Interview mit Markus Völter

DSLs – so was von offensichtlich!?

Markus Völter spricht mit OBJEKTspektrum über Domain-Specific Languages, die helfen, die Trennung zwischen Fachexperten und Informatikern zu überwinden, und es ermöglichen, die Fachexperten in die Entwicklung direkt mit einzubeziehen.

MAINUSCH INTERVIEWT DR. MARKUS VÖLTER

Johannes Mainusch: Markus, ich kenne dich durch deinen Podcast Omega Tau, den du jetzt seit 12 Jahren machst. Außerdem bist du Segelflieger. Aber im Herzen, wahrscheinlich um das alles zu finanzieren, bist du Softwareentwickler.

Markus Völter: Eigentlich habe ich Technische Physik an der Fachhochschule in Weingarten studiert, aber ich habe das nie praktiziert. Da ich als Schüler und Student immer nebenher Software entwickelte, habe ich dann in Informatik promoviert und nicht in Physik.

Und das mit einer Arbeit zum Thema DSL, also Domain-Specific Languages?

Ja, das erweiterbare C. Wir haben quasi C in MPS implementiert und dann jede Menge C-Erweiterungen gebaut, die für die Entwicklung eingebetteter Systeme geeignet sind. Damit haben wir die Sprache C besser an die Anforderungen der Embedded-Entwicklung angepasst.

„Wenn ich mal programmieren darf, dann meistens in MPS"

C kenne ich, MPS sagt mir gerade nichts.

Meta Programming System, also das Sprachbautool von JetBrains, mit dem ich seit 2010 eigentlich ausschließlich arbeite. Wenn ich mal programmieren darf, dann meistens in MPS.

Also, MPS ist eines der Frameworks, mit dem man DSLs implementieren, pflegen und dann in funktionsfähige Programme übersetzen kann.

Genau. Und IDEs, also Entwicklungsumgebungen, baut man damit auch. Man will ja, wenn man DSLs baut, nicht nur die Sprache erstellen und irgendwelche Interpreter oder Generatoren, sondern man möchte dem Benutzer der Sprache auch ein modernes Entwicklungswerkzeug in Form einer IDE zur Verfügung stellen. Also Dinge wie Code-Completion, Syntax-Highlighting, Refactoring, Debugger usw. Das geht mit MPS fast automatisch. Man kann in MPS gar keine Sprache definieren, ohne dass dabei nicht auch eine IDE entsteht.

Wenn man sich eine Skala denkt, auf der links eine einfache Programmiersprache wie etwa TypeScript steht und rechts eine natürliche Sprache wie Deutsch, wo auf der Skala liegt dann eine DSL?

Was ist denn die Achse, auf der du die beiden Beispiele TypeScript und Deutsch aufgetragen hast, also welche Dimension repräsentiert diese Links-rechts-Skala?

Also nehmen wir erst mal die Dimension Anzahl Sprachbausteine, also rechts ist Subjekt-Prädikat-Objekt und links sind drei Sprachelemente Abfolge, Verzweigung und Schleife.

Oder nimm links Lisp: Atom, Liste, Funktionsaufruf, Lambda-Expression, viel mehr hat Lisp ja wirklich nicht. Das ist eine interessante Diskussion. Lisp zeichnet sich dadurch aus, dass es eine Sprache ist, die sehr sehr wenige First-class-Konzepte hat. Die sind aber so gewählt, dass man damit im Prinzip alles andere zusammensetzen kann. Also wenige, sehr mächtige und orthogonal kombinierbare Sprachbausteine. Die meisten DSLs verwenden einen anderen Stil, in dem die Sprache mehr Sprachkonzepte hat, die weniger mächtig bezüglich ihrer Composability sind, aber die mehr Annahmen und Wissen und damit Semantik über die Domäne enthalten.
Konkretes Beispiel: Wir bauen gerade eine DSL zur Steuerberechnung, und da gibt es das Äquivalent einer Variablen, das nennt sich Tax Entry. Jeder dieser Tax Entries hat aber auch direkt Rundungsvorschriften und Limits. Wenn die Formel zur Berechnung einen Wert ergäbe, der über oder unter dem Limit läge, wird abgeschnitten. Dazu hat sie auch Validierungsregeln direkt assoziiert mit der Variablen. Das heißt, man hat mehr domänenspezifische Struktur direkt in der Sprache implementiert.

