Fachbereich Informatik und Medien -
Wissenschaftliches Arbeiten und Schreiben - Master Inf. Prof.
Loose WS 2011/12

Untersuchung von sprachorientierten Programmierparadigmen, deren Ausprägungen und Akzeptanz bei Domänenexperten.

oder

Konzeptionelle Analyse und Umsetzung von ausgewählten Domain Specific Languages unter dem Aspekt der Erweiterbarkeit, Anwendbarkeit und Benutzerakzeptanz.

Einleitung

Vorgelegt von: Nils Petersohn

am: 8.11.2011.

Inhaltsverzeichnis

1 Einleitung

Die Sektionen würden bei der echten Arbeit wegfallen.

1.1 Motivation (Heranführen an das Thema)

Das Wort “Abstraktion” bezeichnet meist den induktiven Denkprozess des Weglassens von Einzelheiten und des Überführens auf etwas Allgemeineres oder Einfacheres [...] also [...] jenen Prozess, der Informationen so weit auf ihre wesentlichen Eigenschaften herab setzt, dass sie psychisch überhaupt weiter verarbeitet werden können. (nach [wika]) Die grundlegenden Abstraktionsstufen in der Informatik sind wie folgt aufgeteilt: Die unterste Ebene ohne Abstraktion ist die der elektronischen Schaltkreise, die elektrische Signale erzeugen, kombinieren und speichern. Darauf aufbauend existiert die Schaltungslogik. Die dritte Abstraktionsschicht ist die der Rechnerarchitektur. Danach kommt eine der obersten Abstraktionsschichten: “Die Sicht des Programmierers”, der den Rechner nur noch als Gerät ansieht, das Daten speichert und Befehle ausführt, dessen technischen Aufbau er aber nicht mehr im Einzelnen zu berücksichtigen braucht. (nach S. 67 [Rec00]). Diese Beschreibung von Abstraktion lässt sich auch auf Programmiersprachen übertragen. Nur wenige programmieren heute direkt Maschinencode, weil die Programmiersprachen der dritten Generation (3GL) soviel Abstraktionsgrad bieten, dass zwar kein bestimmtes Problem aber dessen Lösung beschrieben werden kann. Die Lösung des Problems muss genau in der Sprache beschrieben werden und setzt das Verständnis und die Erfahrung in der Programmiersprache und deren Eigenheiten zur Problemlösung voraus. Das Verständnis des eigentlichen Problems, dass es mit Hilfe von Software zu lösen gilt, liegt nicht immer zu 100Programmierer, der es mit Java oder C# bzw. einer 3GL lösen soll. Komplexe Probleme z.B. in der Medizin, der Architektur oder im Versicherungswesen sind oft so umfangreich, dass die Aufgabe des “Requirements Engeneering” hauptsächlich darin besteht, zwischen dem Auftragnehmer und Auftraggeber eine Verständnisbrücke zu bauen. Diese Brücke ist auf der einen Seite mit Implementierungsdetails belastet und auf der anderen mit domänenspezifischem Wissen (Fach- oder Expertenwissen). Die Kommunikation der beiden Seiten kann langfristig durch eine

DSL begünstigt werden, da eine Abstrahierung des domänenspezifischen Problems angestrebt wird. Die Isolation der eigentlichen Businesslogik und eine intuitiv verständliche Darstellung in textueller Form kann sogar soweit gehen, dass der Domänenexperte die Logik in hohem Maße selbst implementieren kann, weil er nicht mit den Implementierungsdetails derselben und den syntaktischen Gegebenheiten einer turingvollständigen General-Purpose-Language wie Java oder C# abgelenkt wird. Im Idealfall kann er die gewünschten Anforderungen besser abbilden.(vlg. [hei11]). Beispiele für DSL sind z.B.: Musiknoten auf einem Notenblatt, Morsecode oder Schachfigurbewegungen ("Bauer e2-e4") bis hin zu folgendem Satz: "wenn (Kunde Vorzugsstatus hat) und (Bestellung Anzahl ist größer als 1000 oder Bestellung ist neu) dann ... ". "Eine domänenspezifische Sprache ist nichts anderes als eine Programmiersprache für eine Domäne. Sie besteht im Kern aus einem Metamodell einer abstrakten Syntax, Constraints (Statischer Semantik) und einer konkreten Syntax. Eine festgelegte Semantik weist den Sprachelementen eine Bedeutung zu." (vlg. S.30 [mda])

