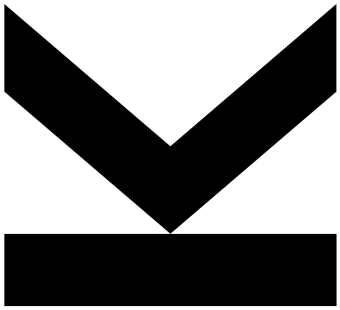
JOHANNES KEPLER
UNIVERSITÄT LINZ

KONZEPTUELLER ENTWURF



258321 DKE Projekt

Gruppe 2

Teammitglieder:
k01607605, Aistleithner Andrea
k01256561, Dusanic Maja
k01356577, Teuchtmann Alexander
k01356229, Tomic Milos

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at

Inhaltsverzeichnis

1. Systemkomponenten

Das System des Evaluierungsframeworks wird in Abbildung 1 graphisch dargestellt. Die Komponenten im grün umrahmten Bereich, sind die zu implementierenden Komponenten.

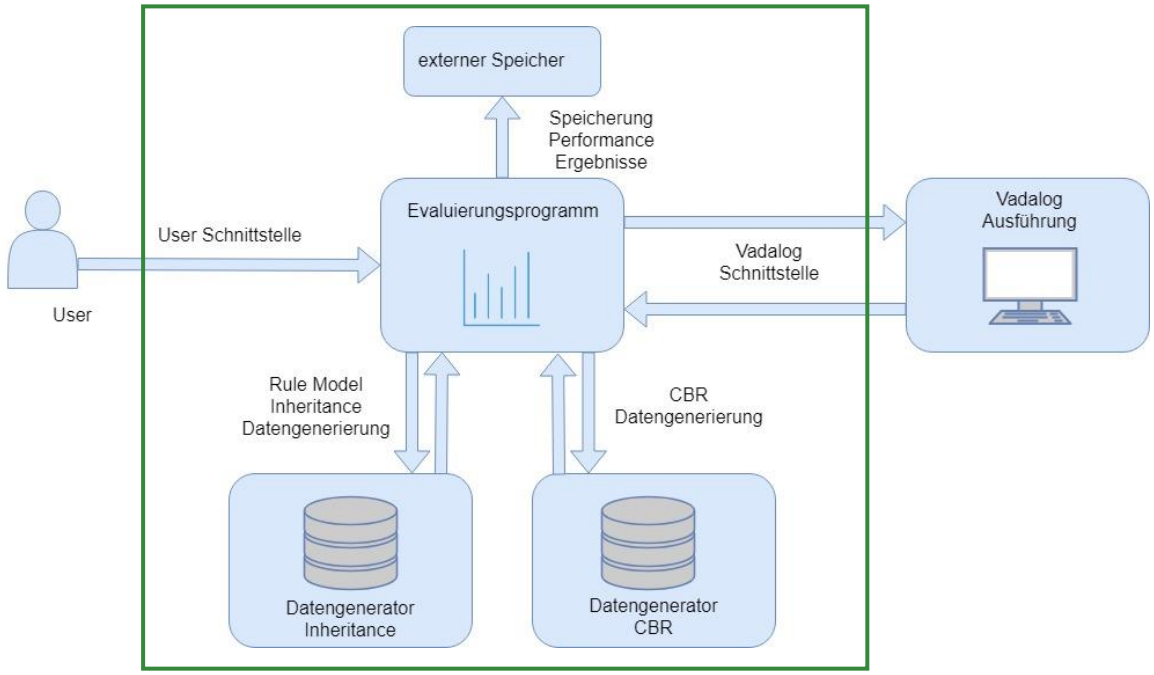


Abbildung 1: Systemkomponenten

Im Folgenden werden die in der Abbildung 1 dargestellten Komponenten genauer beschrieben. Ebenso wird die geplante Implementierung der Komponenten erläutert und schriftlich dargestellt.
Um diese Systemarchitektur umsetzen zu können, werden Schnittstellen benötigt. Eine Übersicht der Schnittstellen folgt nachfolgend. #

2. Schnittstellen

Um es dem Programm zu ermöglichen, mit den einzelnen Komponenten zu kommunizieren, sind folgende Schnittstellen notwendig und zu implementieren. Die benötigten Schnittstellen sind in der folgenden Grafik, in Abbildung 2, orange-rot umrahmt.

