UNIVERSITÄT
KOBLENZ · LANDAU
Institut für Softwaretechnik

FB 4
Informatik

Abbildung von grUML nach XSD soamig

Eckhard Großmann
Sascha Strauß
Tassilo Horn
Volker Riediger

Nr. 15/2009

Arbeitsberichte aus dem Fachbereich Informatik

Universität Koblenz-Landau • Campus Koblenz • Universitätsstraße 1 • 56070 Koblenz
Telefon +49 261 287-0 • http://www.uni-koblenz.de

Die Arbeitsberichte aus dem Fachbereich Informatik dienen der Darstellung vorläufiger Ergebnisse, die in der Regel noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar. Alle Rechte vorbehalten, insbesondere die der Übersetzung, des Nachdruckes, des Vortrags, der Entnahme von Abbildungen und Tabellen – auch bei nur auszugsweiser Verwertung.

The “Arbeitsberichte aus dem Fachbereich Informatik“ comprise preliminary results which will usually be revised for subsequent publication. Critical comments are appreciated by the authors. All rights reserved. No part of this report may be reproduced by any means or translated.

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Herausgeber / Edited by:

Der Dekan:
Prof. Dr. Zöbel

Die Professoren des Fachbereichs:
Prof. Dr. Bátori, Prof. Dr. Beckert, Prof. Dr. Burkhardt, Prof. Dr. Diller, Prof. Dr. Ebert, Prof. Dr. Furbach, Prof. Dr. Grimm, Prof. Dr. Hampe, Prof. Dr. Harbusch, Prof. Dr. Sure, Prof. Dr. Lämmel, Prof. Dr. Lautenbach, Prof. Dr. Müller, Prof. Dr. Oppermann, Prof. Dr. Paulus, Prof. Dr. Priese, Prof. Dr. Rosendahl, Prof. Dr. Schubert, Prof. Dr. Staab, Prof. Dr. Steigner, Prof. Dr. Troitzsch, Prof. Dr. von Kortzfleisch, Prof. Dr. Walsh, Prof. Dr. Wimmer, Prof. Dr. Zöbel

Kontaktdaten der Verfasser

Eckhard Großmann, Sascha Strauß, Tassilo Horn, Volker Riediger
Institut für Softwaretechnik
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
EMail: mmce@uni-koblenz.de, strauss@uni-koblenz.de, horn@uni-koblenz.de, riediger@uni-koblenz.de

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument legt den Standard für die Transformation von grUML-Schemas (GraphUML, [BHR+09]) nach XSD (XML Schema Definition) fest und ist im Rahmen des Arbeitspakets 5.2 „Prototypische SOAMIG-Parser und -Unparser realisieren" im SOAMIG-Projekt entstanden.

Das Ziel ist der Austausch von TGraphen (typisierten, attributierten, angeordneten, gerichte Graphen [ERW08]) über XML-Dokumente. Zur Spezifikation des Austauschformats wird XSD eingesetzt. Dies erlaubt eine Validierung der XML-Instanzen auf syntaktischer Ebene.

Der Ausgangspunkt ist ein gegebenes Schemas in grUML-Notation[1], welches nach XSD transformiert werden soll. Mit der generierten XSD existiert ein Beschreibungsmittel für Graph-Instanzen in XML. Die dadurch beschriebenen XML-Dokumente sind flach, d.h. alle Elemente sind direkt dem *root*-Element untergeordnet.

1.1 Überblick

Ein Beispielschema[2] für einen TGraphen, das einen Ausschnitt des SOAMIG-Java-Schemas darstellt, ist in Abb. 1 dargestellt. Das Ergebnis[3] der Transformation von grUML nach XSD ist in Listing 1 zu sehen.

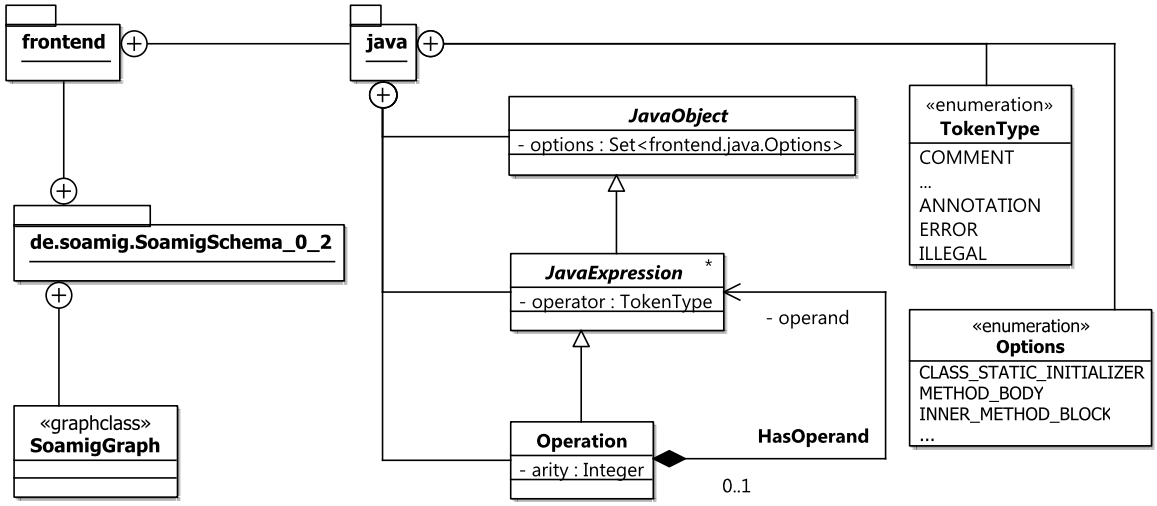

Abbildung 1: Ein Beispielschema für die Beschreibung von Operationen in Java.

Bei der Umwandlung werden ausschließlich nicht-abstrakte Klassen und Assoziationen transformiert, weil ohnehin keine Instanzen von abstrakten Klassen und Assoziationen existieren können. Diese Klassen und Assoziationen werden jeweils durch einen *complexType* und ein *element* abgebildet. Packages werden nicht transformiert, sondern über den vollqualifizierten Namen von Klassen und Assoziationen implizit repräsentiert.