1.2 Begriffserklärung (falls notwendig)

Eine DSL beinhaltet ein Metamodell. In einer Domänengrammatik gibt es das Konzept an sich, das beschrieben werden soll. Konzepte können Daten, Strukturen oder Anweisungen bzw. Verhalten und Bedingungen sein. Das Metamodell oder auch das Semantische Modell besteht aus einem Netzwerk vieler Konzepte. Das Paradigma "Language Orientated Programming" (LOP) identifiziert ein Vorgehen in der Programmierung, bei dem ein Problem nicht mit einer GPL (general purpose language) angegangen wird, sondern bei dem zuerst domänenspezifische Sprachen entworfen werden, um dann durch diese das Problem zu lösen. Zu diesem Paradigma gehört auch die Entwicklung von domänenorientierten Sprachen und intuitive Programmierung (intentional programming). "Intentionale Programmierung ist ein Programmierparadigma. Sie bezeichnet den Ansatz, vom herkömmlichen Quelltext als alleinige Spezifikation eines Programms abzurücken, um die Intentionen des Programmierers durch eine Vielfalt von jeweils geeigneten Spezifikationsmöglichkeiten in besserer Weise auszudrücken." (vlg. [wikb]) Es werden zwei Arten von DSLs unterschieden [FP10]. Eine interne DSL ist eine in die "General Purpose Language" eingebettete "Spracherweiterung", die sich mit den gegebenen Mitteln der Wirtssprache und deren Möglichkeit zur Metaprogrammierung erstellen lässt. Vorzugsweise sind solche Wirtssprachen dynamisch typisiert wie z.B. Ruby, Groovy oder Scala. Eine externe DSL ist völlig eigenständig und besitzt einen eigenen Parser und kann, ohne die "Einschränkungen" einer Wirtssprache, beliebig definiert werden.

1.3 Aufgabenstellung (kurz, knapp, präzise) und Erwartungen

Anhand mehrerer Problemfälle soll das Paradigma der sprachorientierten Programmierung und die damit verbundenen Konzepte der domänenorientierten Programmierung analysiert und geprüft werden. Mehrere vergleichsweise einfache DSLs sollen mit Hilfe von Groovy-Metaprogramming als interne DSLs bzw. mit dem Meta-Programming-System (MPS) sollen überschaubare externe DSLs entworfen werden. Domänenexperten, die keine Erfahrung mit Programmierung haben, sollen an interne bzw. externe DSLs für deren bekannte Domäne herangeführt werden, um diese anschließend nach Lesbarkeit, intuitivem Verständnis und Flexibilität zu bewerten. Dabei bekommen die Probanden mehrere Aufgaben, die sie mit den gegebenen DSLs lösen sollen. Die Erwartungen sind, dass bei kleinen Systemen, der Aufwand zu hoch ist, extra eine DSL einzusetzten. Bei komplexen Systemen stellt eine geeignete DSL jedoch einen unmittelbaren Mehrwert dar.

1.4 Gliederung

Zuerst werden theoretische Grundlagen zum Thema “Sprachorientierte Programmierung” und Metamodellierung erläutert. Der praktische Teil beginnt mit der Vorstellung der Problemdomänen und deren verschiedenen Anforderungen. Dann werden die zu implementierenden DSLs konzeptionell vorgestellt. Die Umsetzung dieser Konzepte mit den Toolsets wird im nächsten Kapitel beschrieben. Weiterhin soll die Testumgebung mit den Probanden spezifiziert werden, um danach die Durchführung und die Ergebnisse auszuwerten und zu beschreiben.

1.5 Abgrenzung

Der Fokus dieser Arbeit soll auf die textuelle und nicht auf die graphische Repräsentation von DSLs abzielen. “Natural language processing” soll nur oberflächlich betrachtet werden. Domänenspezifische Modellierung mit UML soll nur am Rande betrachtet werden. Als Toolset sollen ausschließlich MPS und Groovy-Meta-Programming verwendet werden.

2 Theorie

2.1 Domain Specific Languages

2.1.1 Unterscheidungen

2.2 Sprachorientierte Programmierung

"Language oriented programming is about describing a system through multiple DSLs".(vlg. [Fow05])

Sprachorientierte Programmierung ist nach Fowlers Beschreibung also ein gesamtsystem bestehend aus subsystemen, deren Funktionen mit definierten Sprachmodellen beschrieben werden. Bestenfalls sollten diese Sprachmodelle die fachlichen Probleme auf eine natürliche Art beschreiben können ohne dabei mehr zu beschreiben als das Fachgebiet benötigt.

