Erweiterung von Domänenspezifischen Sprachen um benutzerdefinierte Werttypen

Christin Zahner

Universität Hamburg, Fachbereich Informatik
Vogt-Koelln-Str. 30
22527 Hamburg, Germany
christin.zahner@informatik.uni-hamburg.de

Abstract: Domänenspezifische Sprachen unterstützen die Modellierung von Konzepten einer bestimmten Domäne. Mit Blick auf die Formulierung von Ausdrücken beschränken sich textuelle DSLs allerdings häufig auf die Unterstützung von primitiven Typen und Aufzählungstypen. Fachliche Konzepte, wie Geldbeträge oder Postleitzahlen, die zeit- und zustandslos modellierbar wären, sind nur schwer in eine DSL zu integrieren. Dieser Beitrag stellt Ansätze vor, um die Ausdruckskraft von DSLs um benutzerdefinierte Werttypen zu erweitern.

1 Ausgangssituation

Domänenspezifische Sprachen (DSL) versprechen mit ihrer limitierten Ausdruckskraft und ihrem Fokus auf eine bestimmte Domäne Vorteile gegenüber allgemeinen Programmiersprachen wie z.B. Java. Dazu gehören eine höhere Produktivität in der Programmierung, geringere Wartungskosten und eine verbesserte Kommunikation mit Domänenexperten [MH+05, Fo10]. Die *Modellgetriebene Softwareentwicklung (MDSD)* verfolgt ein ähnliches Ziel. Mit Hilfe von Modellen wird versucht auf einer höheren Abstraktionsebene zu arbeiten, als dies im Vergleich zu allgemeinen Programmiersprachen möglich ist [SV+07, KT08]. Die Modelle dienen dabei nicht mehr nur der Dokumentation, sondern werden auch für Transformationen oder zur Code-Generierung eingesetzt. Grundlegende Idee ist der Umgang mit Modellen auf verschiedenen Metaebenen. Ein Metamodell dient dabei zur Spezifikation einer Modellierungssprache [Se03]. Um die Kompatibilität zwischen Metamodellen sicherzustellen, wird eine gemeinsame Metasprache verwendet, ein Meta-Metamodell. Diese Einteilung findet sich in ähnlicher Weise bei Programmiersprachen wieder [Be01], vgl. Abbildung 1. Metasprache ist hier bspw. die *Erweiterte Backus-Naur-Form (EBNF)* und die Grammatik einer Programmiersprache entspricht ihrem Metamodell.

Im MDSD-Umfeld werden typischerweise Metasprachen eingesetzt, die auf objektorientierten Konzepten basieren. Bekanntes Beispiel ist die Meta Object Facility (MOF)[1]. Nicht alle Konzepte einer Domäne müssen jedoch zustandsbehaftet als Objekt

[1] http://www.omg.org/spec/MOF/

modelliert werden. Beispielsweise ist der Geldbetrag „5,00 EUR“ in den meisten Anwendungsfällen unveränderbar und wird nicht erzeugt. Auch hängen Operationen, wie bspw. die Addition zweier Geldbeträge, von keinem Zustand ab, ändern ihn nicht und liefern stets dasselbe Ergebnis. Eine Abstraktion in Form von zeit- und zustandslosen Werten kann hier helfen, die Konzepte der Domäne besser abzubilden. Daher wird u.a. in [Ma82, Zü04, Ev03] eine konzeptionelle Trennung zwischen *Objekt- und Werttypen* gemacht.

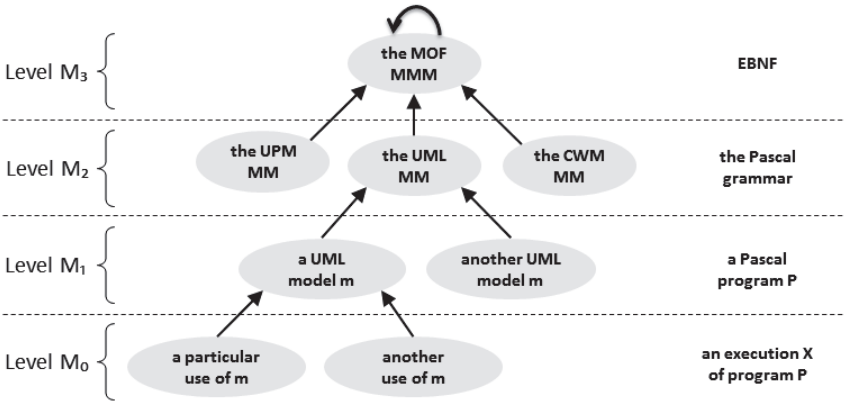

Abbildung 1: Metaebenen in der Modellierung und Programmierung am Beispiel der Modellierungssprache UML und der Programmiersprache Pascal („The OMG four layers standard modeling stack“ in [Be01])

