Objekt Algebra - Ein Lösungsansatz für das Expression Problem

Marco Buchholz, Max Golubew und Florian Winzek

Institute for Software Engineering and Programming Languages, Universität zu Lübeck
marco.buchholz@student.uni-luebeck.de
max.golubew@student.uni-luebeck.de
f.winzek@student.uni-luebeck.de

Zusammenfassung. Das Expression Problem ist ein bekanntes Problem in der Funktionalen Programmierung. Hierbei geht es darum, dass ein Programm, welches aus Datenstrukturen und Funktionen besteht, in diese zwei Dimensionen weiterentwickelt werden soll, ohne das Programm neu kompilieren zu müssen. Die Komplexität des Programs darf ebenfalls nicht größer werden. In diesem Paper behandeln wir einen Ansatz, welcher das Problem lösen kann. In diesem Zusammenhang stellen wir das Konzept der Objekt Algebra. Wir werden Lineare Zeit Logik Formeln (LTL-Formeln) als anschauliches Beispiel benutzen. Um zu zeigen, dass die Objekt Algebra ein Programm auf einfache Art um Operationen und Datenstrukturen weiterentwickeln kann, werden wir zunächst eine Grundmenge von LTL-Formeln betrachten und diese dann um zusätzliche Formeln und Funktionen erweitern.

1 Einleitung

Beim Expression Problem geht es speziell um Computerprogramme oder Algorithmen. Diese werden im Laufe der Entwicklung immer komplexer. Das bedeutet, es kommen neue Funktionen für bestehende Datentypen hinzu, aber auch neue Datentypen für aktuelle Funktionen. Diese beiden Dimensionen sollten beim Entwicklungsprozess möglichst dynamisch gehalten werden, damit das Programm nicht zu komplex wird. Oftmals muss der Code neu geschrieben und kompiliert werden.

Es gibt einige Ansätze um zumindest einen der zwei Dimensionen dynamisch weiterzuentwickeln. Wir werden im folgenden Kapitel auf diese Ansätze eingehen und die Vor- und Nachteile erörtern. Im Anschluss stellen wir einen alternativen Ansatz vor, welcher beide Dimensionen gleichzeitig abdeckt. Zum Schluss zeigen wir eine mögliche Implementierung am Beispiel von LTL-Formeln.

2 Lösungsansätze

In diesem Kapitel wollen wir zunächst ein wenig auf bekannte Software Pattern eingehen, die sich im Kern mit dem Expression Problem auseinandersetzen. Dies

sind zum Einen das sogenannte *Interpreter Pattern* und das *Visitor Pattern*. Ein alternativer Ansatz stellt die sogennante *Objekt Algebra* dar, auf welchen wir insbesondere eingehen werden.

2.1 Interpreter Pattern

2.2 Visitor Pattern

2.3 Objekt Algebra

3 Implementation

In diesem Abschnitt geht es um eine konkrete Implementierung der in subsection 2.3 beschriebenen Objekt Algebra. Dabei haben wir die *Linear Temporal Logik (LTL)* [3] verwendet, um die dort definierten Datentypen und Funktionen nachzubilden. Genauer gesagt muss eine Datenstruktur für den *abstract syntax tree (AST)* für LTL definiert werden. Eine LTL-Formel ist durch die Grammatik in 3 gezeigt.

```
LTL ::= PROPOSITION | "!" LTL | LTL "||" LTL | LTL "&&" LTL
        | LTL "U" LTL | "X" LTL | "F" LTL
PROPOSITION ::= "TRUE" | "FALSE" | VARIABLE
VARIABLE ::= [a-z];
```

Daraus ergibt dich, dass der benötigte AST aus den Datenvarianten *Disjunction*, *Conjunction*, *Unary*, *Next*, *Finally*, *Negation*, *Proposition* und *Variable* besteht. Um die Erweiterbarkeit der verwendeten Objekt Algebra zu zeigen, haben wir die Implementierung in drei evolutionäre Stufen aufgeteilt. Zuerst gibt es nur die Operation *Evaluation* zum Auswerten einer LTL-Formel auf einem gegebenen Wort. Außerdem ist die Datenvariante *Finally* noch nicht implementiert. Diese kommt dann in der zweiten Evolution dazu und in der letzten gibt es eine neue Operation mit dem Namen *PrettyPrint*. Diese liefert eine String-Repräsentation des AST, welche z.B. auf der dem Benutzer ausgegeben werden kann.

Anhand des Klassendiagramms in Figure 1 zeigen wir die Implementierung der Objekt Algebra für LTL-Formeln.

3.1 Evaluation - Evolution 1

Die generische Schnittstelle *ILtLBase* stellt das sogenannte *Object Algerba Interface* dar (vgl. [6, Abschnitt 3 S.6]). Dieses bietet die Definitionen der Datenvarianten an, wobei darauf zu achten ist, dass es keine Methode für *Finally* gibt. Außerdem ist es durch die generische Typ-Definition unabhängig von einer Operation und trotzdem ist eine statische Typ-Sicherheit gegeben. Die Klasse *Evaluation* stellt nun die konkrete *Objekt Algebra* dar.

