Public Beta von Jetbrains' Meta Programming System (MPS)

Version 1.0, Dez, 2008

Markus Völter, Independent/itemis
(voelter@acm.org)

Abstract

Jetbrains, der Hersteller von IntelliJ IDEA, Resharper und anderen Entwicklungwerkzeugen haben am 10. Dezember die Beta-Version von MPS, dem Meta Programming System, veröffentlicht.

MPS ist ein System zur sprachorientierten Programmierung (Language Oriented Programming). Mit MPS lassen sich effizient domänenspezifische Sprachen erstellen, kombinieren und verwenden.

Hintergrund

Domänenspezifische Sprachen (DSLs) sind Sprachen, die auf eine bestimmte Anwendungsdomäne zugeschnitten sind. Eine DSL kann deshalb sehr viel näher an den Abstraktionen und Notationen der Fachdomäne ausgerichtet werden. Dies führt dazu, dass domänenspezifische Sachverhalte sehr viel präziser und knapper formuliert werden können als mit „normalen" Programmiersprachen. Um ein in einer DSL geschriebenes Programm ausführen zu können, muss es – wie bei jeder anderen Sprache auch – entweder interpretiert oder in eine anderweitig ausführbare Form gebracht werden, meist wird Code in einer 3GL wie Java, C oder C# generiert.

Man kann DSLs nach verschiedenen Kriterien unterscheiden. Ein Kriterium ist die Notation: ist sie grafisch (Box-and-Line) oder textuell? Das zweite Kriterium betrifft die Verarbeitung bzw. Ausführung der Programme: interpretiert oder generativ? Das dritte Kriterium unterscheidet DSLs die in Hostsprachen eingebettet sind (interne, oder eingebettete DSLs) von solchen die für sich alleine stehen (externe DSLs).

Was ist MPS

Bei MPS handelt es sich um ein System zur Implementierung textueller, generativer, externer DSLs. Jedenfalls in erster Näherung – die Aussagen benötigen einige Erläuterungen.

Derzeit lassen sich mit MPS nur textuelle DSLs erstellen, allerdings erlaubt die Architektur prinzipiell auch grafische. Dazu später mehr. Die Ausführung der Programme geschieht durch Reduktion, also potentiell mehrstufige, inkrementelle Abbildung auf Java.

Interessant ist die Klassifizierung bzgl. interner und externer DSLs. Prinzipiell sind MPS-DSLs externe DSLs. Sie sind also nicht in anderen Sprachen eingebettet. Allerdings liegt eine Stärke von MPS in der Modularisierung und der Wiederverwendbarkeit von Sprachen. Beispielsweise kann man Sprachen bauen, die andere Sprachen spezialisieren und Teile der „Obersprache“ wiederverwenden. Man kann auch neue Sprachkonzepte in existierende Sprachen integrieren.

Besonders elegant ist, dass MPS mit der sogenannten *BaseLanguage* ausgeliefert wird, einer Art aufgebohrtem Java. BaseLanguage verhält sich wie jede andere mit MPS erstellte Sprache. Daraus ergeben sich faszinierende Möglichkeiten:

- Zunächst kann man – vergleichbar mit beispielsweise openArchitectureWare's Xtext – schlicht seine eigenen DSLs bauen. Dazu definiert man sein eigenes Metamodell (MPS-Terminologie: Concepts) und die konkrete Syntax (Editor).
- Alternativ kann man in seine eigene neue Sprache aber auch andere Sprachen (oder Teile anderer Sprachen) einbetten. Man stelle sich eine textuelle DSL für Zustandsmaschinen vor. Die Sprache besteht aus den Konzepten Zustandsmaschine, Zustand, Transition, Event, usw. Wenn man nun eine Guard-Expression an die Transition anbringen will, so kann man einfach Java's Expression-Teilsprache an dieser Stelle einhängen und hat sofort deren komplette Mächtigkeit zur Verfügung.
- Der dritte Use Case besteht darin, dass man seine eigenen domänenspezifischen Konzepte in Java einbettet – so eine Art interne DSLs in Java. Diese sind aber im Gegensatz zu Ruby's internen DSLs nicht interpretiert, sondern werden transparent in Basis-Java übersetzt und dann compiliert. Natürlich kann man für die eigenen Konzepte auch Compiler-Fehlermeldungen (Constraints), Typgleichungen (zur Integration ins Java Typsystem) und Datenfluss-Definitionen (zwecks Dead-Code Analysis) definieren.

Insbesondere durch den dritten Ansatz erlaubt MPS eine völlig neuartige Herangehensweise an DSLs. Man startet mit der Basisprogrammiersprache (*BaseLanguage* bzw. Java) und ergänzt diese inkrementell um domänenspezifische Konzepte (oder beliebige andere Idiome) die sich im

Laufe eines Projektes herauskristallisieren. Im Gegensatz zum entsprechenden Ansatz in Ruby oder anderen dynamischen Sprachen kann man weiterhin mit einem statischen Typsystem und den entsprechende IDE-Annehmlichkeiten rechnen. Als Konsequenz aus der Art und Weise, wie MPS funktioniert (siehe nächster Absatz) geht eine Spracherweiterung automatisch mit einer IDE-Erweiterung einher. Die neue Syntax verhält sich bezüglich des Toolings exakt so, wie die Syntax der Basissprache.