2 Problemstellung

Sollen in einer DSL Berechnungen oder Bedingungen formulierbar sein, so muss der Sprachumfang Ausdrücke unterstützen. Für benutzerdefinierte Objekttypen ist die Verwendung in einer DSL auf ähnliche Weise denkbar, wie sie in objektorientierten Programmiersprachen erreicht wird: Syntaktisch sind allgemeine Sprachkonstrukte bspw. für Methodenaufrufe mittels Punktnotation vorstellbar. Im Rahmen der semantischen Analyse lässt sich zudem ein polymorphes Typsystem mit Konzepten wie Vererbung und dynamischem Binden implementieren. Für Werttypen ist diese Erweiterung um neue Typen ungleich schwieriger. Folgende Ansätze sind dafür vorstellbar:

a) Erweiterung der DSL-Werkzeuge (Lexer, Parser, Typsystem, etc.) mit jedem neuen Typ

b) Modellierung als Objekttyp

c) Abbildung auf primitive Typen (z.B. String, Boolean, Integer)

Variante a) verhindert die dynamische Erweiterung der DSL, da mit jedem neuen Werttyp die DSL-Werkzeuge um typspezifische Regeln u.a. zu Literalen, Operationen und Typumwandlungen erweitert werden müssen. Dieses Regelwerk wächst stetig und kann schnell sehr umfangreich und komplex werden, da es nicht von einzelnen Werttypen abstrahiert. Variante b) führt dazu, dass die wesentlichen Eigenschaften eines Werttyps nicht mehr durch die DSL sichergestellt werden können, z.B. seine Unveränderbarkeit. Zudem ist die Syntax sehr eingeschränkt. Variante c) wird häufig für DSLs gewählt. Jedoch geht dadurch die Fachlichkeit in der Formulierung von Ausdrücken einer DSL verloren. Der gewählte primitive Typ kann nicht die fachlichen Anforderungen an zulässige Literale und Operationen abbilden. Der arithmetische Ausdruck „5,00 * 0,19“ sagt nichts über die fachliche Bedeutung seiner Literale und Operationen aus. Es könnte sich um die Berechnung der Mehrwertsteuer handeln oder jede beliebige andere Berechnung. Die DSL bleibt in diesem Punkt eine allgemeine Programmiersprache.

3 Forschungsfragen

Ziel der vorgestellten Promotion ist es, die Ausdruckskraft textueller DSLs um benutzerdefinierte Werttypen zu erweitern, ohne dass eine Anpassung der DSL-Werkzeuge erfolgen muss. Die folgenden Fragen gilt es dafür zu beantworten.

1) Welche Informationen muss das Metamodell eines Werttyps enthalten, damit neue Werttypen modellierbar und in Ausdrücken einer DSL verwendbar sind?

2) Wie kann die Implementierung der syntaktischen und semantischen Analyse einer DSL von konkreten Werttypen abstrahieren? Welche allgemeinen Regeln für einen Werttyp können abgeleitet werden?

3) Wie lässt sich die Syntax einer DSL dynamisch um Literale und Operationen eines Werttyps erweitern?

4 Lösungsvorschläge

Zur Lösung der vorgestellten Problemstellung wird zunächst die Modellierung der Schnittstelle eines fachlichen Werttyps betrachtet. Um diese in einer DSL verwenden zu können wird dann die Integration auf semantischer Ebene untersucht. Abschließend wird die syntaktische Repräsentation eines Werttyps betrachtet.

4.1 Modellierung der Schnittstelle fachlicher Werttypen

Für die Modellierung eines neuen Werttyps ist ein passendes Metamodell notwendig, auf dem eine Modellierungssprache basieren kann. Mit Blick auf die bereits erwähnten objektorientierten Meta-Metamodelle gibt es dort eine Meta-Beschreibung sogenannter

„DataTypes“. Abbildung 2 zeigt einen Auszug aus der UML Infrastructure, die Basis der MOF ist. Mit objektorientierten Mitteln wird hier ein Werttyp samt seiner Eigenschaften modelliert. Ähnliche Ansätze existieren, um Werttypen in objektorientierten Programmiersprachen zu beschreiben, vgl. „Value Object“-Pattern in [Fo03, Ev03]. Auffallend ist, dass einige Eigenschaften von Werttypen hier nicht enthalten sind. Bspw. fehlen Subtypbeziehungen oder mögliche Literale eines nichtabzählbaren Werttyps. Der Fokus liegt auf primitiven Typen und Aufzählungstypen.