[1] grUML-Diagramme werden mit normalen UML-Werkzeugen erstellt, wie zum Beispiel dem IBM Rational Software Architect.
[2] Der Aufzählungstyp `TokenType` ist aus Darstellungsgründen gekürzt worden.
[3] Die vordefinierten Typen sind aus dem Ergebnis der Transformation nicht enthalten.

```
<?xml version="1.0" ?>
<xsd:schema
    xmlns:xsd="http://soamig.de/schema/SoamigSchema-0-3"
    xmlns:smg="http://soamig.de/SoamigSchema-0-2"
    targetNamespace="http://soamig.de/schema/SoamigSchema-0-3">
    <!--Default types-->
    ...
    <!--Graph-type-->
    <xsd:element name="SoamigGraph" type="smg:GT_SoamigGraph"/>
    <xsd:complexType name="GT_SoamigGraph">
        <xsd:complexContent>
            <xsd:extension base="smg:BT_Graph">
                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                    <xsd:element name="frontend.java.Operation"  type="smg:VT_frontend.
                        java.Operation"/>
                    <xsd:element name="frontend.java.HasOperand" type="smg:ET_frontend.
                        java.HasOperand"/>
                </xsd:choice>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!--Vertex-types-->
    <xsd:complexType name="VT_frontend.java.Operation">
        <xsd:complexContent>
            <xsd:extension base="smg:BT_Vertex">
                <xsd:attribute name="arity"    type="xsd:ST_INTEGER"/>
                <xsd:attribute name="operator" type="smg:ST_ENUM_frontend.java.TokenType"/
                    >
                <xsd:attribute name="options"  type="smg:ST_SET"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    !--Edge-types-->
    <xsd:complexType name="ET_frontend.java.HasOperand">
        <xsd:complexContent>
            <xsd:extension base="smg:BT_Edge"/>
        </xsd:complexContent>
    </xsd:complexType>
    <!--Enumeration-types-->
    <xsd:simpleType name="ST_ENUM_frontend.java.Options">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="CLASS_STATIC_INITIALIZER"/>
            <xsd:enumeration value="METHOD_BODY"/>
            <xsd:enumeration value="INNER_METHOD_BLOCK"/>
            ...
            <xsd:enumeration value="n"/>
        </xsd:restriction>
    </xsd:simpleType>
    ...
</xsd:schema>
```

Listing 1: Ergebnis der Transformation des Beispielschemas von Operationen in Java nach XSD.

Graphen werden gemäß der hier vorgestellten Transformation analog zum standardisierten Graphenaustauschformat GXL (Graph Exchange Language) [HSESW06] durch flache XML-Dokumente repräsentiert, d.h. die Knoten und Kanten werden als direkte Kinder des XML-root-Elements abgebildet. Im Gegensatz zu GXL werden jedoch Knoten- und Kantentypen durch explizit benannte XML-Elemente dargestellt, sodass hier auf Type-Links und externe Schemagraphen verzichtet werden kann.

1.2 Gliederung

Das Dokument ist folgendermaßen gegliedert: Im Abschnitt 2 werden zunächst die grundlegenden Konzepte von grUML und XSD beschrieben.

Im Abschnitt 3 auf Seite 7 wird die Transformation von grUML nach XSD vorgestellt. Dabei wird zunächst auf das generelle Vorgehen eingegangen. Danach werden vordefinierten Elemente erläutert, und dann wird die Umwandlung von Elementen aus einem konkreten Schema näher betrachtet. Abschließend wird die Abbildung von Attributen und Wertebereichen (Domains) definiert.

Im Abschnitt 4 auf Seite 16 wird kurz erläutert, wie XML-Schemas mit den Klassen `Rsa2Tg` und `SchemaGraph2XSD` erstellt werden können. In Unterabschnitt 4.1 auf Seite 16 werden die genauen Kommandozeilenoptionen beschrieben, die `SchemaGraph2XSD` zur Verfügung stellt.

Im Abschnitt A auf Seite 19 des Anhangs ist das Metamodell für grUML-Schemas enthalten. Der Abschnitt B auf Seite 19 fasst alle Einschränkungen zusammen, die durch die Transformation bedingt sind. Abschnitt C auf Seite 20 zeigt die vollständige XSD des Beispielschemas aus Abb. 1 und Abschnitt D auf Seite 22 enthält einen Beispielgraphen für ein XML-Codebeispiel.

1.3 Notation

Fest stehende Begriffe von grUML werden in einer `monospace`-Schrift geschrieben. Begriffe aus der Domäne von XSD werden *kursiv* notiert.

In späteren Codebeispielen werden Codeausschnitte aus Listing 1 geliefert. Codeausschnitte, die für das jeweilige Beispiel nicht von Belang sind, werden ausgespart bzw. mit ... markiert.
Verwendete Graph-Instanzen dienen der Verdeutlichung der Zusammenhänge zwischen Schema (vgl. Abschnitt 2.1)und Instanz und beziehen sich nicht auf einen konkreten Sachverhalt.

2 Grundlagen

Bevor mit der Transformation von grUML-Schemas begonnen werden kann, müssen zuerst die Begrifflichkeiten aus grUML und XSD erklärt werden.

2.1 grUML

grUML [BHR+09](Graph UML) ist eine Variante von UML-Klassendiagrammen und dient der Beschreibung von TGraph-Schemas [EWD+96]. Mit grUML ist es möglich, die Beziehungen zwischen Knoten und Kanten in einem Graph exakt festzulegen. Dabei sind alle Knoten und Kanten typisiert und attributiert.

Eine Beschreibung der Beziehungen und der Zusammensetzung eines TGraphen wird Schema genannt. Knoten werden in Schemas durch Klassen und Kanten durch Assoziationen bzw. Assoziationsklassen abgebildet.

In einem grUML-Schema sind mehrere grundlegende Elemente enthalten. So gibt es eine `Graph`-Klasse, sowie `Vertex`-, `Edge`-, `Aggregation`- und `Composition`-Klassen. Diese Klassen haben die `AttributedElement`-Klasse als Oberklasse gemein. Jedes `AttributedElement` kann Attribute mit unterschiedlichen Wertebereichen besitzen.

2.1.1 `Graph`-Klasse

Die `Graph`-Klasse beschreibt die Struktur des Graphen. Konkrete Graphen zur Laufzeit sind Instanzen dieser Graph-Klasse.

