Entwicklerdokumentation zum Quality-of-Service-Tool

31. Juli 2014

Inhaltsverzeichnis

1 Überblick

Das Quality-of-Service-Tool ist ein Werkzeug zur Berechnung der Servicequalität von Geschäftsprozessen (QoS), die als ereignisgesteuerte Prozesskette (EPK) modelliert wurden. Obwohl es primär als Add-on für das Modellierungswerkzeug Bflow* Toolbox entwickelt wurde, kann es auch eigenständig über die Kommandozeile aufgerufen werden.

Entwickelt wurde das Quality-of-Service-Tool vollständig in Java. Diese Dokumentation soll dabei helfen, die Struktur des Tools zu verstehen und einen Einstieg in die Weiterentwicklung zu finden, beispielsweise um die Berechnung weiterer QoS-Maßzahlen zu implementieren.

2 Grundlagen zur Struktur des Quality-of-Service-Tools

Die Berechnung der QoS-Maßzahlen wird in mehreren Arbeitsschritten durchgeführt.

1. Laden eines EPK-Modells aus einer Datei
2. Überprüfen ob notwendige Attribute vorhanden und gültig sind
3. Berechnen der QoS-Maßzahlen
4. Ausgabe von Ergebnissen bzw. Fehlern

Diese Arbeitsschritte werden durch entsprechende Komponenten realisiert, die in den nachfolgenden Abschnitten genauer beschrieben werden. Das Komponentendiagramm in Abbildung 1 (S. 4) stellt den Zusammenhang zwischen den Komponenten grafisch dar.

2.1 Aufruf des Tools

Gestartet wird das Tool durch Aufruf der Main-Methode in der Klasse *QoSToolLauncher* aus dem Paket `de.fh_zwickau.bumb_l_b.qos_tool`. Hier werden die Kommandozeilenparameter ausgewertet und es wird eine Instanz der Klasse *QoSTool* aus dem gleichen Paket erzeugt. Diese Klasse steuert den Ablauf der Arbeitsschritte und verbindet die einzelnen Komponenten miteinander.

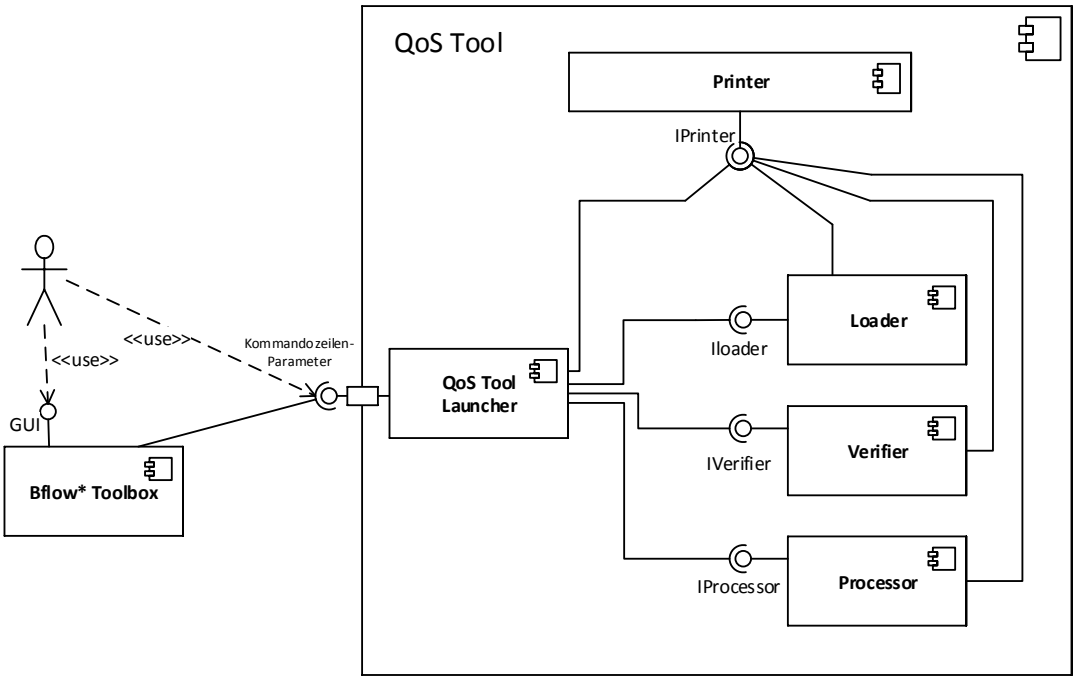

Abbildung 1: Komponenten des QoS-Tools

2.2 Ereignisgesteuerte Prozessketten in Java modellieren

Als Grundlage der Datenstruktur für ereignisgesteuerte Prozessketten wird das JBPT-Framework verwendet. Das Framework umfasst Klassen zur Modellierung verschiedener Geschäftsprozessmodelle als Graph, sowie grundlegende Funktionen zur Modellanalyse und Navigation auf Graphen.