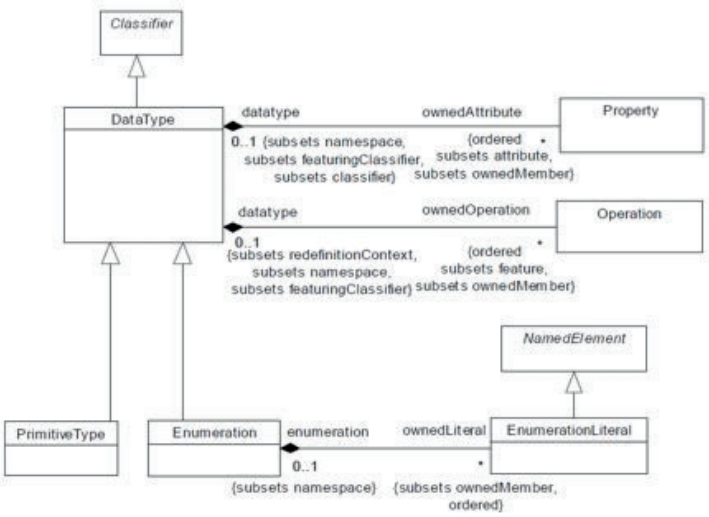

Abbildung 2: DataTypes in der UML Infrastructure [UML11]

Als Lösung wird eine entsprechende Erweiterung des Werttyp-Metamodells angestrebt. Das Meta-Metamodell der UML DataTypes dient dafür als Ausgangsbasis. Ziel ist es, die Schnittstelle eines Werttyps und damit seine abstrakte Syntax abzubilden.

4.2 Semantische Analyse: Integration von modellierten Werttypen in DSLs

Soll ein modellierter Werttyp nun in einer textuellen DSL verwendet werden, so müssen die zur Definition des Werttyps genutzten Modellierungskonzepte in die DSL integriert werden. In [KT08] werden verschiedene Integrationsansätze für Modellierungssprachen diskutiert. Für die vorliegende Problemstellung eignet sich insbesondere die Integration basierend auf einem gemeinsamen Metamodell.

Operationen eines Werttyps lassen sich bspw. in Form von binary methods [BL+95] modellieren. Einen ähnlichen Ansatz verfolgt u.a. Scala, indem Infixoperationen der Form „x + y“ als Methodenaufruf „x.+(y)“ interpretiert werden [OA+04]. Zusätzlich zum Metamodell einer Grammatik basiert die DSL somit auf dem Metamodel des Werttyps, um dessen statische Semantik zu prüfen. Die implementierten Regeln zu möglichen Operationen, zulässigen Operandentypen und Subtypbeziehungen können auf diese Weise von konkreten Werttypen abstrahieren.

4.3 Syntaktische Repräsentation und dynamische Grammatik

Für die syntaktische Repräsentation eines Werttyps muss seine konkrete Syntax innerhalb einer DSL definiert werden. Dies betrifft besonders die zulässigen Formate für Literale. Als Kernidee der Lösung dient hierfür der in [ES+95] beschriebene Ansatz einer Object-Oriented Language Definition. Eine Klassendefinition wird dort um die syntaktische Definition erweitert. Die Summe aller im aktuellen Kontext verwendbaren Objekte bildet dann eine dynamische Grammatik. Bezogen auf die Problemstellung der Werttypen muss also die Klassenbeschreibung des Wertttyps im Metamodell um syntaktische Definitionen erweitert werden. Dies kann bspw. in Form von Annotationen im Metamodell geschehen. Auf diese Weise ist auch die Verwendung des Metamodells in unterschiedlichen DSLs mit unterschiedlicher Syntax denkbar. Basierend auf den Annotationen kann anschließend ein Lexer und Parser generiert werden. Das Hinzufügen eines neuen Werttyps erweitert somit dynamisch auch die Grammatik der DSL.

5 Aktueller Stand der Arbeit

Die Arbeit erfolgt im Rahmen eines Drittmittelprojektes zur Realisierung eines Frameworks für die Modellgetriebenen Softwareentwicklung. Teil dieses Projektes ist die Neuentwicklung einer DSL für eine Regelsprache für Fachanwender. Diese textuelle DSL soll um Typen erweitert werden können, die sich mit Hilfe graphischer Notationen definieren lassen. Als technologische Grundlage kommen der Parser-Generator ANTLR[2] und Java zum Einsatz.

Die beschriebenen Lösungsvorschläge wurden in den vergangenen 2 Jahren im Rahmen der Implementierung dieser Regelsprache konkretisiert. Dafür wurde unter anderem ein Typsystem basierend auf den Informationen eines Werttyp-Metamodells implementiert. Hierbei wurde ebenfalls der Umgang mit primitiven Typen und speziellen Typen wie „null“ oder Sammlungen (Collections) untersucht. Die Lösungsansätze aus Abschnitt 4.1 und 4.2 sind somit in einer ersten Version umgesetzt. Aktuell noch ausstehend ist die Realisierung der dynamischen Erweiterbarkeit der DSL-Syntax. Als nächster Schritt steht hierfür die Formulierung von Annotationen am Werttyp-Metamodell an.