2.1.2 `Vertex`-Klassen

`Vertex`-Klassen beschreiben Knoten-Typen in einem Graph. In grUML werden sie als UML-Klassen mit Attributen modelliert.

2.1.3 `Edge`-, `Aggregation`- und `Composition`-Klassen

`Edge`-, `Aggregation`- und `Composition`-Klassen beschreiben Kanten-Typen in einem Graphen. In grUML werden sie durch UML-Assoziationen, -Aggregationen und -Kompositionen repräsentiert. Enthalten sie Attribute, so werden sie in grUML mit einer Assoziationsklasse modelliert.

2.1.4 Attribute und Wertebereiche

Attribute werden in grUML als UML-Attribute von Klassen und Assoziationsklassen modelliert. Per Konvention werden Attribute mit Namen in camelCase beginnend mit einem Kleinbuchstaben benannt.

Die verwendeten Domains sind vorgegeben. Sie können sein: `Boolean`, `Integer`, `Long`, `Double`, `String`, `Enum`, `Set`, `List`, `Map` und `Record`. `Set`, `List` und `Map` sind analog zu Java-Generics typisiert. `Set` bildet eine Menge und `List` eine Sequenz von Elementen einer bestimmten Domain ab. `Map` bildet Relationen bestehend aus Key-Value-Paaren ab.

2.1.5 Qualified Name

Der Qualified Name beschreibt in grUML den vollqualifizierten Namen eines Elements. Er setzt sich aus den Packages, in denen das entsprechende Element enthalten ist, und dem Namen des Elements zusammen. Die Namen der Packages und des Elements werden jeweils mit einem Punkt getrennt.

Zum Beispiel gibt es das Package „`frontend`" mit dem enthaltenen Package „`java`". Im Package „`java`" befindet sich ein Element namens „`Operation`". Der Qualified Name für „`Operation`" lautet dann: „`frontend.java.Operation`".

2.2 XSD

XSD[W3C04, W3C09a, W3C09b](XML Schema Definition) ist eine Beschreibungssprache für die Syntax von XML-Dokumenten, die das ältere DTD-Konzept (Document Type Definition) zur Beschreibung von XML-Dokumenten stark erweitert. XML-Dokumente können mit Hilfe eines Validator-Tools auf Schema-Konformität überprüft werden. XSD wird in XML geschrieben.

2.2.1 *schema*-Knoten

Der *schema*-Knoten ist der Wurzelknoten einer XSD.

2.2.2 *element*-Knoten

Ein *element* legt den Namen und den zu verwendenden Typen eines XML-Elements fest. Es kann außerdem das Vorkommen, die Abstraktheit und einige andere Eigenschaften des Elements genauer definieren.

2.2.3 *complexType*-Knoten

Ein *complexType* beschreibt einen Typen und legt damit die Struktur von XML-Elementen dieses Typs fest, indem dessen Attribute und Struktur (geschachtelte Kind-Elemente) definiert werden. Es gibt zwei verschiedene Verwendungsarten von *complexType*-Knoten: lokal[4] und global.

Global bedeutet, dass der *complexType*-Knoten als ein Unterknoten des *schema*-Knotens angegeben wird und einen Namen trägt.

Lokal bedeutet, dass der *complexType*-Knoten als Unterknoten des zu beschreibenden XML-Elements definiert wird - ohne Angabe eines Typnamens.

2.2.4 *simpleType*-Knoten

Ein *simpleType* beschreibt den Typen eines XML-Elements oder eines XML-Attributs. Es kann nur ein einfacher Datentyp spezifiziert werden und keine weiteren Attribute. Ein *simpleType* kann ebenfalls lokal[5] oder global definiert sein.

[4] Lokale *complexType*-Knoten werden bei der grUML2XSD-Transformation nicht verwendet.
[5] Lokale *simpleType*-Knoten werden bei der grUML2XSD-Transformation nicht verwendet.

3 Transformation von grUML-Schemas

In diesem Abschnitt wird die Transformation von grUML-Elementen festgelegt. Als generelles Beispiel liegt das Diagramm aus Abb. 1 diesem Abschnitt zu Grunde. Bei der Transformation von grUML-Schemas wird generell wie folgt vorgegangen:

Nicht abstrakte `Vertex`- und `Edge`-Klassen aus den zu transformierenden Schemas werden durch jeweils ein *element* und ein *complexType* in XSD repräsentiert.

Die *element*-Definition in XSD erhält als *name* den Qualified Name der Klasse bzw. Kante.

Die *complexType*-Definition wird global definiert und erhält ebenfalls als *name* den Qualified Name. Zur besseren Unterscheidung erhält jeder Qualified Name den Prefix „VT_" (VertexType) für Knoten- und „ET_" (EdgeType) für Kanten-Typen.

Attribute werden durch *attribute*-Definitions in den entsprechenden *complexType*-Definitionen wiedergegeben. Die Vererbung - im Speziellen die Mehrfachvererbung - von Klassen im grUML-Schema kann nicht durch Ausdrucksmittel von XSD modelliert werden. Aus dem Grund wird sie flach wiedergegeben, indem alle Attribute der Oberklassen in allen Subtypen explizit aufgeführt werden. Durch die qualifizierten Namen der `AttributedElement`-Definitionen werden alle `Package`s implizit definiert.

Die grUML-Basisklassen `AttributedElement`, `Vertex`, `Edge` und `Graph` sind als grundlegende *complexType*-Definitionen mit dem Prefix „BT_" (BaseType) festgelegt. Von diesen Definitionen werden alle Spezialisierungen über eine XSD-*extension* abgeleitet.

3.1 Vordefinierte Klassen

Die XSD-Definition von `AttributedElement` ist im Listing 2 abgebildet.

```
1 <xsd:complexType name="BT_AttributedElement" abstract="true" />
```

Listing 2: Definition der Klasse `AttributedElement` in XSD.

Die beiden Klassen `Graph` und `Vertex` sind Spezialisierungen der `AttributedElement`-Klasse. Zur eindeutigen Identifikation von `Graph`- und `Vertex`-Instanzen wird das Attribut *ID* als *XML-ID* definiert (vgl Listing 3).