In JBPT ist keine Möglichkeit vorgesehen, EPK-Elemente mit beliebigen Attributen (z. B. Kosten, Zuverlässigkeit) zu versehen. Diese Funktionalität wird durch die *Custom*-Klassen* im Paket `de.fh_zwickau.bumb_l_b.qos_tool.customJBPT` ergänzt, welche im QoS-Tool anstatt der ursprünglichen JBPT-Klassen genutzt werden. Die *Custom*-Klassen* sind direkte Subklassen ihrer Pendants *Event*, *Function*, ... aus dem JBPT-Framework. Sie bilden die Knoten bzw. Kanten des Graphen.

`IEpc<ControlFlow<FlowNode>, FlowNode, NonFlowNode>` ist der Datentyp für den Graph selbst. Hierbei wird die ursprüngliche JBPT-Implementierung genutzt.

2.3 Ausgaben realisieren

Das QoS-Tool besitzt keine grafische Oberfläche. Alle Ausgaben erfolgen daher in Textform. Die für Ausgaben verantwortliche Komponente heißt „Printer“. Zugehörige Java-

Klassen befinden sich im Paket `de.fh_zwickau.bumb_l_b.qos_tool.printer`.

Das Interface *IPrinter* definiert die Schnittstelle für Printer-Implementierungen. Es definiert Methoden für die Ausgabe von Nachrichten in drei Klassen: Fehler, Warnungen und Informationen.

Derzeit existieren zwei Printer-Implementierungen, welche die Nachrichten unterschiedlich formatiert auf der Standardausgabe ausgeben.

Die Klasse *DefaultPrinter* gibt ein für Menschen leicht lesbares Format aus. Sie ist für die Verwendung des Tools auf der Kommandozeile vorgesehen.

Die Klasse *BflowPrinter* ist für die Verwendung des Tools als Bflow*-Addon vorgesehen. Sie gibt Nachrichten in einer speziellen Syntax aus, die von der Bflow* Toolbox interpretiert werden kann. Dies ermöglicht Nachrichten in der Bflow*-Problems-View auszugeben und betroffene Elemente im Diagramm zu markieren.

Die Printer-Objekte unterschiedlichen Typs können durch die statische Fabrik-Klasse *StaticPrinterFactory* erzeugt werden. Dieses Konzept wird im *QoSToolLauncher* (siehe Abschnitt 2.1, S. 3) genutzt, um je nach Kommandozeilenparameter unterschiedliche Printer zu verwenden.

2.3.1 Texte verwalten

Die Texte für Ausgaben werden zentral in der Ressourcendatei `src/resources/Messages.properties` verwaltet. Dabei werden den Ausgabetexten Schlüssel zugeordnet, über die sie später abgerufen werden können. Das Konzept erleichtert die Pflege der Texte. Durch Hinzufügen von Ressourcendateien in weiteren Sprachen kann dieses Konzept auch leicht zur Lokalisierung verwendet werden.

Listing 1: Beispiel: Ausschnitt einer Ressourcendatei

```
Kategorie.Meldungsschluessel=Hallo Welt
common.no_epc_loaded=No epc is loaded.
common.loading_file_failed=Loading file %s failed.
```

Die Klasse *Message* im Paket `de.fh_zwickau.bumb_l_b.qos_tool` bietet mit der statischen Methode `get(String key)` die Möglichkeit, über den Schlüssel auf die Nachrichtentexte zuzugreifen.

Listing 2: Beispiel: Zugriff auf Textressource

```
String text = Message.get("Kategorie.Meldungsschluessel");
```

2.4 Laden von EPK-Modellen

Das Laden von EPK-Modellen wird durch die Komponente „Loader“ realisiert. Die zugehörigen Java-Klassen befinden sich im Paket `de.fh_zwickau.bumb_l_b.qos_tool.loader`.

