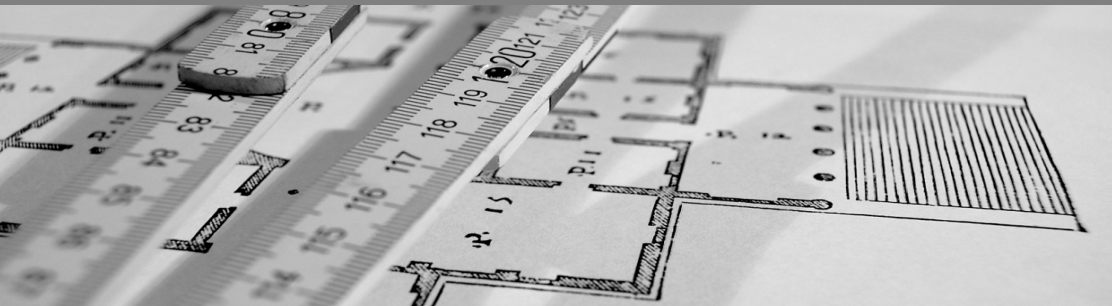


Prologadapter für Java

Betreut von Stephan Seifermann

Johannes Werner | 23. Juli 2019

FAKULTÄT FÜR INFORMATIK



Johannes Werner

Motivation

1 Motivation

Anforderungen

2 Anforderungen

Interpreterwahl

3 Interpreterwahl

Java-Schnittstelle

4 Java-Schnittstelle

Prolog4J

5 Prolog4J

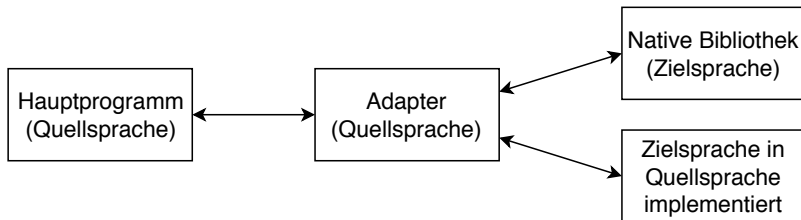
- Architektur
- Analyse

Analyse

Zusammenfassung

6 Zusammenfassung

- Sprachen haben unterschiedliche Stärken
- Stärken kombinierbar?
- Implementierungen einer Sprache haben unterschiedliche Stärken
- Mehrere Implementierungen nutzbar?
- **Lösung:** Adapter einsetzen



- **Kontext:** Adapter soll in Eclipse-Plugin verwendet werden
- Anbindung zu mehreren Interpretern
 - SWI-Prolog unterstützen
 - Mindesten ein Interpreter in Java implementiert
- Interpreter müssen 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
- Java-Implementierung
- Zeitpunkt des letzten Releases

■ Herausforderung

- Es gibt eine Vielzahl an Interpretern
- Wenige erfüllen Kriterien

Johannes Werner

Motivation

Anforderungen

Interpreterwahl

Java-Schnittstelle

Prolog4J

Architektur

Analyse

Zusammenfassung

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

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

Johannes Werner

Motivation

Anforderungen

Interpreterwahl

Java-Schnittstelle

Prolog4J

Architektur

Analyse

Zusammenfassung

Beispiel Prologdatenbank:

```
child_of(Hans, Peter).  
child_of(Britigite, 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) {  
        // How to query the prolog database?  
        // The query to execute:  
        //     child_of(nameOfParent, this.name).  
    }  
}
```



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

```
// class Human { ...
```

```
private Projog engine;
```

