-Plattformen wie Eclipse und Microsoft das Thema aufgreifen, wird klar, dass DSLs in der Praxis einsetzbar sind und ihre versprochenen Vorteile auch ausspielen können.

Kasten: Was sind Domänenspezifische Sprachen

Statt zu versuchen, eine geschlossene Definition des Begriffs "Domänenspezifische Sprache" zu liefern, möchte wir DSLs einfach mit einer Reihe von anderen Technologien vergleichen, und die Unterschiede aufzeigen. Dadurch sollte klar werden, was eine DSL ist.

DSL vs. Programmiersprache. Mit einer Programmiersprache lassen sich beliebige Sachverhalte beschreiben und Probleme lösen, sofern es dafür Algorithmen gibt. Programmiersprachen sind daher sehr mächtig und allgemein anwendbar. Domänenspezifische sprachen sind für die Beschreibung von Sachverhalten und Algorithmik einer bestimmten Fachdomäne optimiert. Daher sind DSLs in aller Regel erheblich einfacher und kleiner. Sehr oft sind sie

auch nicht Turing vollständig, sondern dienen dazu, Domänenspezifische Konzepte präzise und Tool-verarbeitbar zu beschreiben. Durch Codegeneratoren oder Interpreter werden solche Modelle dann ausführbar.

DSL vs. XML. Man könnte argumentieren, dass sich mit XML prinzipiell auch beliebige Domänenspezifische Strukturen und Verhaltensweisen beschreiben lassen. In gewisser Weise handelt es sich bei einem XML Schema um eine Sprachdefinition. Der Haken bei der Sache ist allerdings, dass man die Notation (also die konkrete Syntax) nicht anpassen kann. Alles sind geschachtelte und sich gegenseitig referenzieren spitze Klammern. Dies hat zur Folge, dass man mit XML nicht wirklich für den Menschen gedachte Modelle und Programme beschreiben kann (auch wenn dies ursprünglich mal die Idee von XML war). DSLs unterscheiden sich also vor allem dadurch von XML, dass sich eine beliebige (grafische, textuelle oder anderweitige) Notation definieren lässt.

DSL vs UML. Auch UML ist eine Modellierungssprache, und seit Version zwei auch so formal definiert, dass man da raus prinzipiell ausführbare Systeme generieren kann. Allerdings ist die UML nicht wirklich domänenspezifisch. Man könnte sagen, die Domäne der UML ist "Softwareentwicklung". Wie mit Programmiersprachen lassen sich damit prinzipiell beliebige Systeme beschreiben. Allerdings erreicht man nicht die Vorteile der Domänenorientierung. Man kann die UML mittels Profilen zwar beliebig anpassen, in real existierenden Tools sind diese Anpassungsmöglichkeiten aber sehr beschränkt. Außerdem lassen sich mittels Profilen nur Dinge hinzufügen und nicht wegnehmen (man kann sie nur durch Constraints verbieten).