2.3 Groovy

Groovy ist eine dynamisch typisierte Programmiersprache und Skriptsprache für die Java Virtual Machine. Groovy besitzt einige Fähigkeiten, die in Java nicht vorhanden sind: Closures, native Syntax für Maps, Listen und Reguläre Ausdrücke, ein einfaches Templatesystem, mit dem HTML- und SQL-Code erzeugt werden kann, eine XQuery-ähnliche Syntax zum Ablaufen von Objektbäumen, Operatorüberladung und eine native Darstellung für BigDecimal und BigInteger. (vgl. [unn])

2.4 Metaprogrammierung

Introspection bezeichnet die Fähigkeit, auf Informationen über die Zustäe von Objekte, deren Klassen und Verhalten zur Laufzeit zugreifen zu können. Intercession erlaubt Zustände und Verhalten von Objekten, aber auch deren Klassen zur Laufzeit zu verändern. Intercession setzt meist Introspection voraus.

2.5 Metaprogrammierung in Groovy

Groovy besteht aus einem flexiblen Metaklassenmodell. Im Sinne einer Open Implementation hat der Programmierer nahezu alle Möglich- keiten sein Programm mittels Reflection zur Laufzeit zu verändern und damit an die individuellen Bedürfnisse anzupassen. [**?**] Da Groovy auf der Java VM ausgeführt wird und somit auch den Grundregeln des Javaklassenmodells folgen muss, unterliegt es auch dessen Einschränkungen. In Java ist eine Veränderungen der Klassen zur Laufzeit nicht vorgesehen. Die Java Runtime Environement stellt mit dem java.reflect Package primär eine Mölichkeit zur Introspection bereit. [. . .] Erst Javassist, und Reflex erlauben echtes dynamisches Verhalten, in dem auf Byteco- de Ebene die Klasse verändert wird.

Dazu muss allerdings die Klasse teilweise umständlich entladen und neugeladen werden. Ein Meta Object Protocol auf der Java VM kann somit nur mittels einer zusätzlichen Indirektionsschicht realisiert werden. Dazu werden bestehende Konzepte von Java benutzt und um ein Metaklassenmodell erweitert. Aus dem Java Unterbau ergeben sich folgende Grundsätze: Wie in Java ist in Groovy jede Klasse abgeleitet von Object. Groovy ist in diesem Hinblick aber wesentlich konsequenter, da auf primitive Datentypen bewusst verzichtet wurde, um die Inkonsistenzen bei der Behandlung von pri- mitiven Datentypen und Klassen zu vermeiden. Weiterhin ist jede Klasse in Groovy eine Instanz von Class, womit Class also auch in Groovy die Klasse der Klassen bleibt. [**?**]

getMetaClass setMetaClass Nicht nur eine Klasse hat eine Metaklasse, sondern auch jedes einzelne Groovy Objekt kann eine von der Klasse unabhängige Metaklasse haben (siehe 2.4). Diese instanz- spezifische Metaklasse ist über die beiden Methoden zugänglich.

getProperty setProperty Properties definieren den Zustand eines Objektes und werden in Groovy primär auf Instanz und Klassenebene abgebildet. Jede Groovy Klasse kann durch Überschreiben dieser zwei Methoden dynamische Properties auf Objektebene oder Klassenebene erzeugen.

invokeMethod Das Verhalten eines Objektes ist wiederum eine Ebene höher angesiedelt, spielt sich also auf Klassen- oder Metaklassenebene ab. Normalerweise wird dynamisches Verhal- ten damit auf Metaklassenebene realisiert, sodass diese Methode der GroovyOb-

jects gar nicht erst aufgerufen wird. Nur im Fehlerfall oder mit Hilfe des Tag-Interfaces GroovyInterceptable wird invokeMethod aufgerufen (in 2.5 genau beschrieben).

[...] Während GroovyObject dynamisches Verhalten auf Objekt- und Klassenebene erlaubt, ist das zweite elementare Interface MetaClass die Grund- lage für das sehr ausgewogene Metaklassenmodell in Groovy.