Die Klasse `Edge` spezialisiert ebenfalls `AttributedElement` und gibt mittels der beiden Attribute *FROM* und *TO* vom Typ *IDREF* den Start- und Zielknoten der jeweiligen Kante an. Die Information der erlaubten Start- und Zielknotentypen geht bei dieser Transformation verloren, jedoch ist sie auf Instanzebene nicht erforderlich.[6] Die Klasse `Edge` ist im Listing 4 dargestellt. Zur Repräsentation der Inzidenzreihenfolge gibt es außerdem die optionalen *integer*-Attribute *FSEQ* und *TSEQ*.

Die grUML-Spezialisierungen `Aggregation` und `Composition` werden ebenfalls als `Edge` abgebildet, sodass eine eigene XSD-Spezifikation entfällt.

[6] Die exakte grUML-Schemainformation wird allerdings in einen Kommentar eingefügt, der sich vor der *complexType*-Definition befindet.

```
<xsd:complexType name="BT_Graph" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="smg:BT_AttributedElement">
            <xsd:attribute name="ID" type="xsd:ID" use="required"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="BT_Vertex" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="smg:BT_AttributedElement">
            <xsd:attribute name="ID" type="xsd:ID" use="required"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Listing 3: Definitionen der Klassen `Graph` und `Vertex` in XSD.

```
<xsd:complexType name="BT_Edge" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="smg:BT_AttributedElement">
            <xsd:attribute name="FROM" type="xsd:IDREF" use="required"/>
            <xsd:attribute name="TO"   type="xsd:IDREF" use="required"/>
            <xsd:attribute name="FSEQ" type="xsd:ST_INTEGER"/>
            <xsd:attribute name="TSEQ" type="xsd:ST_INTEGER"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Listing 4: Definitionen der Klasse `Edge` in XSD.

Knoten-, Kanten- und Inzidenzreihenfolgen

In TGraphen sind Knoten und Kanten in einer globalen Knoten- bzw. Kantensequenz angeordnet. Zusätzlich sind die zu einem Knoten inzidenten Kanten durch Inzidenzlisten geordnet. Beide Reihenfolgen müssen in XML-Instanzen dargestellt werden können.

Die Reihenfolge der beiden globalen Sequenzen wird durch die Reihenfolge der Elemente in der XML-Instanz festgelegt. Die Inzidenzreihenfolgen werden durch die optionalen Attribute *FSEQ* und *TSEQ* - jeweils für das from- und das to-Ende - einer Kante definiert.

Die *FSEQ*- bzw. *TSEQ*-Werte aller zu einem Knoten inzidenten Kanten legen eine Sortierung fest. Kommen Ordnungsnummern an einem Knoten mehrfach vor, so wird die Reihenfolge der ensprechenden Kanten anhand des Vorkommens in der XML-Datei ermittelt.

Beim vollständigen Fehlen der *FSEQ*- und *TSEQ*-Attribute werden die Inzidenzen in der Reihenfolge der Kanten-Elemente in der XML-Instanz angeordnet.

Sind nur einige, aber nicht alle inzidenten Kanten eines Knotens bezüglich der Inzidenzreihenfolge spezifiziert, so werden die nicht spezifizierten Kanten in der Reihenfolge der Kanten-Elemente in der XML-Instanz am Ende der Inzidenzliste angefügt.

Ein Beispiel für Inzidenzlisten in einem Graph ist in Abb. 2 dargestellt. Die Abbildung zeigt nur einen Ausschnitt des Graphen. Der entsprechende Graph in XML-Notation ist im Listing 5 dargestellt. Wegen der Größe des Graphen sind sowohl die Abbildung als auch das Listing nur Ausschnitte einer Grapheninstanz. Der komplette Graph als Abbildung und auch als XML-Code ist in Abschnitt D auf Seite 22 enthalten.