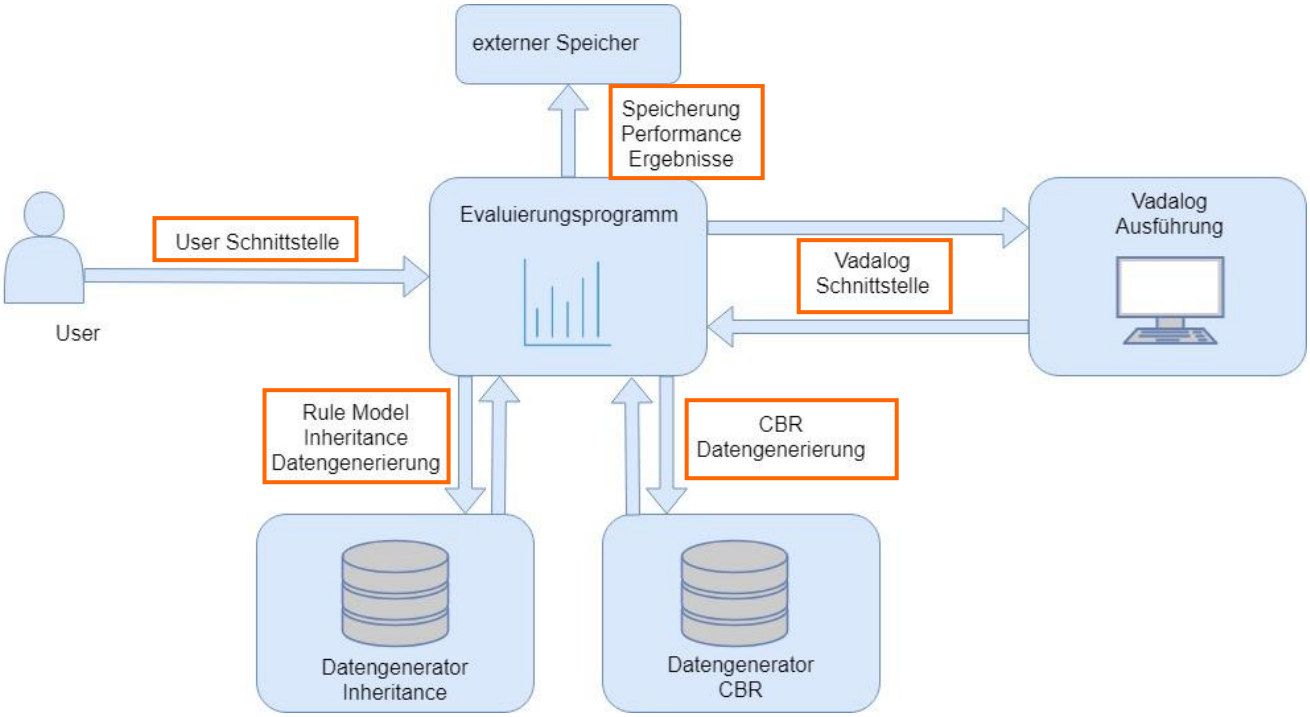


Abbildung 2: Schnittstellen

Um eine funktionierende Kommunikation zwischen den Systemkomponenten zu garantieren, werden die in Abbildung 2 aufgezeigten Systemschnittstellen implementiert. Genauer beschrieben werden die Implementierungen der notwendigen Schnittstellen im nachfolgenden Abschnitt.

3. Umsetzung

Die Umsetzung der Anforderungen erfolgt mit der Programmiersprache Java JDK 8 (oder durch eine neuere Version). Zur Unterstützung der Umsetzung der Datengeneratoren und des Evaluierungsframeworks werden keine Frameworks benutzt, da keine für die Aufgabenstellung passende Frameworks bei der Recherche gefunden wurden.
Um die Umsetzung zu unterstützen werden passende Java Bibliotheken importiert und verwendet.
Die Datenspeicherung wird mithilfe einer mySQL Datenbank ermöglicht und die Verbindung zwischen dem Evaluierungsprogramm wird mithilfe von JDBC umgesetzt, um die zu speichernden Daten eine externe Datenbank übermittelt.

Die geplante Implementierung der vorhergehend beschrieben Systemkomponenten und den dazugehörigen Schnittstellen werden in den folgenden Abschnitten beschrieben. Die Reihenfolge der Beschreibung orientiert sich am Ablauf der Ausführung des Programmes.

3.1. User Schnittstelle

Die User Schnittstelle ist die Verbindung zwischen dem Benutzer des Programmes und des Evaluierungsprogramm, dem Mittelpunkt des Systems. Die Eingabe vom Benutzer erfolgt über die Konsole, wobei die grafische Darstellung vom User Interface nicht zum Ziel gehört.
Der User wählt aus, welche Art von Testfälle er durchführen möchte.
Die Kommunikation zwischen User und Programm erfolgt über die Konsole. Die Konsole gibt dem User die Auswahl, welche Testfälle dieser durchführen möchte:

1 CBR
2 Rule Model Inheritance

Der User gibt die Zahl, als Integer Wert, ein, welche Art von Testfälle dieser durchführen möchte.
Basierend auf der Auswahl des Nutzers, verläuft das Programm wie in den folgenden zwei Abschnitten beschrieben.
Die vom User getroffene Auswahl über CBR oder Rule Model wird an das Evaluierungsprogramm übergeben.

3.1.1. CBR

Hat der User als gewünschte Art der Testfälle 1 CBR gewählt, wird dieser erneut nach einer Auswahl gefragt, um die Anzahl der Parameter, Parameter Values, Contexte und Business Cases festzulegen.

Parameter: (Integer Eingabe User)
Parameter Values: (Integer Eingabe User)
Contexte: (Integer Eingabe User)
Business Cases: (Integer Eingabe User)

Die Eingabe wird von der User Schnittstelle an das Evaluierungsprogramm übergeben.