Das Interface *IEpcLoader* definiert die Schnittstelle für Loader-Implementierungen. Es werden Methoden zum Einlesen von EPKs aus Dateien bzw. Strings definiert. Die Methoden geben jeweils ein Objekt vom Typ `IEpc<ControlFlow<FlowNode>, FlowNode, NonFlowNode>` zurück. Dieser Typ entspricht einer ereignisgesteuerten Prozesskette aus dem JBPT-Framework (vgl. Abschnitt 2.4, S. 6).

Derzeit existiert eine Loader-Implementierung: *EpmlEpcLoader* dient dem Einlesen von Modellen, die im EPML-Format vorliegen. EPML ist ein XML-basiertes Austauschformat für ereignisgesteuerte Prozessketten, welches auch von der bflow* Toolbox unterstützt wird.

Der *EpmlEpcLoader* verwendet die Klasse *EpmlParser*. Dort wird die XML-Datei geparst und entsprechend der XML-Elemente ein JBPT-EPK-Objekt erzeugt.

2.5 Überprüfen von EPK-Modellen

Bevor die eigentliche Berechnung gestartet wird, soll bewertet werden, ob dies überhaupt möglich ist. Zum einen wird geprüft, dass alle EPK-Elemente mit den für die Berechnung notwendigen Attributen versehen sind. Zum anderen wird sichergestellt, dass die Attributwerte den festgelegten Wertebereich nicht verletzen. Gefundene Probleme werden auf der Problems-View in Bflow* ausgegeben und die betroffenen Elemente im Diagramm markiert.

Die Komponente zum Überprüfen von EPK-Modellen heißt „Verifier“. Im Paket `de.fh_zwickau.bumb_l_b.qos_tool.verifier` befinden sich die zugehörigen Java-Klassen.

Das Interface *IEpcVerifier* definiert die Schnittstelle für Verifier-Implementierungen. Es definiert zwei Methoden: Die `verifiy()`-Methode startet die Überprüfung für die übergebene EPK und liefert einen booleschen Wert zurück. Ist dieser Rückgabewert *false*, so enthält das Modell Fehler und kann nicht berechnet werden.

Die Methode `getProcessableAttributeTypes()` liefert für die zuletzt überprüfte EPK ein Menge von QoS-Attributtypen (z. B. Zeit, Kosten), für die eine Berechnung möglich ist. Sie wird sinnvollerweise dann ausgewertet, wenn die `verifiy()`-Methode den Wert *true* zurückliefert.

2.6 Berechnen der Servicequalität

Die Berechnung der Servicequalität wird in der Komponente „Processor“ ausgeführt. Die zugehörigen Java-Klassen befinden sich im Paket `de.fh_zwickau.bumb_l_b.qos_tool.processor`.

Das Interface *IQoSProcessor* definiert die Schnittstelle für Processor-Implementierungen. Es beinhaltet nur eine Methode `process(IEpc<...> epc, AttributeType type)` zur Berechnung von Qualitätsmaßzahlen. Der Methode wird ein Geschäftsprozessmodell und ein Enum für das zu berechnende QoS-Attribut übergeben.

Unterstützt werden aktuell die QoS-Attribute *time*, *cost* und *reliability* (Zeit, Kosten und Zuverlässigkeit). Der Processor berechnet zu jedem dieser Attribute Minimum, Maximum und Durchschnitt für das Geschäftsprozessmodell.

