Implementierung eines Prolog-Adapters für Java

Praktikum von

Johannes Werner

Institut für Software Design und Qualität
Betreuender Mitarbeiter: M.Sc. Stephan Seifermann

1 Einführung

Programmiersprachen haben unterschiedliche Stärken und Schwächen. Das liegt daran, dass Sprachen meist für ein bestimmtes Problem oder eine bestimmte Domäne an Problemen entwickelt werden. Schaut man sich Java und Prolog an, wird dies besonders deutlich. Java ist eine objekt-orientierte Programmiersprache mit dem Ziel, möglichst zugänglich und gleichzeitig möglichst mächtig zu sein. Durch den imperativen Programmierstil ist diese Sprache außerdem für viele Menschen gut verständlich und kann für eine Vielzahl an Problemen eingesetzt werden. Prolog hingegen verfolgt einen ganz anderen Ansatz. Anstatt eine Sequenz an Befehlen zu programmieren, die angeben, *wie* ein Problem gelöst wird, definert man in Prolog lediglich *was* das Problem ist und der Prolog-Interpreter löst es dann. Das ist vorallem dann von Vorteil, wenn man vorrangig logische Probleme lösen will.

Um die Vorteile einer Sprache auch sprachübergreifend nutzen zu können, werden häufig Bibliotheken bereitgestellt, die als Schnittstelle zu anderen Sprachen dienen. Beispielsweise stellen einige Prolog-Interpreter Java-Bibliotheken bereit, die einen einfachen Zugang zu Prolog in Java-Code ermöglichen. Allerdings haben diese Bibliotheken keine einheitliche API, was es erschwert, verschiedene Implementierungen derselben Sprach in einem Programm zu nutzen. Dabei ist eine Auswahl verschiedener Implentierungen genau das, was in manchen Fällen erforderlich sein kann. Gründe dafür können sich per Einsatzszenario ändernde Anforderungen sein, wie beispielsweise Lizenzbedingungen oder Performanceunterschiede. Um dennoch mehrere verschiedene Implemntierungen verwenden zu können, kapselt man die unterschiedlichen APIs in einem Adapter und stellt nach außen hin eine einheitliche API zur Verfügung. Abbildung 1 zeigt, wie ein solcher Adapter shematisch aussieht. Schnittstellen zu einer Zielsprache werden entweder als eine native Bibliothek zur Verfügung gestellt, oder die Zielsprach ist direkt in

der Quellsprache implementiert. Der Adapter sellbst ist in der Quellsprache implementiert und kapselt die unterschiedlichen Schnittstellen-APIs. Dadurch kann das Hauptptogramm einheitlich auf die verschiedenen Implementierungen der Zielsprache zugreifen.

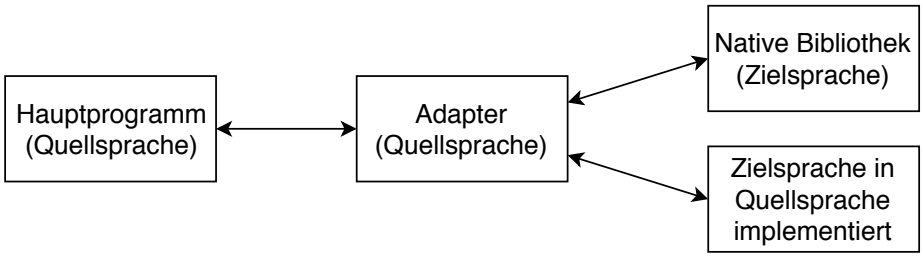

Abbildung 1: Ein Adapter ist in der Quellsprache implementiert und greift entweder auf eine native Bibliothek oder auf eine Quellsprach-Implementierung der Zielsprache zu.