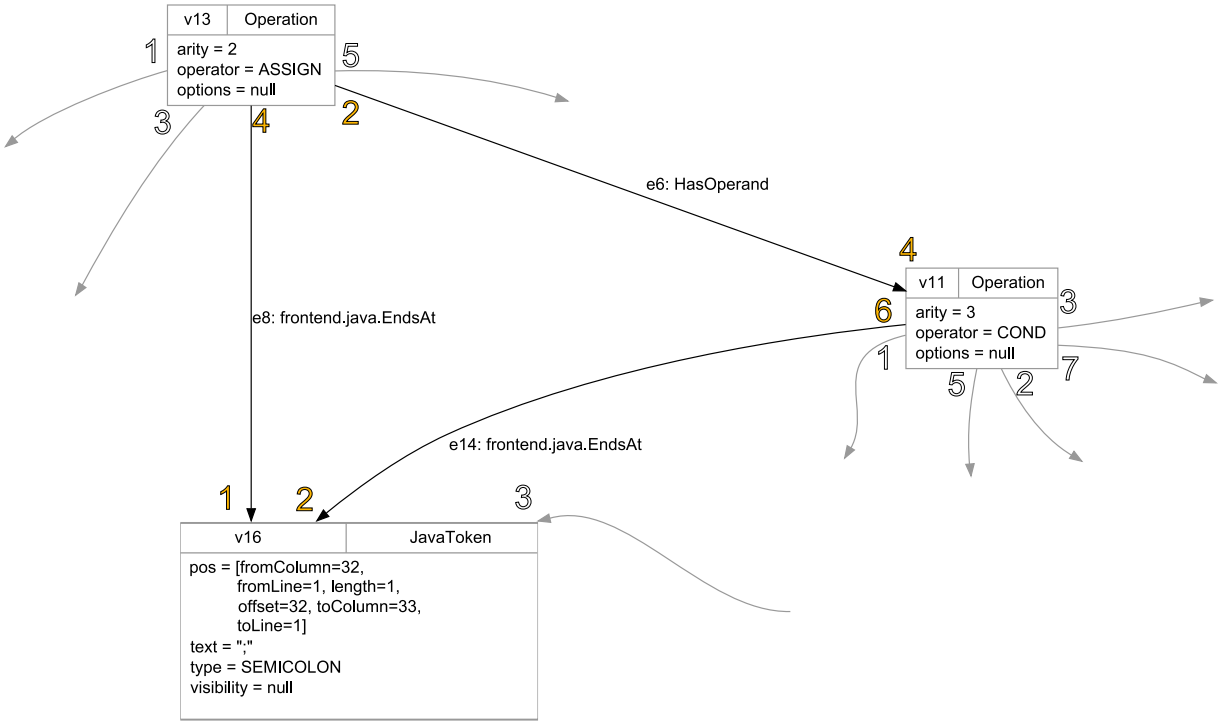

Abbildung 2: Kleiner Beispielgraph in dem die Kantensequenzen eingetragen sind.

```
<?xml version="1.0" encoding="UTF-8"?>
<smg:SoamigGraph
    xmlns:smg="http://soamig.de/schema/SoamigSchema-0-3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://soamig.de/schema/SoamigSchema-0-3 soamig.xsd"
    ID="g82f0c844-4077dbcc-dc06298-cd53151b">

    <frontend.java.Operation ID="v11" arity="3" operator="COND" options="n"/>
    <frontend.java.Operation ID="v13" arity="2" operator="ASSIGN" options="n"/>
    <frontend.java.JavaToken ID="v16" pos="(32 1 1 32 33 1)" text="&quot;;&quot;" type="
        SEMICOLON" visibility="n"/>

    <frontend.java.HasOperand FROM="v13" FSEQ="2" TO="v11" TSEQ="4"/>
    <frontend.java.EndsAt     FROM="v13" FSEQ="4" TO="v16" TSEQ="1"/>
    <frontend.java.EndsAt     FROM="v11" FSEQ="6" TO="v16" TSEQ="2"/>
</smg:SoamigGraph>
```

Listing 5: Beispiel eines Graphen in XML mit Inzidenzordnungen.

Es ist zu sehen, dass zu jedem Knoten die Reihenfolge der inzidenten Kanten gespeichert wird. Jeder Kantenanfang und jedes Kantenende haben eine Nummer entsprechend ihrer Position in der Inzidenzsequenz. Die Inzidenzsequenzen ermöglichen es, alle Kanten in der richtigen Reihenfolge abzulaufen.

In Abb. 2 ist zu sehen, dass von `TokenList` jeweils Kanten mit den Inzidenznummern 2 bis 10 auf `JavaToken` verweisen. Mit der Inzidenzsequenz kann die richtige Reihenfolge der `JavaToken` verfolgt werden, ohne dass jeder Knoten betrachtet werden muss. Anhand der Variable „offset" wird entschieden, in welcher Reihenfolge die `JavaToken` geschrieben werden müssen.

3.2 Schema

Der *schema*-Knoten ist der Wurzelknoten in XSD und erhält den Namen des abzubildenden grUML-Schemas. Der Name setzt sich aus dem PackagePrefix und dem Namen des grUML-Schemas zusammen. Im Beispiel aus Abb. 1 auf Seite 2 ist das Schema „`de.soamig.schema.SoamigSchema-0-3`" zu sehen. Der Name besteht zuerst aus einer Top-Level-Domain-Bezeichnung, dann einem Projektnamen oder Ähnlichem, danach evtl. weiteren Bezeichnungen und schließlich dem Namen des Schemas. Zuerst steht immer „http://" gefolgt von der Projektbezeichnung, einem Punkt „.", der Top-Level-Domain und einem Schrägstrich „/". Alles zusammen ergibt bei dem Beispielschema „http://soamig.de/". Die restlichen Bezeichnungen werden in der vorkommenden Reihenfolge aufgeführt, jeweils mit Schrägstrichen getrennt Aus diesen Informationen wird der neue Namespace gebildet, der des Beispielschemas heißt „http://soamig.de/schema/SoamigSchema-0-3". Der Namespace wird an ein frei definierbares Kürzel gebunden. Bei den gezeigten Codebeispielen wird das Kürzel *smg* für SOAMIG verwendet.

Alle weiteren XSD-Definitionen werden innerhalb des *schema*-Knotens eingefügt.

3.3 `Graph`-Klassen

Die in einem Schema spezifizierte `Graph`-Klasse wird mit einem *element* und einem *complexType* abgebildet und erweitert den vordefinierten Basistypen. Es darf nur eine `Graph`-Instanz im späteren XML-Dokument geben.

Die *element*-Definition wird als direkter Kindknoten der *schema*-Definition eingefügt und erhält als *name* den Qualified Name der zu beschreibenden `Graph`-Klasse.

Die *complexType*-Definition erweitert den vordefinierten *complexType* „BT_Graph" und erbt somit das *ID*-Attribut und erhält das Präfix „GT_" für Graph Type.

Erweitert wird die Definition mit den *element*-Definitionen aller `Vertex`- und `Edge`-Klassen. Sie dürfen im späteren XML-Dokument als Unterelemente des *Graph*-Elements in beliebiger Reihenfolge und Anzahl auftreten. Aus diesem Grund werden alle *element*-Definitionen in einer *choice*-Definition aufgeführt, deren Attribute *minOccurs* auf „0" und *maxOcccurs* auf „unbounded" gesetzt sind. Die *complexType*-Definition enthält gegebenenfalls alle weiteren Attribute der `Graph`-Klasse.

Ein Beispiel für eine `Graph`-Klasse in XSD ist im Listing 6 und eine entsprechende Instanz im Listing 7 gegeben. Die beiden Typen der *element*-Definitionen sind in den Listings 8 auf der nächsten Seite und 9 auf Seite 12 definiert.

```
<xsd:element name="SoamigGraph" type="smg:GT_SoamigGraph"/>

<xsd:complexType name="GT_SoamigGraph">
    <xsd:complexContent>
        <xsd:extension base="smg:BT_Graph">
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="frontend.java.Operation"  type="smg:VT_frontend.java.
                    Operation"/>
                <xsd:element name="frontend.java.HasOperand" type="smg:ET_frontend.java.
                    HasOperand"/>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Listing 6: Beispiel für eine spezialisierte `Graph`-Klasse in XSD.