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

Johannes Werner

Motivation

Anforderungen

Interpreterwahl

Java-Schnittstelle

Prolog4J

Architektur

Analyse

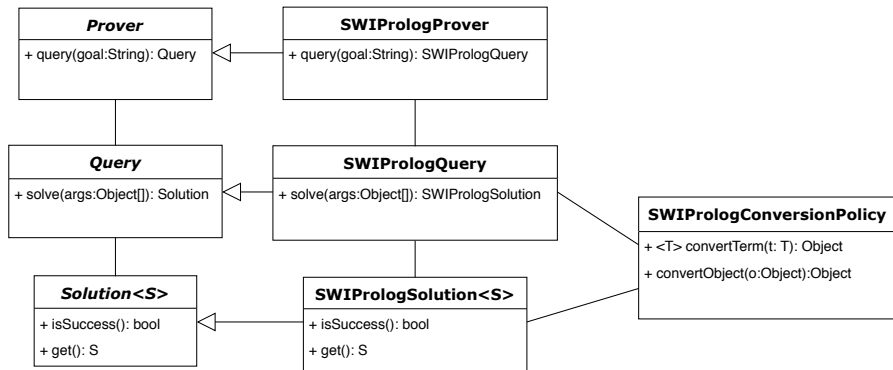
Zusammenfassung

- Java-zu-Prolog Adapter
- MIT-Lizenz (EPL-kompatibel)
- Unterstützte Interpreter
 - SWI-Prolog
 - TuProlog
 - JTrolog
 - JLog
- **Probleme**
 - Veraltet (Letzter Commit in 2011)
 - Fehlende Funktionaliät
 - Unterstützt nicht alle gewünschten Interpreter

```
// class Human { ...
```

```
private Prover p;
```

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



- Unterstützt nicht alle Interpreter
 - Entfernen nicht benötigter Interpreter
 - Projog unterstützen
- Veraltet
 - Aktualisieren der POM-Dateien (Maven)
 - Aktualisieren des Codes
- Fehlende Funktionalität
 - API erweitern
 - Interpreter-Discovery mit OSGI

- 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)`

Johannes Werner

Motivation

Anforderungen

Interpreterwahl

Java-Schnittstelle

Prolog4J

Architektur

Analyse

Zusammenfassung

- 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)
```

Johannes Werner

Motivation

Anforderungen

Interpreterwahl

Java-Schnittstelle

Prolog4J

Architektur

Analyse

Zusammenfassung

- 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

Johannes Werner

Motivation

Anforderungen

Interpreterwahl

Java-Schnittstelle

Prolog4J

Architektur

Analyse

Zusammenfassung

- **Open Services Gateway initiative**
 - Einbinden modularer Services zur Laufzeit
- Einbindung in Laufzeitumgebung von Eclipse
- Laufzeiterkennung verfügbarer Interpreter
 - Kein festes Einprogrammieren nötig
 - Leichtes Einbinden zukünftiger Interpreter

Johannes Werner

Motivation

Anforderungen

Interpreterwahl

Java-Schnittstelle

Prolog4J

Architektur

Analyse

Zusammenfassung

- Ausführen von Prolog-Befehlen in Java
- Adapter soll an mehrere Interpreter anbinden
 - SWI-Prolog
 - TuProlog
 - Projog
- Java-Schnittstellen der Interpreter sind unterschiedlich
- Existierende Bibliothek Prolog4J bietet was wir brauchen
 - Muss modernisiert und erweitert werden
 - Hinzufügen von OSGI Discovery
- Mögliche Fortsetzung
 - Mehr Interpreter anbinden
 - Ausführlicher testen

Johannes Werner

Motivation

Anforderungen

Interpreterwahl

Java-Schnittstelle

Prolog4J

Architektur

Analyse

Zusammenfassung

■ Trust 4.0

- Softwaresystem soll Datenschutz garantieren
- Architekturmodell wird um Datenflussannotationen ergänzt
- Durch Analyse können Risiken gefunden werden

■ Architekturmodell erstellen mit Eclipse-Plugin (**Java**)

■ Datenflussanalyse mittels **Prolog**

⇒ **Java-zu-Prolog Adapter**

- Darstellung einer 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 `' [] '`