„Wir bauen gerade eine DSL zur Steuerberechnung"

(lacht) ... bei Rundungsregel muss ich lachen: Ich versuchte kürzlich, die Rundungsregeln bei der Gewebesteuerermittlung für unsere Firma nachzuprogrammieren. Die ist wirklich endkaputt.

(lacht) Also, eins hab ich in diesem Projekt gelernt: Die Steuergesetzgebung ist komplex und es ist kein Wunder, dass Firmen, die Steuersoftware entwickeln, viele Mitarbeiter beschäftigen, um das alles irgendwie zu verstehen, zu entwirren und dann in Software zu gießen. Und um den

Dr. Markus Völter

Begeistert sich für:
Segelfliegen, Wissenschaft & Technik

Produziert:
Omega Tau. Der mehrmals pro Monat erscheinende Podcast von Nora Ludewig und Markus Völter zu Wissenschaft und Technik

Ist Experte für:
Domain-Specific Languages und modellgetriebene Softwareentwicklung

Sprachen (unvollständige Liste):
Schwäbisch, Deutsch, Englisch, C, Java, MPS, diverse DSLs, ...

Letztes Buch:
Once You Start Asking: Insights, stories and experiences from ten years of reporting on science and engineering

Bogen zu schließen: Genau deshalb bauen wir ja eine DSL, damit die Leute, die die Steuerlogik verstanden haben, diese dann direkt mit einer DSL hinschreiben können. Ganz wichtig ist, dass die Fachexperten gleich mit der DSL testen können, sodass der Workflow vom Steuergesetz in die Software nur über den Fachexperten und seine Tools läuft. So wird dann der technisch geprägte Softwareentwickler, der sich um Plattformen und Frameworks kümmert, nicht mehr gebraucht, um jedes bisschen konkrete Fachlogik umzusetzen.

Okay, also der Fachexperte hat in seinen DSL-Werkzeugen links seine Sprachregeln, dann drückt er aufs Knöpfchen und anschließend kann er rechts sehen, ob alles funktioniert und ob die hinterlegten Tests alle grün gelaufen sind.

Genau, er schreibt mittels der DSL ein semantisch reichhaltiges, gut verständliches und syntaktisch möglichst nah am Steuergesetz liegendes Programm, plus die Tests dazu. Dann fährt er solange im Kreis, bis die Tests grün sind und bis er eine sinnvolle Testabdeckung hat. Ich habe dazu einen Screenshot mitgebracht :-)

Und dann?

Dann checkt er das Zeug ein und übergibt es an den Buildserver, der den ganzen Kram übersetzt, nach C, Java oder TypeScript, und schließlich landet das in ausführbarer Software, die der Steuerberater verwendet, um für Leute wie dich tatsächlich die Gewerbesteuer zu berechnen.

„Das Ganze muss langlebig genug sein, damit sich der Aufwand für die DSL lohnt"

Lohnt sich das?

Bei meinem Kunden gibt es schon seit Jahrzehnten einen in Teilen DSL-basierten Entwicklungsprozess. Diese Trennung zwischen Fachlern und Technikern wird in deren Abteilungen unterschiedlich gelebt. So entstanden über die Jahre mehrere Generationen von Sprachen, die unterschiedlich viel DS im L haben. Die, die wir jetzt bauen, ist erheblich domänenspezifischer – mit sehr viel mehr deklarativen Elementen.
Die Charakteristika, die so eine Domäne haben muss, damit DSLs sich im Einsatz lohnen, sind folgende: Das System muss eine gewisse Mindestkomplexität haben, eine Gruppe von Fachexperten muss vorhanden sein, also etwa Steuerexperten, es muss eine gewisse Änderungsrate geben, etwa die jährlichen gesetzlichen Anpassungen, und das Ganze muss langlebig genug sein, damit sich der Aufwand lohnt. Wenn du jetzt so eine klassische Consul-