```
<smg:SoamigGraph ID="g1"
    xmlns:smg="http://soamig.de/schema/SoamigSchema-0-3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://soamig.de/schema/SoamigSchema-0-3 soamig.xsd">
    <frontend.java.Operation  ID="v2" arity="0" operator="COMMENT" options="{}" />
    <frontend.java.Operation  ID="v3" arity="3"  operator="ERROR" options="{METHOD_BODY
        INNER_METHOD_BLOCK}" />
    <frontend.java.HasOperand FROM="v2"  TO="v3" FSEQ="1" TSEQ="2" />
</smg:SoamigGraph>
```

Listing 7: Eine Instanz des Graphen `SoamigGraph` mit zwei Knoten und einer Kante.

3.4 `Vertex`-Klassen

Alle spezifizierten `Vertex`-Klassen werden ebenfalls mit jeweils einem *element* und *complexType* angegeben und von dem vordefinierten Basistypen abgeleitet.

Die *element*-Definition erhält als *name* den Qualified Name der Klasse und wird in der *choice*-Definition des *complexType* der entsprechenden `Graph`-Klasse in der XSD aufgeführt.

```
<xsd:complexType name="VT_frontend.java.Operation">
    <xsd:complexContent>
        <xsd:extension base="smg:BT_Vertex">
            <xsd:attribute name="arity"    type="xsd:ST_INTEGER"/>
            <xsd:attribute name="operator" type="smg:ST_ENUM_frontend.java.TokenType"/>
            <xsd:attribute name="options"  type="smg:ST_SET"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Listing 8: Beispiel für eine spezialisierte Vertex-Klasse in XSD, die verschiede Attribute besitzt.

Die *complexType*-Definition erweitert den vordefinierten *complexType* „BT_Vertex" und erbt somit das *ID*-Attribut und wird um weitere im grUML-Schema spezifizierte Attribute ergänzt. Der *complexType* erhält als Präfix „VT_" für Vertex Type.

Ein Beispiel für eine `Vertex`-Klasse in XSD ist im Listing 8 und zwei Instanzen im Listing 7 zu sehen. Die *element*-Definition ist im Listing 6 in Zeile 7 aufgeführt.

3.5 `Edge`-Klassen

Alle Spezialisierungen der `Edge`-Klasse werden analog zu Spezialisierungen der `Vertex`-Klasse transformiert. Zwischen `Edge-, Aggregation`- und `Composition`-Klassen wird nicht weiter unterschieden. Der *complexType* erweitert "BT_Edge" und erbt somit die Attribute *TO*, *FROM*, *FSEQ* und *TSEQ*. Er erhält das Präfix „ET_" für Edge Type.

Ein Beispiel für eine `Edge`-Klasse in XSD ist im Listing 9 und eine Instanz im Listing 7 angegeben. Die *element*-Definition ist im Listing 6 in Zeile 8 aufgeführt.

```
1 <xsd:complexType name="ET_frontend.java.HasOperand">
2     <xsd:complexContent>
3         <xsd:extension base="smg:BT_Edge"/>
4     </xsd:complexContent>
5 </xsd:complexType>
```

Listing 9: Beispiel für eine spezialisierte `Edge`-Klasse in XSD.

3.6 Attribute

Attribute aus grUML können mittels XSD-Beschreibungselementen auf XML-Attribute abgebildet werden. Die Abbildung der Wertebereiche (Domains) wird in den Abschnitten 3.6.1, 3.6.2 und 3.6.3 beschrieben und auf einfache Datentypen von XSD zurückgeführt.

Bei dem Beispiel in Abb. 1 auf Seite 2 liegt die Klasse `frontend.java.Operation` mit einem eigenen Attribut und zwei geerbten Attributen vor. Die drei Attribute sind vom Typ `Set<frontend.java.Operation>` (eigenes Attribut), `integer` bzw. `frontend.java.Token-Type`. In grUML-Schemas werden Attribute als Attribute von (Assoziations-)Klassen modelliert. In XSD werden die Attribute als *attribute*-Definitionen von *complexType*-Definitionen angegeben. Das Ergebnis ist in Listing 8 in den Zeilen vier bis sechs zu sehen und zeigt wie eine Transformation der Attribute aussieht. Ein Beispiel für eine Instanz ist in Listing 7 dargestellt.

In grUML-Schemas existieren verschiedene Domains für Attribut-Wertebereiche, die in XSD abgebildet werden müssen. Die Domains sind: `Boolean`, `Integer`, `Long`, `Double`, `String`, `Enum`, `Set`, `List`, `Map` und `Record`.

Jede Domain besitzt einen Qualified Name. Der Qualified Name wird in XSD der *name* des Datentyps.

3.6.1 Primitive Domains

`Boolean`, `Integer`, `Long`, `Double` und `String` sind einfache Datentypen. Für sie gibt es entsprechende vordefinierte Datentypen in XSD:*boolean*, *integer*, *long*, *double* und *string*.

Von ihnen wurde jeweils ein eigener Typ abgeleitet mit den Namen (ST_≙Simple Type): ST_BOOLEAN, ST_INTEGER, ST_LONG, ST_DOUBLE und _ST_STRING.

Das Beispiel im Listing 8 verdeutlicht in Zeile vier, wie das Attribut „arity" mit dem Typ `integer` in XSD abgebildet wird. Eine Instanz der Klasse `frontend.java.Operation` in XML ist in Listing 7 zu sehen.

Primitive Datentypen werden immer dem Default-Package zugeordnet.

Boolean

Die Repräsentation von `boolean` wird nicht vom XSD-Datentypen *boolean* abgeleitet, sondern wird durch eine Enumeration ausgedrückt. Sie hat den Wertebereich {*f*, *t*}. Das resultiert aus der Überlegung, dass in Zukunft unter Umständen auch Null-Werte bei Wahrheitswerten unterstützt werden sollen.

String

Ein `string` in zur transformierten XSD validen XML-Dokumenten wird eingeleitet mit „"", dann folgt der zu speichernde String, und zum Schluss wird er mit einem „&qout;" abgeschlossen. Alle Zeichen, die sich nicht innerhalb des Wertebereichs 32 bis 127 befinden, werden mit „\uXXXX" dargestellt. „XXXX" steht hierbei für den entsprechenden Unicode-Zahlenwert des darzustellenden Zeichens. Der String „myDouble" wird im XML-Dokument durch „&qout;myDouble&qout;", die null-Referenz durch „n" dargestellt. Weitere Beispiele können dem Listing 16 auf Seite 23 entnommen werden. Der Knoten `JavaToken` eignet sich besonders gut zur Verdeutlichung, da er ein Attribut „text" vom Typ String enthält.

