

Prologadapter für Java

Betreut von Stephan Seifermann

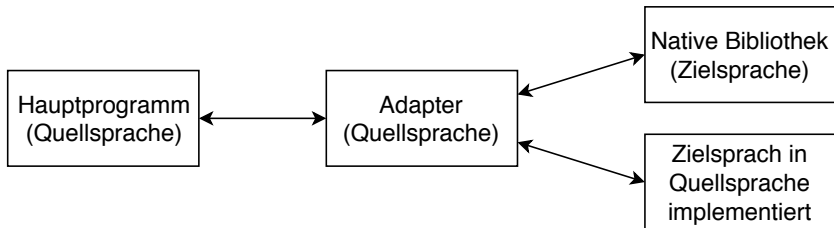
Johannes Werner | 21. Juli 2019

FAKULTÄT FÜR INFORMATIK



- 1 Motivation
- 2 Anforderungen
- 3 Interpreterwahl
- 4 Java-Schnittstelle
- 5 Prolog4J
 - Architektur
 - Anpassungen

- Sprachen haben unterschiedliche Stärken
 - z.B. Haskell vs. C
- Wie kann man diese Stärken kombinieren?
- Wie kann man mehrere Sprachen im selben Programm nutzen?
- **Lösung:** Adapter einsetzen



- Softwaresystem soll Datenschutz garantieren
 - Architekturmodell wird um Datenflussannotationen ergänzt
 - Durch Analyse können Risiken gefunden werden
-
- Architekturmodell
 - Eclipse-Plugin in **Java**
 - Datenflussanalyse
 - Datenflussmodell wird zu **Prolog**-Statements transformiert und ausgewertet
 - ⇒ **Java-zu-Prolog Adapter**

- Anbindung zu mehreren Prolog-Interpreter
 - Mndst. 1 in Java implementierter Interpreter
- Interpreter muss EPL kompatible Lizenz haben
- Java-Objekte als Antwort vom Interpreter
 - Anfrage: `String('2+2')`
 - Antwort: `Integer(4)`

■ Kriterien (* Pflicht)

- Java-Anbindung*
- Passende Lizenz*
- Paket im Maven-Repository
- In Java implementiert
- Zeitpunkt des letzten Releases

■ Herausforderung

- Prolog Community ist klein, weshalb es nicht viele aktive Projekte gibt

	SWI-Prolog	TuProlog	Prolog
Lizenz	BSD	LGPL	Apache 2.0
Letzter Release	Juli'19	Okt'18	Dez'18
Maven-Paket	✓	✓	✓
In Java implementiert	×	✓	✓

- SWI-Prolog ist der weitestverbreitetste und am aktivsten weiterentwickelte Interpreter

Beispiel Prologdatenbank:

```
child_of(Hans, Peter).  
child_of(Britgitte, Peter).
```

Ziel: Prolog-Statement in Java ausführen

```
child_of(Norbert, Peter).
```



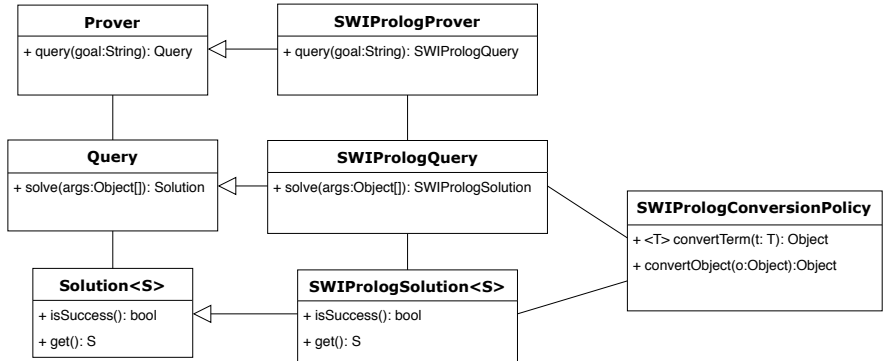
```
class Human {  
    private String name;  
    // Getter and Setter are present  
}  
  
public boolean isChildOf(Human parent, Human childToCheck) {  
    // How to query the prolog database?  
    // The query to execute:  
    //     child_of(nameOfParent, nameOfChildToCheck)  
}
```

```
public boolean isChildOf(Human parent, Human childToCheck) {  
    Atom parentName = convert(parent);  
    Atom childName = convert(childToCheck);  
    // goal = child_of(parentName, childName)  
    Term goal =  
        Util.textParamsToTerm(  
            "child_of(?parent, ?child).",  
            parentName, childName);  
    Query query = new Query(goal);  
    return query.hasMoreSolutions();  
}
```

```
private Projog engine;  
  
public boolean isChildOf(Human parent, Human childToCheck) {  
    String pName = parent.getName();  
    String cName = childToCheck.getName();  
    QueryResult solution =  
        engine.solve("child_of("+pName+", "+cName+").");  
    return solution.next();  
}
```