3.1.2. Rule Model Inheritance

Hat der User als gewünschte Art der Testfälle 2 Rule Model Inheritance gewählt, wird dieser erneut nach einer Auswahl gefragt, welche der Rule Model Inheritance Testfälle durchgeführt werden soll.

1 abstractionOnly
2 conformanceOnly
3 dynamicBehavioralDetectionOnly
4 inheritanceOnly

5 staticBehaviouralDetectionOnly
6 structuralDetectionOnly

Der User gibt die Zahl, als Integer Wert, ein, welche Art von Testfälle dieser durchführen möchte. Basierend auf der Auswahl des Users, verläuft das Programm wie in den folgenden Abschnitten beschrieben. Die vom User getätigten Eingaben in den folgenden Abschnitten werden anschließend an das Evaluierungsprogramm übergeben.

abstractionOnly

Um diese Art der Testfälle durchzuführen wird als Input ein Module gefordert, für das alle Vererbungsbeziehungen aufgelöst wurden – Resolved Module.
Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, sowie Anzahl von Output und Input Prädikaten Werte welche der User als Zahl, Integer, eingeben soll.
Rules: (Integer Eingabe User)
Facts: (Integer Eingabe User)
Output: (Integer Eingabe User)
Input: (Integer Eingabe User)

conformanceOnly

Um diese Art der Testfälle durchzuführen werden als Input erkannte/gemeldete Änderungsvorgänge und Modifikationsbeschränkungen gefordert – detected/reported modification and operations modification restrictions.
Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, sowie Anzahl von Output und Input Prädikaten Werte welche der User als Zahl, Integer, eingeben soll.
Rules: (Integer Eingabe User)
Facts: (Integer Eingabe User)
Output: (Integer Eingabe User)
Input: (Integer Eingabe User)

dynamicBehavioralDetectionOnly

Um diese Art der Testfälle durchzuführen werden als Input Meta-Repräsentationen von Resultset-Paaren (Eltern- und Child-Resultsets) und Vererbungsbeziehungen von Modulen, für die Ergebnismengen angegeben sind gefordert – Meta-representations of resultset pairs (parent and child resultsets) and inheritance relations of modules for which resultsets are given.
Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, sowie Anzahl von Output und Input Prädikaten Werte welche der User als Zahl, Integer, eingeben soll.
Rules: (Integer Eingabe User)
Facts: (Integer Eingabe User)

Output: (Integer Eingabe User)
Input: (Integer Eingabe User)

inheritanceOnly

Um diese Art der Testfälle durchzuführen werden als Input ein Modul, das abgeleitet werden soll (toDerive) eine Modul-Metadarstellungen des relevanten Vererbungsbaums gefordert – module to be derived (toDerive) and module meta-representations of relevant inheritance tree.
Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, sowie Anzahl von Output und Input Prädikaten Werte welche der User als Zahl, Integer, eingeben soll.
Rules: (Integer Eingabe User)
Facts: (Integer Eingabe User)
Output: (Integer Eingabe User)
Input: (Integer Eingabe User)

staticBehaviouralDetectionOnly

Um diese Art von Testfälle durchzuführen wird als Input eine Meta-Repräsentationen von Modulen im Vererbungs-Teilbaum gefordert – Meta-representations of modules in inheritance subtree.
Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, sowie Anzahl von Output und Input Prädikaten Werte welche der User als Zahl, Integer, eingeben soll.
Rules: (Integer Eingabe User)
Facts: (Integer Eingabe User)
Output: (Integer Eingabe User)
Input: (Integer Eingabe User)

structuralDetectionOnly

Um diese Art von Testfälle durchzuführen wird als Input eine Metadarstellungen des relevanten Modulvererbungsbaums gefordert – Meta-representations of relevant module inheritance tree.
Hierfür wird eine bestimmte Anzahl an Regeln und Fakten gefordert, sowie Anzahl von Output und Input Prädikaten Werte welche der User als Zahl, Integer, eingeben soll.
Rules: (Integer Eingabe User)
Facts: (Integer Eingabe User)
Output: (Integer Eingabe User)
Input: (Integer Eingabe User)