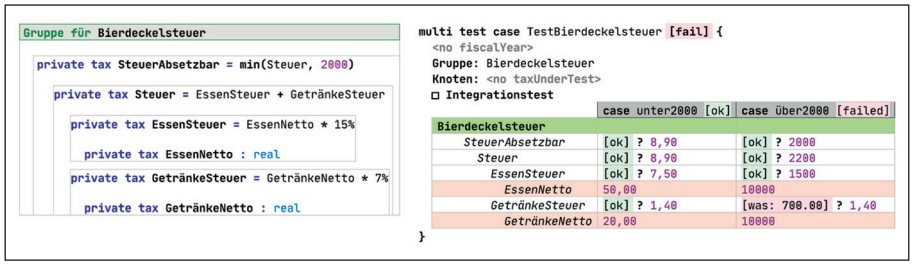


Abb. 1: Screenshot von Programm und Test

ting-Bude bist, die vierteljährlich ein anderes Projekt in einer anderen Domäne bearbeitet, dann wirst du dir keine DSLs bauen, das macht einfach keinen Sinn.

Verstehe. Die Experten, die DSLs entwickeln, das ist doch in Deutschland wirklich eine sehr kleine Community, oder?

Ja, DSLs entwickeln, dazu braucht es Spezialwissen. DSLs sind ein Ansatz, der es bisher nicht in den Mainstream der Softwareentwicklung geschafft hat, was mich massiv frustriert. Und zwar gar nicht aus meiner persönlichen Perspektive, sondern weil es für mich so was von offensichtlich ist, dass man so vorgehen sollte. Ich verstehe nicht, warum das nicht öfter gemacht wird.

Nun redest du mit mir mit jemandem, der noch in der Höhle sitzt. Du kannst zehnmal sagen, da draußen ist die Welt schöner. Welchen Weg müsste ich als TypeScript-Hacker beschreiten, um die Erleuchtung der DSLs zu sehen?

Da sind drei Dinge wesentlich: Das Erste ist eine Sensibilisierung für den Kontext, wo es sinnvoll ist. Darüber haben wir gerade geredet.
Zweitens muss man verstehen, dass die Werkzeuge, mit denen man solche DSLs und die nachgelagerten Tools baut, einfach nicht die sind, die man vom Informatikstudium noch aus dem Compilerbau-Kurs in Erinnerung hat. Anstelle von antlr, lex oder yacc gibt es heute viel ausgereiftere und bessere Werkzeuge, beispielsweise Xtext oder MPS. Damit hat man beim Sprachbau eine ganz andere Produktivität.
Das Dritte, was man tun muss, ist, einfach ein paar Vorurteile beiseite zu räumen. Ich provoziere hier bewusst etwas: Ich höre immer wieder, wir sind eine Firma in der Domäne X, wir bauen keine Tools. Toolbau widerspricht agilen Ansätzen, wir machen jetzt alles mit Prozessen und wir reden viel miteinander. Und deshalb sind wir jetzt auf einmal produktiv. Aber auch im Agilen legt man ja sehr viel Wert auf Automation.

„Abstraktion ermöglicht Automation"

Also Automation weitergedacht.

Ja klar. Und Abstraktion ermöglicht Automation. Die Informatik und die Programmiersprachenentwicklung über die Jahrzehnte, was haben die denn gemacht? Die haben immer höhere Abstraktionen gefunden, und dadurch wurden wir produktiver.

Die große Abstraktion, die wir haben, ist ja die natürliche Sprache ...

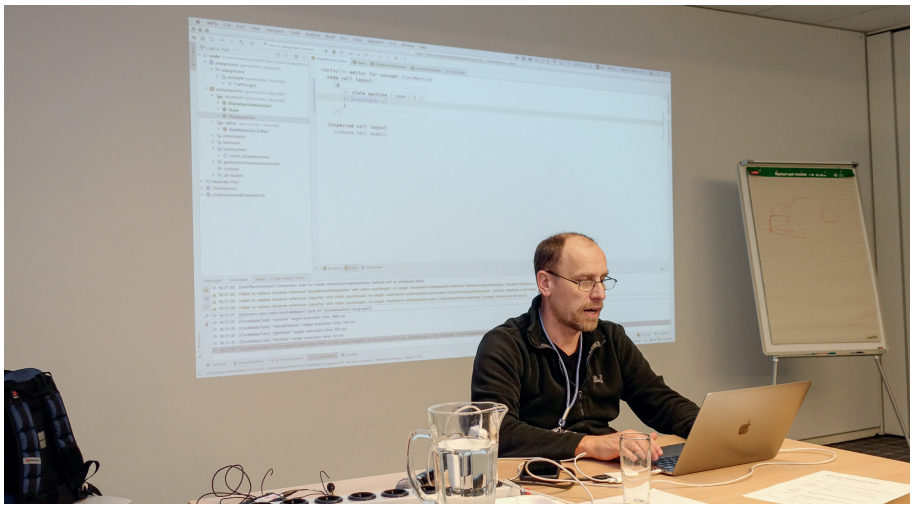
Da musst du aufpassen, weil natürliche Sprache ein bisschen auf einer anderen Achse liegt. Eine natürliche Sprache auf semantischen Inhalt und Eindeutigkeit zu analysieren, ist sehr kompliziert. Eine DSL wird anders gebaut, und zwar so, dass solche Analysen sehr einfach sind …