- Java-zu-Prolog Adapter
- MIT Lizenz (EPL kompatibel)
- Unterstützte Interpreter
 - SWI-Prolog
 - TuProlog
 - JTrolog
 - JLog
- Letzter Commit: **Januar 2011**

```
private Prover prover;  
  
public boolean isChildOf(Human parent, Human childToCheck) {  
    Query query = p.query("child_of(?parent, ?child).");  
    Solution<?> solution =  
        query.solve(parent, childToCheck);  
    return solution.isSuccess();  
}
```



- Entfernen ungewollter Interpreter
- Aktualisieren der POM-Dateien
 - Maven als Build-System
- Aktualisieren des Codes
- Projog Interpreter hinzufügen
- API um nötige Funktionalität erweitern
- Interpreter-Discovery mit OSGI

- Darstellung einer leeren Liste in Standard Prolog

```
'.'([], [])
```

- SWI-Prolog Darstellung

```
'[]'([], [])
```

- Begründung

As of version 7, SWI-Prolog lists can be distinguished unambiguously at runtime from `./2` terms and the atom `'[]'`

- Neu hinzugefügte Regeln für einen Funktor überschreiben vorherige Regeln
 - Regeln werden in einer Hashmap gespeichert und überschreiben beim Einfügen den vorherigen Wert des Funktors
- Folgende Regeln sollen der Datenbank hinzugefügt werden
`human(socrates)`, `human(euklid)`, `human(plato)`

- Folgender Code funktioniert

```
addTheory(human(socrates), human(euklid)), human(plato)).
```

- Resultierende Datenbank

```
human(socrates)
```

```
human(euklid)
```

```
human(plato)
```

- Folgender Code funktioniert **nicht**

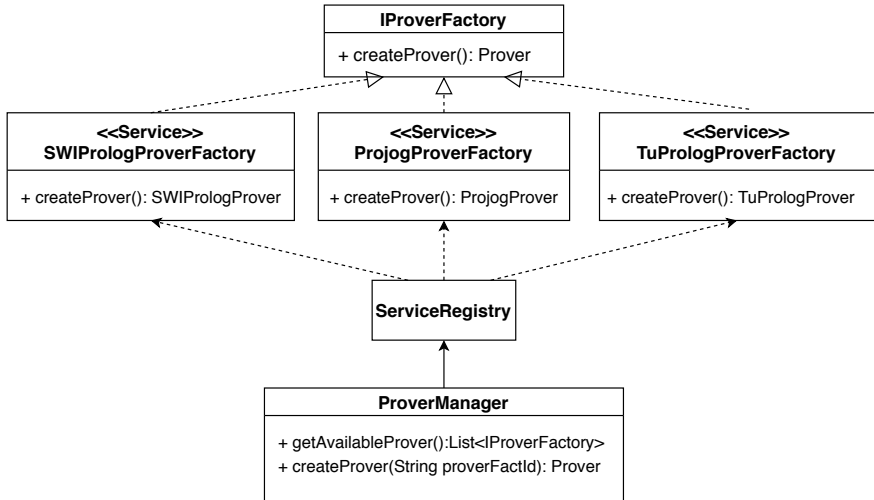
```
addTheory(human(socrates)).  
addTheory(human(euklid)).  
addTheory(human(plato)).
```

- Resultierende Datenbank

```
human(plato)
```

- Eine Liste mit allen Regeln per Funktor wird parallel gehalten
- Einfügen einer Regel:
 - 1 Einfügen der Regel in die eigene Datenbank
 - 2 Einfügen der gesamten Datenbank in die Projog-Datenbank

- **Open Services Gateway initiative**
 - Einbinden modularer Services zur Laufzeit
- Ermöglicht Laufzeiterkennung der verfügbaren Interpreter
 - Kein festes Einprogrammieren nötig



- Ausführen von Prolog-Befehlen in Java
- Adapter soll an mehrere Interpreter anbinden
 - SWI-Prolog
 - TuProlog
 - Prolog
- Java-Schnittstellen der Interpreter sind unterschiedlich
 - Gutes Design des Adapters erforderlich
- Existierende Bibliothek Prolog4J bietet was wir brauchen
 - Muss modernisiert werden
 - Hinzufügen von OSGI Discovery