3.2. Evaluierungsprogramm

Die Komponente des Evaluierungsprogramms steht im Mittelpunkt des Systems und agiert als User Interface mit dem Benutzer. Die Implementierung einer grafischen Darstellung vom User Interface wird als unwichtig eingestuft, weil dies nicht zum Ziel dieses Projekts gehört und wird nicht durchgeführt.
Die Eingaben der Daten vom User wird über die Konsole erfolgen. Der User agiert mit der Komponente des Evaluierungsprogrammes über die User Schnittstelle.
Die vom User eingegeben Daten, über die gewünschten Testfälle, werden im Evaluierungsprogramm erfasst und über eine der Datengeneratoren Schnittstellen an den jeweils betroffenen Datengenerator weitergegeben.
Die vom Datengenerator erzeugten Daten werden zurück an das Evaluierungsprogramm übermittelt, über die betroffene Datengenerator Schnittstelle.
Erhält das Evaluierungsprogramm die generierten Testdaten vom Datengenerator wird ein Zeitobjekt erstellt. Dieses Zeitobjekt enthält Datum und die Uhrzeit, wann der Test gestartet wurde.
Anschließend übergibt das Evaluierungsprogramm den Code (bzw. die Testfälle), der vom Datengenerator erzeugte wurde, über die Vadalog Schnittstelle, welche im Abschnitt 2.4 genauer beschrieben wird, an die von der Universität zur Verfügung gestellte Vadalog Ausführung, damit diese den Code (bzw. die Testfälle) ausführt und die Performanceergebnisse zurückliefert.
Die vom Benutzer eingegeben Daten und auch die Performanceergebnisse des Testfalles werden vom Evaluierungsprogramm an die Schnittstelle zur Datenspeicherung übergeben. Diese Schnittstelle wird in Abschnitt 2.5 genauer erläutert.

3.1. CBR Datengenerierung Schnittstelle

Die Schnittstelle für die CBR Datengenerierung ist die Verbindung zwischen dem CBR Datengenerator und dem Evaluierungsprogramm.
Das Evaluierungsprogramm ruft über diese Schnittstelle den CBR Datengenerator auf. Dabei werden die vom Benutzer eingegeben Daten, die Anzahl der Rules und die Anzahl der Facts, welche im Evaluierungsprogramms gespeichert wurden, über diese Schnittstelle an den Datengenerator übergeben.
Basierend auf diesen Eingaben generiert der Datengenerator die gewünschten Testdaten.
Die vom Generator generierten Daten werden anschließend über diese Schnittstelle an das Evaluierungsprogramm übergeben.

3.2. Datengenerator CBR

Hier wird die Funktonalität 4.1.1 aus der Anforderungsanalyse (bzw. Datengenerierung von CBR Datengenerator) realisiert.

Wünscht der Benutzer Testfälle betreffend CBR, erhält der Datengenerator vom Evaluierungsprogramm über seine zugehörige Schnittstelle die Informationen, wie viele Testdaten er generieren soll.
Der Datengenerator erhält vom Evaluierungsprogramm die vom der User Schnittstelle übertragenen Daten zu den zu generierenden Daten, diese beinhalten:

- Anzahl Parameter (Integer)
- Anzahl Parameter Values (Integer)
- Anzahl Contexte (Integer)
- Anzahl Business Cases (Integer)

Diese geforderten Testdaten werden vom Datengenerator generiert.
Die Datengenerierung erfolgt mit mehreren Methoden, welche alle einen spezifischen Teil des Codes generieren mithilfe von definierten Templates, also einer Art von Vorlage, woran der Code generiert wird und mithilfe von zufällig generierten Worten ergänzt wird.
Die Generierung der CBR Testdaten erfolgt mithilfe eines selbst implementierten Random Wort Generator.
Die konkrete Implementierung dieses Wort Generators sieht folgendermaßen aus:

```
public class GeneratorRandomString {

	public static String getRandomString(int length) {
		String ALPHABET = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
		final SecureRandom RANDOM = new SecureRandom();

		StringBuilder sb = new StringBuilder();
		for (int i = 0; i < length; ++i) {
			sb.append(ALPHABET.charAt(RANDOM.nextInt(ALPHABET.length())));
		}
		return sb.toString();
	}

}
```

Die oben gezeigte Implementierung kann bei der Implementierung des gesamten Evaluierungsprogrammes noch abgeändert werden.
Diese Implementierung liefert ein Wort, bzw eine zufällig aneinander gereihte Variation von Buchstaben, mit der durch den Parameter bestimmten Länge.
Für die Implementierung der Testdaten wird eine Zufallszahl zwischen 4 und (exklusive) 8 ausgewählt, welche die Länge der Testdaten bestimmen soll, um zu vermeiden, dass alle generierten Wörter dieselbe Länge haben.
Diese Zufallszahl wird folgendermaßen, mithilfe einer Java Bibliothek, ThreadLocalRandom, generiert:

```
int randomNumber = ThreadLocalRandom.current().nextInt(4, 8);
```

Der CBR Testcode wird durch eine Methode generiert. Diese Methode ruft weitere Hilfsmethoden auf, um eine Übersichtlichkeit und Strukturiertheit des Codes zu gewährleisten. Jede Hilfsmethode generiert einen Abschnitt des gesamten CBR Testcodes.
Die Attribute und Hauptmethode können daher wie folgt aussehen:

Um die generierten Daten im Programm dynamisch zu erhalten, werden die generierten Daten in Listen oder einer anderen passenden Collection gespeichert, dies kann bei der konkreten Implementierung noch abgeändert werden.

```
//Attribute
private static String CBRCode;
private List<String> contexts;
private List<String> businessclasses;
private List <String> contextclasses;
private List <String> parameters;
private List <String> parameterValues;
```

```
//Hauptmethode
	private static String CBRCode;

	public static String generateCBRCode(int parameters, int parameterValues, int contexts, int businessCases) {

		CBRCode = "";
		CBRCode += generateContextClass();
		CBRCode += generateBusinessCaseClass();
		CBRCode += generateParameters(parameters);
		CBRCode += generateParameterValuesAndHierachys();
		CBRCode += generateContexts(contexts);
		CBRCode += generateDetermineParemeterValues(parameterValues);
		CBRCode += generateBusinessCases(businessCases);
		CBRCode += generateStaticCode();

		return CBRCode;
	}
```

Der CBR Testcode ist daher vom Objekttyp ein String, welcher aus den String Rückgabewerten der Hilfsmethoden zusammengesetzt wird.