3.6.2 Enumerations

Der Datentyp `Enum` wird in XSD durch eine Beschränkung des Datentyps *string* repräsentiert. In die Einschränkung wird jeder einzelne Aufzählungswert mit einer *enumeration*-Definition deklariert. Zusätzlich zu den aufgezählten Konstanten wird die Konstante „n" deklariert, um eine null-Referenz auszudrücken. Der *simpleType* einer Enumeration erhält als *name* den Qualified Name der Enumeration und als Prefix „ST_ENUM_" für Simple Type Enumeration. Ein Beispiel für diese Deklaration ist in Listing 10 zu sehen.

Eine Instanz der Klasse `frontend.java.Operation` mit einer Beispielbelegung ist in Listing 11 zu sehen.

```
<xsd:complexType name="VT_frontend.java.Operation">
    <xsd:complexContent>
        <xsd:extension base="smg:BT_Vertex">
            <xsd:attribute name="operator" type="smg:ST_ENUM_frontend.java.TokenType"/>
            ...
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="ST_ENUM_frontend.java.TokenType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="COMMENT"/>
        ...
        <xsd:enumeration value="ANNOTATION"/>
        <xsd:enumeration value="ERROR"/>
        <xsd:enumeration value="ILLEGAL"/>
        <xsd:enumeration value="n"/>
    </xsd:restriction>
</xsd:simpleType>
```

Listing 10: Deklaration und Einbindung einer Enumeration namens `TokeType`.

```
<frontend.java.Operation operator="COMMENT" ... />
<frontend.java.Operation operator="ILLEGAL" ... />
```

Listing 11: Zwei Instanzen der Klasse `frontend.java.Operation` mit unterschiedlicher Attributbelegung.

3.6.3 Komplexe Domains

`Set`, `List`, `Map` und `Record` zählen zu den komplexen Domains. Da XML-Attribute nur einfache Datentypen enthalten können, können sie z.B. keine `RecordDomain` abbilden. Aus diesem Grund müssten die komplexen Domains durch XML-Elemente repräsentiert werden. Besonders aufwendig würde die Repräsentation mit geschachtelten komplexen Domains werden. Da eine Transformation in XSD-Elemente die XML-Instanzen mit XML-Elementen überfrachten würde, wird der Wert eines Attributs mit komplexer Domain durch eine einfache Stringrepräsentation dargestellt. Dabei geht die Angabe des genauen Typs in einer komplexen Domain verloren.

Das Beispiel aus Listing 12 zeigt die Verwendung eines komplexen Datentyps. Das Beispiel ist auf ein `Set` beschränkt. Das `Set` besteht aus Elementen des Typs `frontend.java.Options`. Die restlichen komplexen Domains sind analog umsetzbar und werden deshalb nicht einzeln aufgeführt.

Eine Instanz in XML sieht wie in Listing 13 aus.

Die komplexen Domains `Set`, `List` und `Map` erhalten die Namen „ST_SET", „ST_LIST" und „ST_MAP". Die komplexe Domain `Record` trägt den Qualified Name und erhält den Prefix „ST_RECORD_".

```
<xsd:complexType name="VT_frontend.java.Operation">
    <xsd:complexContent>
        <xsd:extension base="smg:BT_Vertex">
            <xsd:attribute name="options"  type="smg:ST_SET"/>
            ...
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="ST_ENUM_frontend.java.Options">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="CLASS_STATIC_INITIALIZER"/>
        <xsd:enumeration value="METHOD_BODY"/>
        <xsd:enumeration value="INNER_METHOD_BLOCK"/>
        ...
        <xsd:enumeration value="n"/>
    </xsd:restriction>
</xsd:simpleType>
```

Listing 12: Ein Beispiel für die Deklaration eines Attributs einer SetDomain.

```
<frontend.java.Operation options="{METHOD_BODY INNER_METHOD_BLOCK}" ... />
```

Listing 13: Beispiel für eine Instanz mit einem Attribute des Typs `Set`.

3.6.4 Stringrepräsentationen komplexer Domains

Die EBNF Regeln aus Listing 14 beschreiben Stringrepräsentationen komplexer Domains. Sie definieren die Stringrepräsentationen der verschiedenen Domains.

```
ListValue   ::= "n" | "[" [ Value { " " Value } ] "]";
SetValue    ::= "n" | "{" [ Value { " " Value } ] "}";
RecordValue ::= "n" | "("   Value { " " Value }   ")"; % alphabetische Reihenfolge der
    Bezeichner.
MapValue    ::= "n" | "{" [ Value "-" Value { " " Value "-" Value }] "}"
```

Listing 14: EBNF-Regeln für die Umsetzung der komplexen Datentypen `Set`, `List`, `Map` und `Record` als String.

Eine Liste (`List`) wird von „`[]`", eine Menge (`Set`) von „`{}`", eine Map (`Map`) von „`{}`" und ein Record (`Record`) von „`()`" als Klammernpaar umschlossen. Alle Werte werden innerhalb der Klammern aufgelistet und jeweils mit einem Leerzeichen getrennt. Bei der `Map` werden Schlüssel-Wert-Paare getrennt durch "-" aufgelistet. Die enthaltenen Paare werden mit einem Leerzeichen getrennt.

Jede `List`, `Set` oder `Map` kann leer sein oder eine beliebige Anzahl von Elementen enthalten. Eine `Record` muss genau so viele Werte enthalten, wie sein Urbild im grUML-Schema Komponenten hat. Die Elemente eines `Record` werden in alphabetischer Reihenfolge der Komponentenbezeichner angegeben. Der Wert aller komplexen Domainen kann auch null sein und wird in diesem Fall mit „`n`" repräsentiert.

Die folgenden Beispiele veranschaulichen diese Definition:

`List` mit Elementen des Typs `Integer` und den Werten: 0, 1, 1.

```
[ 0 1 1 ]
```

`Set` mit Elementen des Typs `Integer` und den Werten: 0, 1, 2.

```
{ 0 1 2 }
```

`Map` mit dem Key-Typ `String` und dem Value-Typ `Integer` und den Werten: („myDouble", 1), („myInt", 3).

```
{ &qout;myDouble&qout;-1 &qout;myInt&qout;-3 }
```

`Record` mit Komponenten „offset" und „length" jeweils vom Typ `Integer` und den Werten: offset = 33, length = 0.

```
( 0 33 )
```

4 Nutzung der Programme

Bei der Transformation eines Schemas von grUML nach XSD sind drei Schritte durchzuführen.