Dafür wird die Schnittstelle *ILtlBase* implementiert und dessen generischer Parameter an die Schnittstelle *IEval* gebunden. *IEval* stellt unsere Operation dar

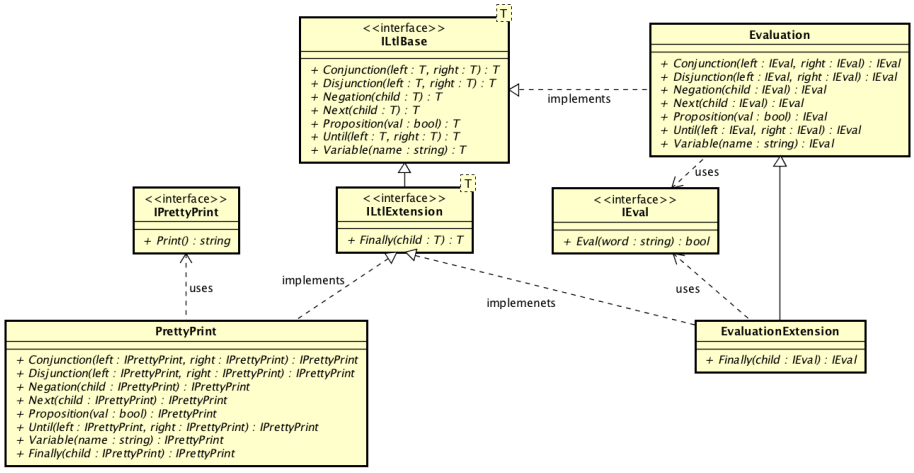

Abb. 1. Ausschnitt des Klassendiagramms. Gezeigt sind die Klassen für die Objekt Algebra

und definiert daher die möglichen Methoden, die auf dem AST ausgeführt werden können. Durch die Bindung des generischen Parameters auf die Operation wird nun von jeder Methode in *Evaluation* eine konkrete Implementierung von *IEval* zurückgeliefert. Das bedeutet die Objekt Algebra arbeitet vergleichbar zu einer *Factory*, da sie neue Typen des Interfaces (anonym) erstellt und zugleich dessen Instanzen erzeugt.

3.2 Finally - Evolution 2

Nun kommt die neue Datenvariante *Finally* dazu. Dafür kann einfach eine neue Schnittelle erstellt werden, welche das vorhandene *Object Algebra Interface* erweitert. *ILtlExtension* realisiert genau diese neue Schnittstelle. Dabei ist darauf zu achten, dass auch diese ebenfalls generisch sein muss und ihren Template Parameter an die generalisierte Schnittstelle *ILtlBase* übergibt und nicht durch eine konkrete Realisierung ersetzt.

Nun kann auch dafür eine *Objekt Algebra* erstellt werden. Dafür muss eine Klasse nur das neue *Object Algebra Interface* implementieren. In unserer Implementation ist dies die Klasse *EvaluationExtension*. Um die Methoden aus der ersten Evolution nicht erneut implementieren zu müssen, wird *Evaluation* erweitert.

3.3 PrettyPrint - Evolution 3

In der letzten Stufe wollen wir zeigen, wie eine neue Operation (*PrettyPrint*) hinzugefügt werden kann. Dafür wird zuerst eine neue Schnittstelle *IPrettyPrint* erstellt. Jetzt wird nur noch eine entsprechende *Object Algebra* benötigt. Die

Klasse *PrettyPrint* ist genau diese konkrete Realisierung. Da sich in diesem Fall keine Datenvariante geändert hat, kann das vorhandene *Object Algebra Interface ILtlExtension* implementiert werden. Die Klasse *PrettyPrint* dient wieder als Factory und erzeugt somit Instanzen, welche unsere Schnittstelle *IPrettyPrint* implementieren. Das ist auch der Grund, warum **nicht** die Klasse *EvaluationExtension* erweitert werden kann.

4 Zusammenfassung

Literatur

1. Wadler, P.: The Expression Problem. E-Mail Discussion (1998), `http://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt`
2. Odersky, M., Zenger, M.: Independently Extensible Solutions to the Expression Problem. In FOOL'05
3. Pnueli, A.: The temporal logic of programs. In 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, pages 46-57, IEEE Computer Society, 1977
4. Gamma, E., Helm R., Johnson R. and Vlissides J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series, Pearson Education, 1994
5. Parr, T.: Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages. Pragmatic Bookshelf, 2009
6. Oliveira, B., Cook, W.: Extensibility for the Masses - Practical Extensibility with Object Algebras. In ECOOP 2012 – Object-Oriented Programming: 26th European Conference, Beijing, China, June 11-16, 2012, pages 2-27, Springer Berlin Heidelberg
7. Guttag, J., Horning, J.: The algebraic specification of abstract data types. In Acta Informatica Vol. 10, pages 27-52, March 1978