Die einzelnen Code Teile werden anhand von definierten Bestandteilen des Codes in Verbindung mit zufällig generierten Worten, als Parameter z.B., generiert.
Dies lässt sich am Beispiel von der Generierung der Parameter zeigen, welches wie folgt aussehen kann und in der endgültigen Implementierung noch abgeändert werden kann.

```
private static String generateParameters(int count) {

		List<String> parameters = new ArrayList<String>();
		int paramCount = count;
		String generatedParameters = "";

		for (int i = 0; i < paramCount; i++) {
// to prevent to have all parameters at the same length, a random Number generator is used.

			int randomNumber = ThreadLocalRandom.current().nextInt(4, 8);
			parameters.add(GeneratorRandomString.getRandomString(randomNumber));
		}
		for (int i = 0; i < parameters.size(); i++) {
			generatedParameters += "hasParameter(\"aimCtx,\"" + parameters.get(i) +
"\"). ";
		}
		generatedParameters += "\n";

		for (int i = 0; i < parameters.size(); i++) {
			generatedParameters += "parameter(\"" + parameters.get(i) + "\").\n";
		}
		generatedParameters += "\n";

		return generatedParameters;
	}
```

Mit dieser Methode werden auch die anderen Bestandteile des CBR Codes generiert. Die statischen Teile des Codes werden unverändert übernommen und zu dem Return String hinzugefügt, mithilfe der Methode generateStaticCode(), welche alle statischen Teile des Programmes generiert und zurückliefert.
Die Testdaten werden also vom Generator erzeugt und der fertige Test Code wird über die Schnittstelle wieder zurück an das Evaluierungsprogramm übergeben.

3.3. Rule Model Inheritance Datengenerierung Schnittstelle

Diese Schnittstelle dient als Verbindung zwischen dem Rule Model Inheritance Datengenerator und des Evaluierungsprogramms.
Das Evaluierungsprogramm ruft über diese Schnittstelle den Rule Model Inheritance Datengenerator auf. Dabei werden die vom Benutzer eingegeben Daten, die Anzahl der Rules und die Anzahl der Facts, welche im Evaluierungsprogramms gespeichert wurden, über diese Schnittstelle an den Datengenerator übergeben.
Basierend auf diesen Eingaben generiert der Datengenerator die gewünschten Testdaten, dies wird im Abschnitt 1.2 genauer erläutert.
Die vom Generator generierten Daten werden anschließend über diese Schnittstelle an das Evaluierungsprogramm übergeben.

3.4. Datengenerator Rule Model Inheritance

Hier wird die Funktonalität 4.1.2 aus der Anforderungsanalyse (bzw. Datengenerierung von Inheritance Datengenerator) realisiert.

Wünscht der Benutzer Testfälle betreffend der Rule Model Inheritance, erhält der Datengenerator vom Evaluierungsprogramm über seine zugehörige Schnittstelle die Informationen, wie viele Testdaten er generieren soll. Der Datengenerator erhält vom Evaluierungsprogramm die vom der User Schnittstelle übertragenen Daten, diese beinhalten:

- Anzahl Fakten (Integer)
- Anzahl Regeln (Integer)

Diese geforderten Testdaten werden vom Datengenerator generiert.
Die Testdaten werden vom Generator erzeugt und der fertige Test Code wird über die Schnittstelle wieder zurück an das Evaluierungsprogramm übergeben.
Die Datengenerierung erfolgt mit mehreren Methoden, welche alle einen spezifischen Teil des Codes generieren mithilfe von definierten Templates, also einer Art von Vorlage, woran der Code generiert wird und mithilfe von zufällig generierten Worten ergänzt wird.
Die Generierung der RMI Testdaten erfolgt mithilfe eines selbst implementierten Random Wort Generator, welcher im Abschnitt Datengenerator CBR näher beschrieben und beispielhaft abgebildet wurde.
Die Attribute und Hauptmethode können daher wie folgt aussehen:

```
//Attribute
```

```
private static String RMICode;
private static String program;
private List<String> rules;
private List<String> relationalAtoms;
private List <String> nonRelationalAtoms;
private List <String> annotations;
private List <String> terms;
```

```
//Hauptmethode
```

```
public static String generateRMICode(int rules, int facts, int input, int output) {

            RMICode = "";
            RMICode += generateProgram();
            RMICode += generateRules(int rules);
```

```
            RMICode += generateRelationalAtoms(int facts);
            RMICode += generateNonRelationalAtoms(int facts);
            RMICode += generateAnnotaions();
            RMICode += generateTerms();

            return RMIcode;
        }
```

Durch die Methode generateProgram() wird der Name des Programms, mit Hilfe vom Random Wort Generator, erstellt. Dieser Name wird in Form eines Strings an den bestehenden String RMICode angehängt. In die Methode gehen keine Paramater hinein.