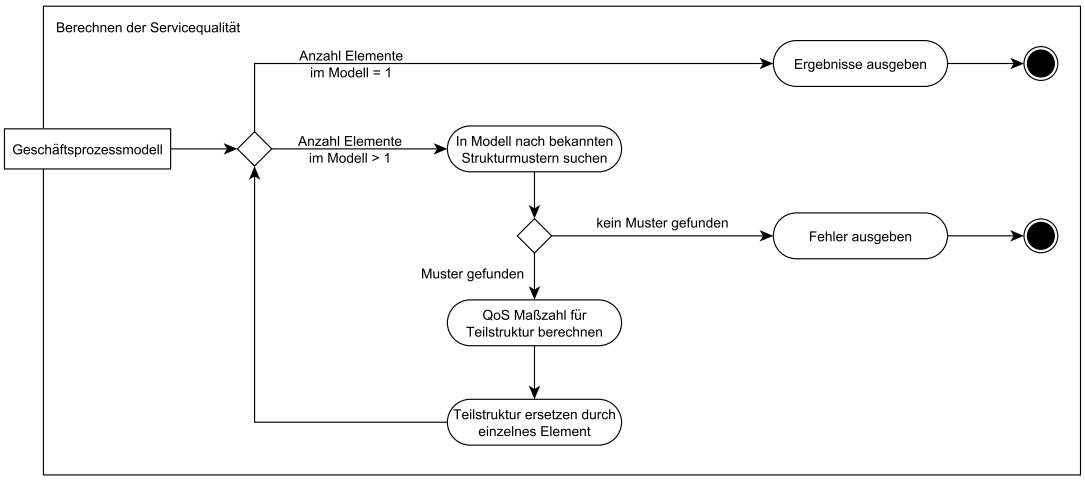

Abbildung 2: Berechnen der Servicequalität

Das Aktivitätsdiagramm in Abb. 2, S. 7 zeigt den grundsätzlichen Ablauf für die Ermittlung der Servicequalität.

Der Processor erhält das Geschäftsprozessmodell in Form eines JBPT-EPK-Objekts. Das Geschäftsprozessmodell wird mittels Mustersuche nach Teilstrukturen durchsucht, die einem vorgegebenen Muster entsprechen. Für eine gefundene Teilstruktur werden die Minimal-, Maximal- und Durchschnittswerte der QoS-Attribute berechnet und in einem einzelnen, neuen EPK-Element gespeichert. Anschließend wird die gefundene Teilstruktur durch dieses einzelne Element ersetzt.

Dieser Reduktionsvorgang wird solange wiederholt, bis die EPK nur noch aus einem Element besteht oder die Mustersuche keine weiteren Ergebnisse liefert. Im letzteren Fall kann die EPK nicht auf ein einzelnes Element reduziert werden. Ihre ursprüngliche

Struktur entspricht nicht den Anforderungen des QoS-Tools - die Qualitätsmaßzahlen können nicht bestimmt werden.

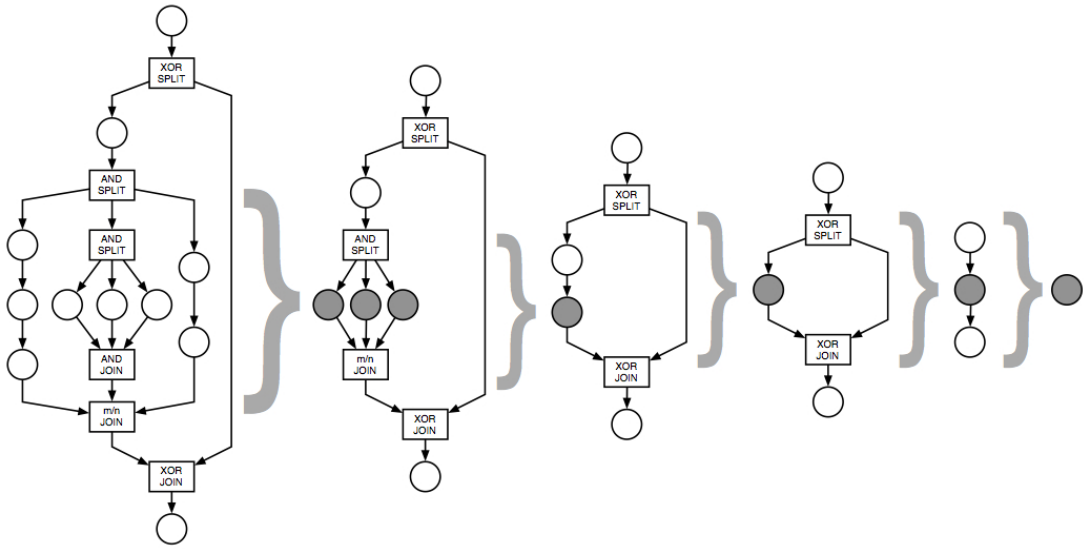

Abbildung 3: Reduktion einer EPK (aus Jaeger, 2006)

Abbildung 3 (S. 8) verdeutlicht den Reduktionsvorgang an einem Beispiel. Auf der linken Seite ist das ursprüngliche Modell dargestellt. Nach rechts hin wird die EPK nach dem jeweils nächsten Reduktionsschritt dargestellt.