Wie funktioniert das ganze?

Das Kernprinzip von MPS ist Structured Editing. Im Gegensatz zu klassischen Programmiersprachen bearbeitet man in MPS keinen Quelltext, sondern arbeitet direkt auf dem Syntaxbaum, der allerdings genauso aussieht wie Quelltext und sich auch *fast* so verhält. Wenn man beispielsweise *int i;* eingibt, so erstellt man tatsächlich eine Instanz des Konzepts *VariableDeclaration* dessen Namensattribut den Wert „i" hat. Ein klassischer Parsevorgang findet nie statt. Die Vermeidung des Lexens und Parsens hat den Vorteil, dass man bei der Kombination von Sprachen keine Grammatiken verknüpfen muss. Die Verknüpfung von Grammatiken resultiert bei klassischen Parsern leicht in Grammatiken, die nicht mehr eindeutig sind, was die Kombinationsfähigkeit von Grammatiken stark einschränkt.

Um beispielsweise ein neues Schlüsselwort in Java einzuführen, geht man folgendermaßen vor: Man definiert eine neue Sprache die von *BaseLanguage* (MPS' Java) erbt. In dieser neuen Sprache definiert man ein neues Konzept welches seinerseits von dem Java- Konzept *Statement* erbt. Dann definiert man einen Editor, der die syntaktische Struktur des neuen Statements definiert. Wenn man dann eine Instanz (==Programm) dieser neuen Sprache anlegt, kann man das neue Statement überall dort verwenden, wo Java ein *Statement* erwartet. Um es dann auch ausführen zu können, muss man noch Reduktionsregeln definieren (das sind mehr oder weniger Codegenerierungstemplates) die das neue Sprachkonzept auf original Java Code zurückführen.

Wo ist der Haken?

Weil natürlich im richtigen Leben alles seine Nachteile hat, sollten wir auch die von MPS betrachten. Zum einen merkt man beim Editieren schon, dass man nicht mit „echtem Text" arbeitet, sondern mit einen Projektion des Baumes in Form von Text. Jetbrains haben es zwar recht gut hinbekommen, und ich bin überzeugt, dass man sich daran gewöhnen kann, aber man merkt den Unterschied.

Zum anderen wird das Programm natürlich nicht als Quelltext sondern als serialisierter Syntaxbaum abgespeichert (in Form von XML). Das bedeutet, dass man nicht ohne weiteres mit anderen, textbasierten Werkzeugen interoperieren kann. Auch diff/merge sollte man in MPS durchführen,

damit man es auf der Ebene der konkreten Syntax (und nicht in einem XML Baum!) tun kann. MPS kommt mit einer Integration in die üblichen Sourcecode Management Systeme – aus deren Sicht sind MPS Resourcen einfach XML Dateien.

Im ober erwähnten dritten Use Case (domänenspezifische Erweiterung einer Basissprache) steht derzeit nur Java als Basis zur Verfügung. Wer beispielsweise domänenspezifische Erweiterungen an C durchführen wollte, der müsste zunächst die Spache C MPS-ifizieren. Ich gehe davon aus, dass es im Lauf der Zeit neben *BaseLanguage* (also Java) noch weitere Sprachen geben wird die man spezialisieren kann.

Eine Runde Sache

MPS existiert schon seit einer ganzen Weile innerhalb von Jetbrains; die haben es bereits zur Entwicklung mehrerer Produkte verwendet (unter anderem ist MPS zum Großteil selbst mit MPS entwickelt), und es ist auch schon seit einer Weile als (halböffentliches) Early Access Program verfügbar – in dieser Phase hatte ich es mir immer wieder angeschaut, hatte aber Probleme damit klar zu kommen. So richtig intuitiv ist es nicht. Inzwischen haben sie aber einen ausführlichen User Guide und ein richtig gutes Tutorial erstellt, sodass man eine realistische Chance hat, das Tool zu verstehen. Ein Diskussionforum steht zwecks Support auch zur Verfügung.

Die richtig gute Nachricht ist, dass Jetbrains sich dazu entschlossen haben, das Produkt größtenteils als Open Source unter der Apache 2.0 Lizenz zur Verfügung zu stellen (nur das aus IntelliJ extrahierte IDE Framework ist nicht offen). Es kann also auch für kommerzielle Entwicklung kostenfrei eingesetzt werden. Mit einem 1.0-er Release ist in Q1 2009 zu rechnen.

Fazit

Ich persönlich finde MPS ein sehr interessantes Produkt und werde mich sicher damit beschäftigen. Es ist eine nützliche und innovative Ergänzung in der Welt der DSL Tools und eröffnet völlig neue Möglichkeiten (siehe Use Case 3). Die letzten Ecken und Kanten wird Jetbrains im Rahmen der Betaphase fixen – Feedback ist dabei sicher willkommen!