Durch die Methode generateRules(int rules) werden die Regeln, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Die Anzahl dieser Regeln bestimmt ein Integer, der als der einzige Paramater in die Methode hineingeht. Anschließend wird diese Liste in Kombination mit dem Namen des Programms als ein String zurückgegeben.

Durch die Methode generateRelationalAtoms(int facts) werden Atome, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Die Anzahl dieser Atome bestimmt ein Integer, der als der einzige Parameter in die Methode hineingeht. Anschließend wird diese Liste in Kombination mit seinem Namen (hasName), einer Regel (hasPositive/NegativeHeadAtom) (hasPositive/NegativeBodyAtom) und einem Term mit Zahl (hasArgument) als String zurückgegeben.

Durch die Methode generateNonRelationalAtoms(int facts) werden Atome, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Die Anzahl dieser Atome bestimmt ein Integer, der als der einzige Parameter in die Methode hineingeht. Anschließend wird diese Liste in Kombination mit seinem Namen (hasName), einer Regel (hasPositive/NegativeHeadAtom) (hasPositive/NegativeBodyAtom), einem Term mit Zahl (hasArgument) und einem String (hasSerialisation) als String zurückgegeben.

Durch die Methode generateAnnotations() werden die Annotationen, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Anschließend wird diese Liste in Kombination mit einer Regel (hasAnnotation), ihrem Namen (hasName) und einem Term mit einer Zahl (hasArgument) als String zurückgegeben.

Durch die Methode generateTerms() werden die Terms, die in Kombination mit dem Random Wort Generator erstellt wurden, in eine String Liste gespeichert. Anschließend wird diese Liste in Kombination mit einem Literal (hasSerialization) als String zurückgegeben.

3.5. Vadalog Schnittstelle

Die Vadalog Schnittstelle ist die Verbindung zur Ausführung des generierten Codes. Diese Schnittstelle, wird von der Universität zur Verfügung gestellt.
Der generierte Code wird zur Ausführung vom Evaluierungsprogramm übergeben.

Bei der Implementierung unseres Programms wird ein Dummy anstatt der Vadalog Schnittstelle implementiert.
Im Programm wird eine Dummy Methode callVadalog() implementiert. Diese gibt die Performancewerte der Code Ausführung zurück.
Im speziellen werden folgende Werte zurückgegeben:

- Die Ausführungsdauer in Sekunden
  - Wird durch einen Random Zahlengenerator erzeugt
  - Datentyp: Double
- Fehler
  - Gibt an, ob ein Fehler vorgelegen hat
  - Wird durch einen Random Zahlengenerator, im Wertbereich zwischen 0 und 1 erzeugt
  - Datentyp: Boolean
- Die durchschnittliche Auslastung des Prozessors in Prozent
  - Wird durch einen Random Zahlengenerator erzeugt
  - Datentyp: Double

Diese zufällig generierten Werte werden anschließend wieder an das Evaluierungsprogramm übergeben um es zu ermöglichen, diese Ergebnisse über die Speicherung Schnittstelle zu übergeben und im externen Speicher dauerhaft zu speichern.

3.6. Speicherung Schnittstelle

Mithilfe der Speicherung Schnittstelle werden die Performance Ergebnisse vom Evaluierungsprogramm an den externen Speicher übergeben.
Diese Verbindung wird durch eine JDBC Verbindung (Java Database Connectivity) ermöglicht. Die zu speichernden Daten werden mithilfe von JDBC in der SQL Datenbank des externen Speichers persistent erhalten.

Konkret werden folgende Daten an die Datenbank zur Speicherung übergeben:

- Eine einzigartige ID des Tests

    - ID
        - Datentyp: Integer

- Das Datum des Durchführungszeitpunkt
    - Tag des Datums
        - Datentyp: Integer
    - Monat des Datum
        - Datentyp: Integer
    - Jahr des Datums
        - Datentyp: Integer
- Die Uhrzeit zum Durchführungszeitpunkt
    - Stunde der Ausführung
        - Datentyp: Integer
    - Minute der Ausführung
        - Datentyp: Integer
    - Sekunde der Ausführung
        - Datentyp: Integer

- Der Testtyp, welchen der Benutzer auswählt
    - Testtyp
        - Datentyp: String

- Die Anzahl der zu testenden Elemente
    - Anzahl der Parameter
        - Datentyp: Integer
    - Anzahl der Regeln
        - Datentyp: Integer
    - Anzahl der Fakten
        - Datentyp: Integer
    - Anzahl der Contexte
        - Datentyp: Integer
    - Anzahl der Business Cases
        - Datentyp: Integer
    - Anzahl der Parameter Values
        - Datentyp: Integer
    - Anzahl der Input Prädikate
        - Datentyp: Integer
    - Anzahl der Output Prädikate

        - Datentyp: Integer

- Diverse Testergebnisse
    - Dauer der Ausführung
        - Datentyp: Double
    - Fehlermeldung
        - Datentyp: Boolean
    - Durchschnittliche Prozessorauslastung
        - Datentyp: Double

Die Schnittstelle zur Speicherung wird vom Evaluierungsprogramm aufgerufen um Daten zur Speicherung an die Datenbank, den externen Speicher, zu übergeben.