Im ersten Suchdurchlauf werden drei einfache Teilstrukturen identifiziert: Es werden zwei Aneinanderreihungen von einfachen Elementen, also Funktionen bzw. Ereignisse gefunden. Weiterhin wird ein And-Konnektor-Paar entdeckt, bei dem auf allen inneren Kontrollflusskanten nur ein einfaches Element enthalten ist. Im Beispiel entsprechen diese drei Teilstrukturen den Pfaden zwischen den beiden äußeren And-Konnektoren. Die Servicequalität wird berechnet und die drei Teilstrukturen werden durch jeweils ein Element (grau dargestellte Kreise) ersetzt.

Im zweiten Schritt wird das vormals äußere And-Konnektor-Paar als einfache Teilstruktur identifiziert und analog zum ersten ersten Schritt ersetzt.

In den nachfolgenden Schritten werden immer wieder einfache Teilstrukturen durch einzelne Elemente ersetzt, bis die EPK aus nur noch einem Element besteht, welches das Ergebnis enthält.

3 Hinweise zum Einfügen weiterer QoS-Attribute

Dieser Abschnitt soll als Leitfaden zur Implementierung der Berechnung weiterer Qualitätsmaßzahlen im Tool dienen. Am Beispiel der Qualitätsmaßzahl Sicherheit (*safety*) wird gezeigt, welche Änderungen an den Komponenten nötig sind.

3.1 Hilfsklasse zur Attributverwaltung anpassen

Die Klasse *AttributeHelper* im Paket `de.fh_zwickau.bumb_l_b.qos_tool.customJBPT` ist eine Hilfsklasse zur Attributverwaltung, in der das neue Attribut eingefügt werden muss.

EPK-Elemente verwalten ihre Attribute in einer Map. Die Werte der Schlüssel sind vom Typ `AttributeType`, einem Enum, welches in *AttributeHelper* definiert ist. Zunächst sollte `AttributeType` um das neue Attribut *safety* erweitert werden.

```
public enum AttributeType {
  SAFETY,
  SAFETY_MIN,
  SAFETY_MAX,
  SAFETY_AVG,
  ...
}
```

In den Konstanten `QOS_FUNCTION_ATTRIBUTES`, `QOS_EVENT_ATTRIBUTES`, ... wird definiert, welche Attributtypen für welche EPK Elemente geeignet sind. Das neue Attribut im Beispiel soll an Funktionen verwendet werden.

```
private static final Set<AttributeType> QOS_FUNCTION_ATTRIBUTES =
    new HashSet<AttributeType>(Arrays.asList(AttributeType.TIME,
    AttributeType.COST, AttributeType.RELIABILITY, AttributeType.
    SAFETY));
```

Weiterhin sollte in der Hilfsmethode `isAttributeOfType(AttributeType type, Object obj)` ergänzt werden, von welchem Datentyp das Attribut ist.

3.2 Die Loader-Komponente anpassen

Der vorhandene Loader für EPML-Dateien soll nun erweitert werden, so dass er auch das *safety*-Attribut an Funktionen einlesen kann. Das Einlesen der Attribute findet in der Klasse *EpmlParser* statt, die an zwei Stellen geändert werden muss.

Zunächst muss das neue Attribut in die `ATTRIBUTE_MAP` eingefügt werden. Die Map dient dazu, den Attributbezeichnungen aus der EPML-Datei Attributtypen zuzuordnen. Die Attributbezeichnung entspricht dabei dem, was in der „Attribute View“ der Bflow* Toolbox unter „Attribute Name“ angegeben wird (vgl. Abbildung 4, S. 10).

```
static {
  ...
  ATTRIBUTE_MAP.put("safety", AttributeType.SAFETY);
}
```

Im Beispiel wird davon ausgegangen, dass das Attribut in Bflow* als „safety“ angegeben wird. Der zugeordnete Attributtyp ist der in Abschnitt 3.1 (S. 9) angelegte.

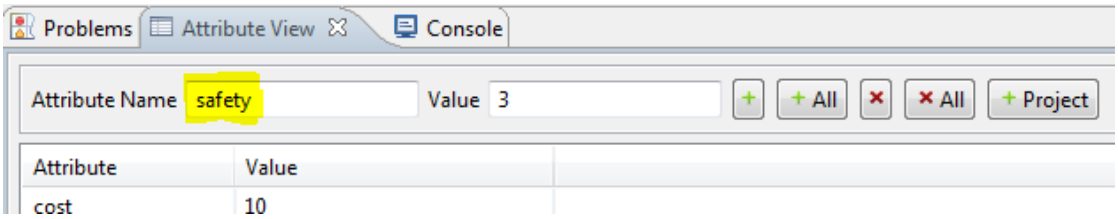

Abbildung 4: Attribute View der Bflow* Toolbox

In der Methode `parseAttributeforElement(IContainAttribute epcElement, String name, String value)` muss analog zu den bereits vorhandenen Attributen noch festgelegt werden, ob und wie der Wert des neuen Attributs als Zahl interpretiert wird. Standardmäßig werden Attributwerte als String interpretiert.

