# Implementierung eines Prolog-Adapters für Java

**Praktikum von**

Johannes Werner

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

## 1 Einführung

Jede Programmiersprache wurde mit einem bestimmten Motto entwickelt. Dadurch zeichnet sich auch jede Programmiersprach durch eigene Stärken aus. So ist zum Beispiel C eine hervorragende Sprache, wenn es darum geht, hardwarenah und effizient zu programmieren. Verliert aber schnell an Macht, wenn es darum geht umfangreiche Business Software. Java hingegen eignet sich hervorragend für solche Software, braucht allerdings eine JVM, was Java weitesgehendst untauglich macht, wenn Speicher und Rechenleistung knapp sind. Beispiele finden, wo die Kombination Sinn macht.

Aus diesem Grund kann es wünschenswert sein, mehrere Sprachen im selben Projekt zu verwenden. Zum einen ist dies möglich in der Form C und Assembler, wo durch spezielle Instruktionen Assemblercode direkt in C-Code eingebettet werden kann. Allerdings gibt es für andere Sprachen keine vorgesehenen Schnittstellen. Dann wird es erforderlich, einen Adapter für die jeweilige Sprache zu benutzen. Diese ermöglicht es, Methoden oder Befehle einer Sprache aus dem Code aus einer anderen Sprache heraus aufzurufen. In meinem Praktikum habe ich einen solchen Adapter für das Ausf"hren von Prologanfragen in Java implementiert.

## 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