3.7. Externer Speicher

Hier wird die Funktonalität 4.1.12 aus der Anforderungsanalyse (bzw. Speicherung der Evaluierungswerte auf einem Programmexternen Ort) realisiert.

Im externen Speicher werden die Ergebnisse der Performanceevaluierung gespeichert. Zusätzlich werden die vom Benutzer geforderten Testdaten hier persistiert, um die Tests nachvollziehen zu können und gegebenenfalls die Testergebnisse für spätere Evaluierungsarbeiten wiederverwenden zu können. Somit besteht auch die Möglichkeit, nicht nur ein Testergebnis mehrerer Durchläufe zu analysieren, sondern auch zeitlich unabhängige Tests untereinander vergleichen zu können.

Bei dem externen Speicher handelt es sich um eine mySQL Datenbank (Version 5.7), welche von einem externen Hoster zur Verfügung gestellt wird, und somit grundsätzlich dauerhaft verfügbar (99,9% Verfügbarkeit) ist. Dadurch ist es möglich, mithilfe des Evaluierungsprogramms die gewünschten Daten extern zu speichern, um sie persistent zu erhalten, wenn das Programm geschlossen wird. Durch die Datenbank ist ebenso der Mehrbenutzerbetrieb, sowie eine programmunabhängige Speicherung der Daten möglich. Die Einrichtung und Bearbeitung der Datenbank und Tabellen erfolgt über das phpMyAdmin Interface (Version 4.8.3) des Hosters.

Adresse:	https://e42776-phpmyadmin.services.easyname.eu
Server:	`e42776-mysql.services.easyname.eu`
Benutzer:	`u48005db23`
Datenbank:	`u48005db23`
Passwort:	`prdke2018`

Die Daten werden über die Speicherung Schnittstelle vom Evaluierungsprogramm an den Speicher übergeben. Nachfolgend in der Tabelle wird die Struktur der Tabelle, genauer der Spalten, erklärt:

Name der Spalte	Datentyp*	Größe	Kurzbeschreibung
id	INT	3	die einzigartige ID des durchgeführten Tests
day	INT	2	Tag des Datums der Durchführung des Tests
month	INT	2	Monat des Datums der Durchführung des Tests
year	INT	4	Jahr des Datums der Durchführung des Tests
hour	INT	2	Stunde des Datums der Durchführung des Tests
minute	INT	2	Minute des Datums der Durchführung des Tests
second	INT	2	Sekunde des Datums der Durchführung des Tests
testtype	VARCHAR	15	der durchgeführte Testtyp (CBR o. RMI Variante)
noParam	INT	2	Anzahl der Parameter
noRules	INT	2	Anzahl der Regeln
noFacts	INT	2	Anzahl der Fakten
noCont	INT	2	Anzahl der Contexte
noBusCase	INT	2	Anzahl der Business Cases
noParamVal	INT	2	Anzahl der Parameter Werte
noInPr	INT	2	Anzahl der Input Prädikate
noOutPr	INT	2	Anzahl der Output Prädikate
exTime	DOUBLE	4.2	Dauer der Ausführung
Errors	BOOLEAN	2	Fehlermeldungen
noErrSec	DOUBLE	2.2	Anzahl der Fehlermeldungen je Sekunde
cpuUsage	DOUBLE	2.2	durchschn. Prozessorauslastung in Prozent

*Datentyp und Schreibweise gemäß mySQL

Tabelle 1: SQL Datenbank Tabellen

3.8. Klassenstruktur

Das Evaluierungsframework wird als Java Projekt „EvaluationFramework“ implementiert. Die geplante Klassenstruktur ist in der nachfolgenden Grafik, Abbildung 3, dargestellt.