3.3 Die Verifier-Komponente anpassen

Der Verifier iteriert über die Elemente der EPK und prüft dabei das Vorhandensein von Attributen sowie deren Wertebereiche. Zum Einfügen des neuen Attributs *safety*, welches an Funktionen notiert wird, muss die Klasse *EpcVerifier* angepasst werden.

Zunächst sollten zwei boolesche Klassenvariablen deklariert werden.
Die Variable `hasAnySafetyAttributes` speichert dabei, ob das Attribut an wenigstens einem EPK Element notiert wurde. Die Variable `safetyCanBeCalc` speichert, ob das Attribut durch die Processor-Komponente berechnet werden soll.

```
private boolean hasAnySafetyAttributes = false;
private boolean safetyCanBeCalc = true;
```

Die Methode `hasAnyAttributes(IContainAttribute i)` ermittelt, ob ein Attribut an wenigstens einer Funktion notiert ist. Hier sollte das neue Attribut *safety* hinzugefügt werden.

```
public void hasAnyAttributes(IContainAttribute i) {
  ...
  if(checkValue(i,AttributeType.SAFETY) != null) {
    hasAnySafetyAttributes = true;
  }
}
```

Analog dazu existieren die Methoden `xorEdgesHaveAttributes(...)` für die Attribute an XOR-Konnektoren und `andConnectorGetAttributes(...)` für die Attribute an AND-Konnektoren.

Weiterhin sollten die Wertebereiche des neuen Attribut überprüft werden. Hierzu wird die Methode `checkAttributes(...)` erweitert. Sie dient der Überprüfung von Attributen an Funktionen.

```
public void checkAttributes(IContainAttribute i, String name,
    String id) {
  ...
  if(hasAnySafetyAttributes) {
    Double value = checkValue(i, AttributeType.SAFETY);
    if(value == null) {
      //Attribut ist nicht an jeder Funktion
      safetyCanBeCalc = false;
      epcRejected = true;
    }
    else {
      if(value < 0) {
        //Wertebereich verletzt
        safetyCanBeCalc = false;
        epcRejected = true;
      }
      ...
    }
  }
}
```

Analog dazu existieren die Methoden `checkXorAttributes(...)` und `checkAndAttributes (...)` zur Überprüfung von Attributen an Konnektoren bzw. damit verbundenen Kontrollflusskanten. Für das Beispielattribut *safety* müssen diese nicht verändert werden.

In der Methode `verify(...)` muss entsprechend des Wertes von `safetyCanBeCalc` noch der Attributtyp für *safety* zur Menge der berechenbaren Attribute hinzugefügt werden.

```
public boolean verify(IEpc<ControlFlow<FlowNode>, FlowNode,
    NonFlowNode> epc) {
```

```
    ...
    if(hasAnySafetyAttributes) {
      if(safetyCanBeCalc) {
        calculatableAttributeTypes.add(AttributeType.SAFETY);
      }
      else { epcRejected = true; }
    }

    //accept or decline of epc
    if(epcRejected) {
    ...
  }
```

3.4 Die Processor-Komponente anpassen

Die Processor-Komponente verwendet eine Mustersuche, um innerhalb des Geschäftsprozessmodells Teilstrukturen zu finden, die simplen Mustern entsprechen (vgl. Abs. 2.6, S. 7). Beispiele für die verwendeten Suchmuster werden in Abbildung 5 (S. 14) grafisch dargestellt.

Die Berechnung der Servicequalität für diese Teilstrukturen ist abhängig vom Attributtyp. Daher existiert für jeden berechenbaren Attributtyp eine eigene Ersetzungsstrategie. Die Strategien implementieren das Interface *IReplacementStrategy*. Das Interface definiert eine Reihe von Callback-Methoden, die für gefundene Teilstrukturen die Servicequalität berechnen und in Form eines einzelnen EPK-Elements zurückliefern.