Metaklassen, Klassen und Instanzen Läuft ein Programm ohne Intercession, so gelten für die Metaklassen der Klassen und Instanzen folgen- de Grundaussagen: Jede Klasse hat eine Metaklasse, die eine Instanz von Me- taClassImpl ist und damit MetaClass implementiert. Diese Instanzen werden dynamisch erzeugt und sind bis auf wenige Ausnahmen direkt MetaClassImpl Instanzen. Java Klassen erhalten eine Instanz von ExpandoMetaClass als Meta- klasse, damit auch diesen Klassen Methoden hinzugefügt werden können. Neue Instanzen von Klassen werden über die Metaklasse der Klasse erstellt und haben diese Metaklasse als Metaklasse.

getProperties getMethods getMetaMethods Die Methoden der Introspection in Groovy. Der Unterschied zwischen getMethods und getMetaMethods wird in 2.5 näher erläutert.

getClassNode Liefert den AST der Metaklasse sofern verfügbar. Erlaubt auch die Modifikation dieses ASTs und ist damit sowohl für Introspection als auch Intercession geeignet. Aufgrund der Komplexität des ASTs wird allerdings dynamisches Verhalten häufiger über die ExpandoMetaClass realisiert und der AST verwendet, um Quelltextfragmente neu zu interpretieren. In den Standardbibliotheken wird so zum Beispiel eine Closure auto- matisch in ein SQL Statement umgewandelt, um das SELECT Statement performant zu benutzen.

invokeMethod Jeder Methodenaufruf wird primär von der Metaklasse behandelt und entweder an die invokeMethod Funktion des GroovyObjects oder aber an die entsprechende Methode der Klasse delegiert. Das Erzeugen einer eigenen Metaklasse und überschreiben dieser Methode ist die Hauptmöglichkeit für dynamisches Verhalten in Groovy außer der ExpandoMetaClass.

getProperty setProperty Die Methoden zur Property-Unterstützung auf Metaklassenebene werden von der Standardimplementierung von den entsprechenden Methoden von GroovyObject auf- gerufen. Die Aufrufreihenfolge ist im Vergleich zu invokeMethod also genau anders- herum. Auch diese Methoden sind für Intercession geeignet.

invokeMissingMethod invokeMissingProperty Diese Backupmethoden werden jeweils aufgerufen, wenn die normalen Aufrufmecha- nismen fehlgeschlagen sind. Intercession mit diesen Methoden erlaubt zum Beispiel die Erweiterung von Klassen und Objekten um zusätzliche Properties zur Laufzeit.

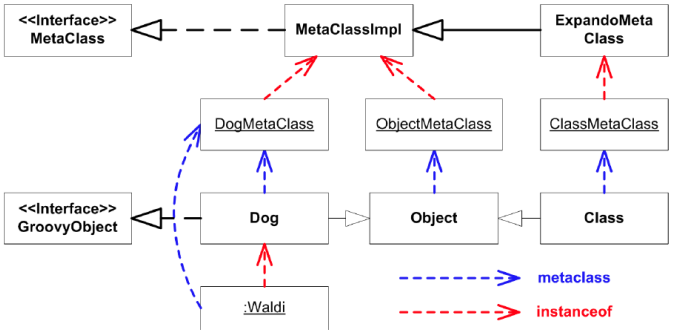

Abbildung 2.1: Beziehung von Metaklassen, Klassen und Instanzen [**?**])

2.5.1 Closures

2.5.2 Kategorien

2.5.3 Expando-MetaClass

3 Praktischer Teil

3.1 Die Fachliche Domäne

4 Auswertung

5 Zusammenfassung und Schlussbetrachtung

Literaturverzeichnis

[Fow05] Fowler, M.: Language workbenches: The killer-app for domain specific languages. In: *Accessed online from: http://www. martinfowler. com/articles/languageWorkbench. html* (2005)

[FP10] Fowler, M. ; Parsons, R.: *Domain-specific languages.* Addison-Wesley Professional, 2010

[hei11] heise: *JetBrains verffentlicht Meta Programming System 2.0.* `http://www.heise.de/developer/meldung/JetBrains-veroeffentlicht-Meta-Programming-System-2-0-1327137.html`. Version: 22.08. 2011

[mda] *modelgetriebene softwareentwicklung (techniken, engeneering, management)*

[Rec00] Rechenberg, P.: *Was ist Informatik?: eine allgemeinverständliche Einführung.* Hanser Verlag, 2000

[unn] `http://de.wikipedia.org/wiki/Groovy`

[wika] *Abstraktion.* `http://de.wikipedia.org/wiki/Abstraktion`

[wikb] *Intentional Programming.* `http://de.wikipedia.org/wiki/Intentional_Programming`