Das ist die Grundvoraussetzung überhaupt, wenn du im Computersegment unterwegs bist. Also Ambiguitäten vermeiden, sodass es – vielleicht mit der Ausnahme von KI – eine Nachvollziehbarkeit und eine Eindeutigkeit von Funktionen gibt. Es heißt ja nicht umsonst formale Sprache, also auch ein DSL wird am Ende des Tages eine formale Sprache sein.

Ja natürlich. Jede Sprache, die ich mit einem Metamodell oder einem Generator definiere, ist formal in dem Sinne, dass sie bezüglich ihrer Semantikdefinition eindeutig ist, sonst funktioniert die Mechanik der Idee gar nicht.
Aber wenn du dich beispielsweise mal in einen Compiler hineinversetzt, dann musst du als Compiler die Semantik nur insoweit verstehen, dass du daraus Maschinensprache generieren kannst. Und die Maschinensprache ist weniger abstrakt, das heißt, du musst eigentlich nur nach unten übersetzen. Wenn du jetzt aber auf – sagen wir mal – so einem Java- oder C++-Programm Analysen machst, die eben nicht nach unten relevant sind, sondern nach oben, also die quasi die abstraktere Semantik deiner Domäne betreffen, dann ist das extrem aufwendig. Will man etwa die semantische Bedeutung aus Legacy-Code herauslesen, so kann man das mit modernen Code-Analysetools versuchen. Aber es ist ein Riesenaufwand, um aus der Abstraktion, die der Entwickler in Java reinschreibt, wieder abstraktere Bedeutung herauszudestillieren.
Deshalb sind DSLs genau so gebaut, dass sie einerseits nach unten formal übersetzbar und andererseits nach oben hin semantisch reichhaltig sind.

Bedeutet semantisch reichhaltig nur, dass sie menschenverständlich sind?

Nein, sie müssen toolverständlich sein. Wenn ich einem Nutzer einer DSL eine Fehlermeldung gebe, dann will ich ihm ja nicht sagen: Hey, ich Compiler verstehe etwas nicht in deinem Programm, ich kann es nicht nach Java übersetzen. Ich möchte vielmehr sagen: Hey, du hast gerade so eine Steuer-Variable definiert in dem



Sinn, die ich vorhin als Beispiel erwähnt hatte, aber die Rundungsregel passt nicht zu der Rundungsregel bei der anderen Variablen da drüben. Deshalb muss die DSL bezüglich der Domäne semantisch reichhaltig sein, und das schaffst du nur – der Kreis schließt sich –, wenn du Konzepte der Domäne als First-class-Konstrukte verstehst. Deshalb haben DSLs oftmals mehr Schlüsselworte als Lisp, denn Lisp ist eben semantisch eindeutig nur nach unten.

Gib mal ein Beispiel aus der Steuerdomäne.

Wenn du zum Beispiel zwei Zahlen verrechnest, die unterschiedlich gerundet sind, dann kann es zu komischen Artefakten kommen, also Rechenfehler. Oder wir haben Sprachkonstrukte, die direkt ausdrücken, dass eine Variable nicht einfach nur ein Wert ist, sondern dass sie einen Wert pro Jahr oder pro Monat hat, sodass man dann Arithmetik auf solchen Jahres- und Monatsverläufen machen kann, ohne dass man das jeweils ausprogrammieren muss, denn diese Semantik, dass es sich um jährliche oder monatliche Strukturen handelt, steckt schon im Sprachkonstrukt. Damit muss man weniger hinschreiben, der Interpreter kann automatisch mehr Dinge tun, die Fehlermeldungen können jetzt von Jahren und Monaten reden und nicht von Listenindex 7.