Bei der Implementierung einer neuen Ersetzungsstrategie ist es sinnvoll, eine bereits vorhandene Strategie wie *CostReplacementStrategy* zu studieren oder als Grundlage für eine eigene Strategie zu nutzen. Das nachfolgende Listing zeigt eine (gekürzte) Ersetzungsstrategie für das Beispielattribut *safety*.

```
public class SafetyReplacementStrategy implements
    IReplacementStrategy {
  @Override
  public FlowNode replaceEmpty(final PatternMatcherResult result) {
    IContainAttribute replacement = new CustomFunction("Dummy");
    addMissingAttributeValues(replacement);
    return (FlowNode)replacement;
  }

  @Override
  public FlowNode replaceLinear(final PatternMatcherResult result) {
```

```
  IContainAttribute replacement = new CustomFunction("LinearDummy")
      ;

  //TODO: Berechnung der Werte für Minimum, Maximum, Durchschnitt

  replacement.setAttribute(AttributeType.SAFETY_MIN, ...);
  replacement.setAttribute(AttributeType.SAFETY_MAX, ...);
  replacement.setAttribute(AttributeType.SAFETY_AVG, ...);
  return (FlowNode)replacement;
}

@Override
public FlowNode replaceAnd(final PatternMatcherResult result) {
  //analog zu replaceLinear implementieren
  ...
}

@Override
public FlowNode replaceXor(final PatternMatcherResult result)
    throws ReplacementException {
  //analog zu replaceLinear implementieren
  ...
}

@Override
public FlowNode replaceCycle(final PatternMatcherResult result)
    throws ReplacementException {
  //analog zu replaceLinear implementieren
  ...
}

@Override
public void addMissingAttributeValues(final IContainAttribute node)
    {
  Double safety;
  if(node.hasAttribute(AttributeType.SAFETY)) {
    safety = (Double) node.getAttribute(AttributeType.SAFETY);
  }
  else {
    safety = 0.0;
    node.setAttribute(AttributeType.SAFETY, safety);
  }
  //prepare min, max, avg
  node.setAttribute(AttributeType.SAFETY_MIN, safety);
```

```
    node.setAttribute(AttributeType.SAFETY_MAX, safety);
    node.setAttribute(AttributeType.SAFETY_AVG, safety);
  }
}
```