6 Forschungsmethodik, Vorgehen & Innovation

Um die geforderte Erweiterung der DSL um Werttypen zu erreichen, wird ein inkrementelles Vorgehen gewählt. Für diesen Prozess eignet sich insbesondere das Paradigma des *Design Reseach* [HM+04], dessen Fokus auf dem Entwurf und der Evaluation von Artefakten liegt. Dazu zählen Konzepte, Modelle, Methoden aber auch konkrete Realisierungen. Die beschriebene Methodik soll in mehreren Forschungszyklen umgesetzt werden. Hierfür werden Lösungen auf Basis der erwähnten Regelsprache erarbeitet. Die gewonnenen Ergebnisse sollen anschließend verallgemeinert, auf weitere Anwendungsfälle übertragen und somit evaluiert werden. Die Verallgemeinerung der

[2] http://www.antlr.org/

Lösung ist in Form einer allgemeinen Ausdruckssprache geplant, die in unterschiedlichen DSLs zum Einsatz kommen kann. Diese soll die Konzepte und Regeln für Werttypen zusammenfassend darstellen.

Die Innovation dieser Arbeit liegt auf zwei Ebenen. Einerseits wird der Benutzer einer DSL unterstützt, indem die Ausdruckskraft von DSLs verbessert wird. Fachliche Konzepte, die wertartig modellierbar sind, lassen sich dann mit einer konkreten Syntax in der DSL darstellen. Es muss nicht mehr auf primitive Typen zurückgegriffen werden. Dies ermöglicht auch eine bessere, fachliche Überprüfung von Ausdrücken auf Fehler. Außerdem wird diese Arbeit den DSL-Entwickler in Entwurf und Implementation von textuellen DSLs unterstützen. Sie wird Ansätze liefern, um modellierte Werttypen in die Werkzeuge einer DSL zu integrieren.

Literaturverzeichnis

[Be01] Bézivin, J.: From object composition to model transformation with the MDA. In Proceedings of TOOLS, Vol. 1, S. 350-354, 2001.

[BL+95] Bruce, K.; Leavens, G. T.; Castagna, G.; Cardelli, L.; Pierce, B.: On binary methods. In SYMPOSIUM ON OBJECT-ORIENTED PROGRAMMING: SYSTEMS, LANGUAGES, AND APPLICATIONS, ACM, S. 227-256. Harvard University Press, 1995.

[ES+95] Evered, M.; Schmolitzky, A.; Kölling, M.: A Flexible Object Invocation Language based on Object-Oriented Language Definition, 1995.

[Ev03] Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003.

[Fo03] Fowler, M.: Patterns of Enterprise Application Architecture (The Addison-Wesley Signature Series). Addison Wesley, 1 edition, 2003.

[Fo10] Fowler, M.: Domain-Specific Languages (Addison-Wesley Signature Series). Addison-Wesley Professional, 1 edition, 2010.

[HM+04] Hevner, A. R.; March, S. T.; Park, J.; Ram, S.: Design Science in Information Systems Research. MIS Quarterly, Volume 28, 2004.

[KT08] Kelly, S.; Tolvanen, J.-P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Pr, 3 2008.

[Ma82] MacLennan, B. J.: Values and objects in programming languages. SIGPLAN Not., 17(12): S. 70-79, 1982.

[MH+05] Mernik, M.; Heering, J.; Sloane, A. M.: When and how to develop domain-specific languages. ACM Comput. Surv., 37(4): S. 316-344, December 2005.

[OA+04] Odersky, M; Altherr, P.; Cremet, V.; Emir, B.; Maneth, S.; Micheloud, S.; Mihaylov, N.; Schinz, M.; Stenman, E.; Zenger, M.: An overview of the scala programming language. Technical report, Technical Report IC/2004/64, EPFL Lausanne, Switzerland, 2004.

[Se03] Seidewitz, E.: What models mean. Software, IEEE, 20(5): S. 26-32, Sept.-Oct. 2003.

[SV+07] Stahl, T.; Völter, M.; Efftinge, S.; Haase, A.: Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management Dpunkt Verlag, 2007.

[UML11] UML Infrastructure Specification, Version 2.4.1, August 2011, http://www.omg.org/spec/UML/2.4.1/, zuletzt aufgerufen: 20.11.2012.

[Zü04] Züllighoven, H.: Object-Oriented Construction Handbook: Developing Application-Oriented Software with the Tools & Materials Approach. Elsevier Science, 2004.