Wie verbreitet man dieses Wissen und schafft eine größere Gemeinde an DS-Lern?

Mich darfst du das nicht fragen. Ich habe es die letzten 10 Jahre versucht und bin mehr oder weniger gescheitert, eine größere Awareness in der breiten Informatiker-Community zu erzeugen. Bei der itemis AG zum Beispiel haben wir über die letzten 10 Jahre ein Team von 3 auf 20 Leute aufgestockt, und es gibt inzwischen auch andere Firmen, die uns Konkurrenz machen. Das heißt, da hat es sich zumindest mal herumgesprochen. Was ich verbreiten möchte, ist diese Idee der Separation of Fachlichkeits-Concern von Technik-Concern, um damit den Entwicklungsprozess für Fachlichkeit zu beschleunigen. Das ist mit Zahlen nachweisbar. Ich denke, der Hauptgrund ist, die Informatiker glauben nicht, dass es funktioniert.

Jetzt musst du die 10 Erfolgsbeispiele bringen, damit wir die Entwickler-Community vergrößern.

Ja gut, soll ich jetzt Firmen zitieren … Also bei DATEV sind wir seit Jahren unterwegs. Wir haben auch im Systems-Engineering-Umfeld einiges gebaut, von der Satellitensteuerung über Fahrtenschreiber (das waren zwar teils nicht wir, sondern „befreundete“ Firmen, aber ist ja auch egal, das ist trotzdem ein Erfolg für den Ansatz). Die niederländische Steuerbehörde macht eigentlich seit Jahren ihre komplette Public-Benefits-Berechnung so, und arbeitet grade auch an den Steuerberechnungen. Die BA (Bundesanstalt für Arbeit) setzte DSL schon lange ein. Außerdem haben wir Projekte im Medizinumfeld, wie die Firma Voluntis, die mit dem Ansatz digital therapeutics applications baut …

Du hast in einem Vortrag gesagt, es ließen sich Faktoren von 50 Prozent bis Faktor 10 an Vereinfachung der Umsetzung oder Implementierung neuer Regeln erzielen.

Absolut. Wir haben Fälle, gerade im Bereich Digital Therapeutics, da haben wir Zahlen in einem Paper, das ich zusammen mit Bernd Kolb von der itemis AG und anderen geschrieben habe, wo Dinge wirklich von Wochen auf wenige Stunden Aufwand reduziert wurden. Dabei muss man bedenken, dass die natürlich in einem sicherheitskritischen und stark regulierten Umfeld unterwegs sind, das viel Dokumentation und Test-Overhead hat, und der lässt sich sehr einfach automatisieren.

Abb. 2: http://voelter.de/data/pub/MPS-in-Safety-1.0.pdf

Wie wäre es denn, einfach das uralte Überzeugungsmittel zu nehmen, ihr lasst euch einfach dreimal besser bezahlen als heute und erzählt es allen.

Ja klar, den Effekt gibt es natürlich, dass man nicht darüber redet, dass man Dinge so macht, und einfach den marktüblichen Preis nimmt, aber natürlich dann fünfmal schneller ist in der Implementierung.

Den Schieberegler von Lisp zu Schwäbisch, den haben wir jetzt auf der Skala von 0 auf 10 cm um ca. 3 mm nach rechts geschoben, oder?

Jaa, ich tue mich tatsächlich mit diesem Schieberegler relativ schwer, denn natürliche Sprachen und DSLs entlang einer Dimension zu vergleichen ist schwierig. Man bräuchte einen fünfdimensionalen Schieberegler.

Markus, vielen Dank für das Gespräch!

Das Interview führte …

Dr. Johannes Mainusch
(johannes.mainusch@kommitment.works)
Berater für Unternehmen, die Bedarf im Bereich IT, Architektur und agiles Management haben. Dr. Mainusch ist seit 2012 Mitglied der OBJEKTspektrum-Redaktion.