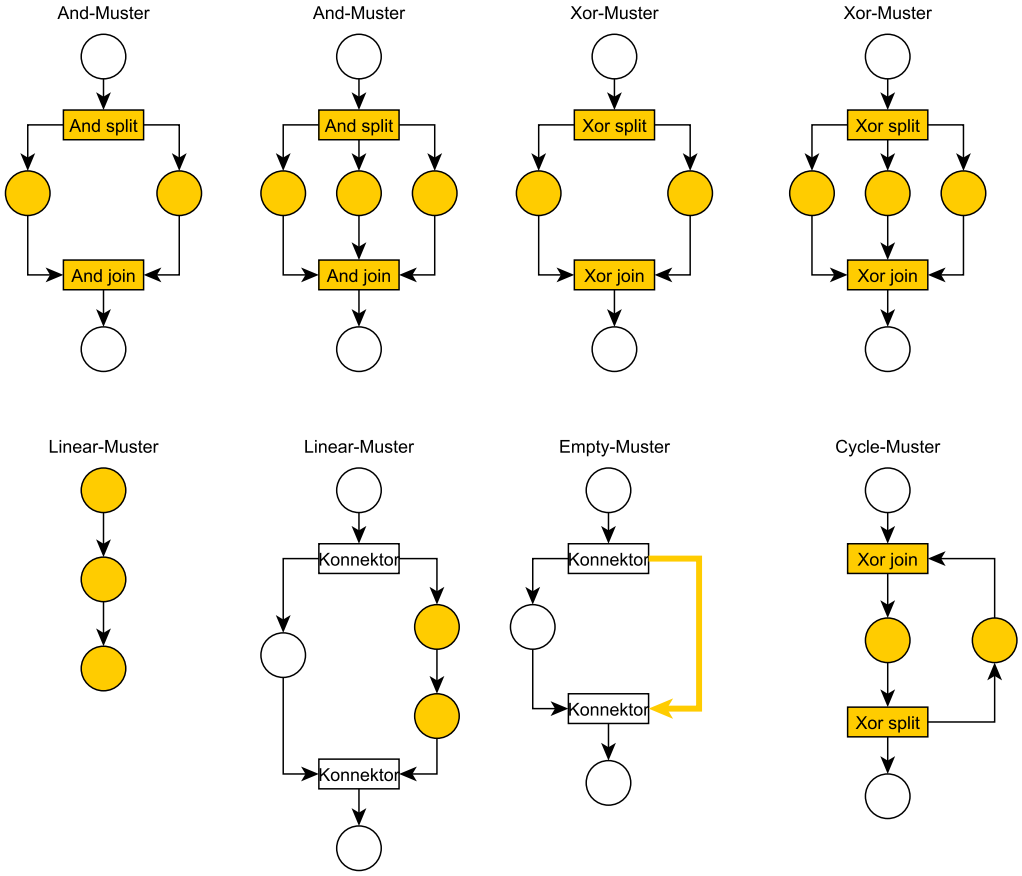

Abbildung 5: Beispiele für Suchmuster

Die Methode `replaceEmpty(...)` der Ersetzungsstrategie wird aufgerufen, falls das Empty-Muster gefunden wurde. Da hierbei ein zusätzliches Element in die EPK eingefügt wird, sollte die Methode ein Element mit Werten zurückliefern, welche die weitere Berechnung nicht beeinflussen. Bei Attributen, die üblicherweise addiert werden, wie z. B. Zeit und Kosten, ist der neutrale Wert die Null - das Element verursacht dann keinen Kosten oder Zeitaufwand.

Die Methode `replaceLinear(...)` wird aufgerufen, falls das Linear-Muster gefunden wurde. Dieses trifft Teilstrukturen, die nur aus einer Reihe von Funktionen und Ereignissen bestehen und keine Konnektoren enthalten.

Die Methode `replaceAnd(...)` wird aufgerufen, falls das And-Muster gefunden wurde. Dieses trifft Teilstrukturen, die aus einem Paar von And-Konnektoren bestehen, auf deren inneren Pfaden jeweils genau ein Element angeordnet ist.

Die Methode `replaceXor(...)` wird aufgerufen, falls das Xor-Muster gefunden wurde. Dieses trifft Teilstrukturen, die aus einem Paar von Xor-Konnektoren bestehen, auf deren inneren Pfaden jeweils genau ein Element angeordnet ist.

Die Methode `replaceCycle(...)` wird aufgerufen, falls das Cycle-Muster gefunden wurde. Dieses trifft Teilstrukturen, die aus einem Paar von Xor-Konnektoren bestehen, die durch genau zwei Pfade, auf denen jeweils genau ein Element angeordnet ist, miteinander verbunden sind. Die Pfade verlaufen dabei in entgegengesetzt Richtung - sie bilden einen Zyklus. Bei der Berechnung von Zyklen muss beachtet werden, dass die Ausführwahrscheinlichkeiten von Pfaden wie bei Xor-Splits als Einzelwahrscheinlichkeit im Attributtyp `PROBABILITY` angegeben sein kann oder alternativ als Wahrscheinlichkeitstupel im Attributtyp `PROBABILITY_LIST` an der Kante vom Split zum Join (Rücksprungkante) gegeben ist.

Die Methode `addMissingAttributeValues(...)` wird für alle Funktionen und Ereignisse aufgerufen. Falls für eines dieser Elemente ein Attributwert nicht gesetzt ist, wird ein neutraler Attributwert gesetzt. Weiterhin werden die Werte für Minimum, Maximum und Durchschnitt initialisiert.

In der Klasse *QoSProcessor* muss die neue Ersetzungsstrategie nun noch bekannt gemacht werden. Damit sollte das neue Attribut implementiert sein.

```
public class QoSProcessor implements IQoSProcessor {
  ...
  public void process(IEpc<ControlFlow<FlowNode>, FlowNode,
      NonFlowNode> epc, AttributeType type) {
    ...
    switch(type) {
      case SAFETY:
        reducer = new Reducer((IEpc<ControlFlow<FlowNode>, FlowNode,
            NonFlowNode>) cloneEpc(epc), new SafetyReplacementStrategy
            ());
        resultMin = AttributeType.SAFETY_MIN;
        resultMax = AttributeType.SAFETY_MAX;
        resultAvg = AttributeType.SAFETY_AVG;
        break;
      ...
    }
    ...
  }
  ...
```

Literatur

[Jaeger 2006] Jaeger, Michael C.: *Optimising quality of service for the composition of electronic services.*, Berlin Institute of Technology, Dissertation, 2006