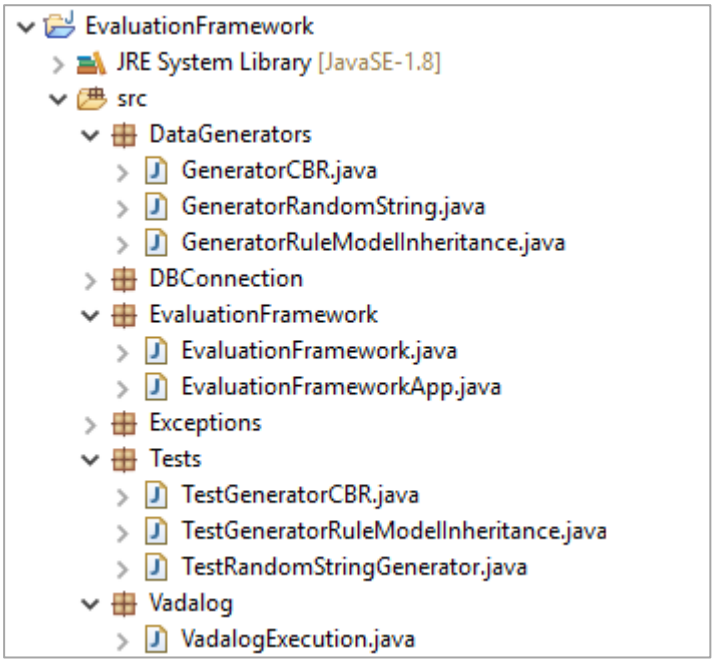


Abbildung 3: geplante Klassenstruktur

Die Implementierung wird geplant aus Packages bestehen:

- DataGenerators
- DBConnection
- EvaluationFramework
- Exceptions
- Tests
- Vadalog

Im Folgenden werden die Packages und die dazugehörigen geplanten Klassen beschrieben.

3.8.1. DataGenerators

In diesem Package befinden sich die beiden Datengeneratoren, den Datengenerator für CBR und den Datengenerator für Rule Model Inheritance. Sie bilden die Funktionalität der oben beschriebenen Systemkomponenten der Datengeneratoren ab. Jeder der beiden Datengeneratoren erhält eine Klasse, um die Funktionalität der Codegeneration durch strukturierte Methoden abbilden zu können.

Ebenso befindet sich in diesem Package der selbst implementierte RandomStringGenerator, welcher einen beliebig langen String aus Groß- und Kleinbuchstaben, bestimmt durch den mitgegebenen Parameter, erstellt. Dieser wird für die Codegeneration benötigt.

Zuordnung Funktionalitäten Anforderungsanalyse DataGenerators
Hier wird die Funktonalität 4.1.3 aus der Anforderungsanalyse (bzw. Durchführung von CBR Performancetests) realisiert.
Der Datengenerator CBR erhält die von der User Schnittstelle übertragenen Daten. Diese beinhalten:

- Anzahl Parameter (Integer)
- Anzahl Parameter Values (Integer)
- Anzahl Contexte (Integer)
- Anzahl Business Cases (Integer)

Zusätzlich werden folgende Funktionalitäten realisiert:

- 4.1.4 Durchführung von Inheritance Performancetests mit Singe-Inheritance ohne Modifikation
- 4.1.5 Durchführung von Inheritance Performancetests mit Multi-Inheritance ohne Modifikation
- 4.1.6 Durchführung von Inheritance Performancetests mit Singe-Inheritance mit Modifikation
- 4.1.7 Durchführung von Inheritance Performancetests mit Multi-Inheritance mit Modifikation
- 4.1.8 Durchführung von Inheritance Performancetests mit Singe-Inheritance mit Restrictions
- 4.1.9 Durchführung von Inheritance Performancetests mit Singe-Inheritance ohne Restrictions
- 4.1.10 Durchführung von Inheritance Performancetests mit Multi-Inheritance mit Restrictions
- 4.1.11 Durchführung von Inheritance Performancetests mit Multi-Inheritance ohne Restrictions

Für alle oben angeführten Funktionalitäten (bzw. von 4.1.4 bis 4.1.11) erhält der Datengenerator RMI die von der User Schnittstelle übertragenen Daten. Diese beinhalten:

- Anzahl Regeln (Integer)
- Anzahl Fakten (Integer)

3.8.2. DBConnection

Dieses Package bildet die Funktionalität der Datenspeicherung ab und beschäftigt sich mit der Implementierung von JDBC um die persistente Datenspeicherung zu ermöglichen.

3.8.3. EvaluationFramework

In dem Package „EvaluationFramework“ befindet sich das Herzstück des Programmes, die Umsetzung der Systemkomponente des Evaluierungsprogrammes.

In der Klasse „EvaluationFramework“ werden die Funktionalitäten des Evaluierungsprogrammes realisiert. Die Klasse „EvaluationFrameworkApp“ beinhaltet die ausführbare Main Methode, welche die Funktionen aufruft und auch als Schnittstelle zum User agiert und die Eingaben des Users entgegennimmt.

3.8.4. Exceptions

In dem Package „Exceptions“ werden alle Excpetions implementiert und verwaltet, welche für das Fehler Handling notwendig sind. Es sollen so z.B. ungültige Eingaben vom User verhindert werden können.

3.8.5. Tests

In diesem Package werden die Tests für die Generatoren durchgeführt und verwaltet. Um die richtige Funktionalität zu gewährleisten, müssen die Datengeneratoren strukturiert getestet werden.

3.8.6. Vadalog

In dem „Vadalog“ Package werden die Dummy Methoden implementiert, welche die Performanceergebnisse simulieren sollen. Diese Ergebnisse werden durch mehrere Methoden simuliert und in der Klasse „VadalogExecution“ implementiert.

4. Abbildungsverzeichnis

5. Tabellenverzeichnis