1.1 Projektrahmen

Dieses Praktikum findet im Rahmen des Projektes Trust 4.0[1] statt. Trust 4.0 hat es sich als Ziel gesetzt, Datenschutz in einem Softwaresystem garantieren zu können. Dazu wird mittels eines Plugins in Eclipse ein Architekturmodell des zu prüfenden Softwaresystems erstellt und um Datenflussannotationen ergänzt. Diese Annotation beschreiben, welchen Weg personenbezogne Daten über verschiedene Software-Module hinweg nehmen und aufgrunddessen kann man analysieren, ob es an einer Stelle zu Datenlecks kommen kann. Das Eclipse-Plugin selbst ist in Java implementiert, jedoch hat man sich bei der Datenflussanalyse für Prolog entschieden. Wie bereist weiter oben erläutert, ist Prolog speziell für rein logischen Probleme entwickelt worden, weshalb es sich für die Analyse besser eignet als Java. Um es in Zukunft dem Nutzer zu ermöglichen, von dem Eclipse-Plugin aus die Analyse auf einem frei gewählten Prolog-Interpreter auszuführen, ist es das Ziel dieses Praktikums, einen Prolog-Adapter für Java zu entwickeln.

2 Anforderungen

Der Adapter soll im Projekt Trust 4.0 (Citation) eingesetzt werden. Daher gibt es einige Anforderungen. Zum einen müssen die angebunden Interpreten eine EPL kompatible Lizen haben (Citation). Da das Plugin, welches den Adapter einsetzt, unter EPL lizenziert sind, muss Software, die verwendet wird, auch EPL-kompatibel lizenziert sein. Außerdem soll mindestens ein Interpreter standardmäßig mit ausgeliefert werden. Das heißt es muss mindestens ein Interpreter in Java implementiert sein. Er muss in Java implementiert sein, dass er zusammen mit den anderen Jar-Dateien gepackaged werden kann. Der Grund dass mindestens einer mit ausgeliefert werden soll, ist, dass das Plugin out-of-the-box funktionieren soll und der user sich nur bei Bedarf andere Interpreter installieren muss. Daher sind auch keine grßen Anforderdungen bezüglich Speicherverbauch oder Performance

an den Interpreter zu stellen. Es sollen allerdings mindestens zwei Interpreter angebunden werden, da Auswahl sowieso immer gut ist und man so eben auch einen Interpreter w"hlen kann, der von der Performance her besser ist (Keine Ahnung). Eine weiter Anforderung ist, dass die Antworten vom Interpreter in Java Objekte transformiert werden und nicht als plain text zurück kommen. Vom Interface her ist es außerdem erforderlich, dass eine Prologdatenbank aus einer Datei geladen werden kann.

3 Interpreter

Bei der Wahl der Interpreter, die angebunden werden, gibt es verschiedene Kriterien, die zu beachten sind. **Java-Schnittstelle** Es ist erforderlich, dass der Interpreter eine eigene Java-Schnittstelle mitbringt, von der aus abstrahiert werden kann. Andernfalls wäre es im Rahmen dieses Praktikums zu aufwändig eine komplett eigene Schnittstelle zu entwickeln. **Passende Lizenz** Wie bereits im vorherigen Kapitel erwähnt, muss der Interpreter eine zu EPL kompatible Lizenz haben, um später auch verwendet werden zu können **Paket im Maven Repository** Diese Anforderung ist nicht erforderlich, allerdings erleichtert es das Bauen des Adapters erheblich, wenn ein Paket einfach aus dem Maven-Repository geladen werden kann, anstatt dass die Libraries selber verwaltet werden müssen. **Java-Implementierung** Diese Anforderung gilt nur für mindestens einen Interpreten. Allerdings ist es nicht schlecht, wenn auch mehr Interpreter in Java implementiert sind. **Projektaktivität** Es ist auch wichtig, ob die Interpreter aktiv gewartet werden, oder ob der Zeitpunkt des letzten Releases mehere Jahre zurück liegt

3.1 SWI-Prolog

Dieser Interpreter ist Quasi-Standard in der Prolog-Welt

3.2 TuProlog

3.3 Projog

4 Prolog4J

5 Fazit

6 Aussicht

Literatur

[1] Robert Heinrich Stephan Seifermann und Ralf Reussner. “Data-Driven Software Architecturefor Analyzing Confidentiality”. In: (2019). URL: https://sdqweb.ipd.kit.edu/publications/pdfs/seifermann2019b.pdf.