Ist ein Schema in grUML mit Hilfe eines UML-Modellierungsprogramm erstellt worden, so muss es in das Dateiformat XMI exportiert werden. Das Dateiformat ist bei IBM Rational Software Architect im Export-Dialog unter OTHERS →UML 2.1-XMI zu finden.

Als nächstes Programm wird die Klasse `Rsa2Tg` ausgeführt. Die generierte XMI-Datei dient dabei als Eingabe. Als Ausgabe werden mehrere Dateien erzeugt. Die Datei mit der Endung „*.gruml.tg" wird in dem weiteren Schritt benötigt.

Als letztes Programm wird die Klasse `SchemaGraph2XSD` ausgeführt. Sie erhält als Eingabe die Datei mit der Endung „*.gruml.tg". Außerdem müssen ein Prefix für den Namespace und der Name der zu generierenden XSD-Datei angegeben werden.

Die genauen Aufrufe der Klassen `Rsa2Tg` und `SchemaGraph2XSD` stehen im Hilfetext, der mit dem Parameter „–help" angezeigt wird.

4.1 Benutzung des Tools

Um aus einem Schema `beispielschema.tg` ein XSD zu generieren, verwendet man das Tool `SchemaGraph2XSD` auf folgende Weise:

```
java de.uni-koblenz.jgralab.utils.xml.SchemaGraph2XSD
    -s beispielschema.tg -n bsp -o beispielschema.xsd
```

Mit der Option `-s` (für Schema) wird auf die TG-Datei verwiesen, die das zu konvertierende Schema enthält. Liegt das Schema bereits als SchemaGraph vor, kann stattdessen die Option `-g` (für Graph) verwendet werden, um auf die TG-Datei zu verweisen, die den SchemaGraphen enthält. Die Option `-n` definiert das XML-Namespace-Prefix. Mit der Option `-o` (für Output) wird die Ausgabedatei spezifiziert. In dieser einfachen Form wird das komplette Schema zur Erzeugung der XSD-Datei verwendet.

Es ist möglich nur einen Teil des Schemas zur Generierung der XSD-Datei zu verwenden. Dazu dient die Option `-p`. Diese Option erwartet beliebig viele Include- oder Exclude-Patterns, die auch gemischt auftreten dürfen. Include-Patterns definieren, welche Teile des Schemas übernommen werden. Exclude-Patterns definieren analog dazu, welche Teile des Schemas nicht übernommen werden. Beide Patterns arbeiten auf den Qualified Names von Knoten- und Kantenklassen. Ist die Option `-p` gesetzt, muss mindestens ein Pattern angegeben werden. Ist das erste Pattern ein Exclude-Pattern, wird das komplette Schema zunächst zur Konvertierung markiert. Ist das erste Pattern ein Include-Pattern, wird das komplette Schema zunächst nicht zur Konvertierung markiert. Alle weiteren Patterns werden in der Reihenfolge, in der sie auftreten, abgearbeitet. Domains werden übernommen, soweit sie gebraucht werden. Ist die Option `-p` nicht gesetzt, werden alle Domains des Schemas übernommen, unabhängig davon, ob sie gebraucht werden, oder nicht[7].

Die Syntax von Patterns ist identisch mit der Syntax für reguläre Ausdrücke in Java. Include-Patterns beginnen immer mit einem "+" und Exclude-Patterns beginnen immer mit einem "-". Das Zeichen "." ist ein Wildcard. Wird der Punkt als Zeichen benötigt, muss ein "\" davor geschrieben werden. Jedes Pattern sollte mit Anführungszeichen umschlossen werden, damit es bei der Verwendung der Zeichensequenz ".*" nicht zu Problemen mit der Shell kommen kann, d.h. damit die Shell kein Globbing vornimmt.

Wenn beispielsweise im Soamig-Schema nur der Teil exportiert werden soll, der die Darstellung von Syntaxgraphen für Programmiersprachen abdeckt, kann folgendes Pattern verwendet werden, da sich die entsprechenden Metamodellteile ausschließlich im Paket `frontend` befinden:

```
-p "+frontend\..*"
```

Wenn gleichzeitig das Unterpaket "common" nicht mit übernommen werden soll, kann das mit diesen beiden Patterns erreicht werden. Hier wird das erstgenannte Include-Pattern durch das letztgenannte Exclude-Pattern weiter eingeschränkt:

```
-p "+frontend\..*" "-frontent\.common\..*"
```

Abstrakte Knotenklassen werden nur übernommen, wenn auch mindestens eine ihrer Spezialisierungen übernommen wird. Kantenklassen werden nicht übernommen, sobald mindestens eine inzidente Knotenklasse nicht übernommen wird[8]. Daher ist es mit dieser Konfiguration möglich, inkonsistente Schemas zu exportieren, da nicht geprüft wird, ob Spezialisierungen nicht zu übernehmender Klassen übernommen werden.

Im obigen Beispiel sind alle Knoten- und Kantenklassen des Pakets `frontend.java` Spezialisierungen von Klassen des Pakets `frontend.common`. Die Klassen aus `frontent.java` dürften somit eigentlich nicht übernommen werden. Um eine Prüfung einzuschalten, die dies feststellt, kann die Option `-x` aktiviert werden. Dann werden die Spezialisierungen nicht übernommener Klassen automatisch deselktiert. Diese Operation ist optional, da ihre Verwendung recht zeitaufwändig sein kann.

[7] Für gewöhnlich werden alle Domains eines Schemas an irgendeiner Stelle benutzt.

[8] Abstrakte Klassen tauchen in der XSD-Datei zwar nicht auf, aber für die Entscheidung ob Kantenklassen übernommen werden sollen oder nicht, ist die Markierung abstrakter Klassen wichtig.

Um festzustellen, welche Klassen übernommen wurden und welche nicht, muss nicht auf die XSD-Datei zurückgegriffen werden. Mit Hilfe der Option `-d` kann eine Datei angelegt werden, die die Namen aller Klassen des Schemas enthält. In ihr sind alle übernommenen Klassen mit dem Präfix `IN:` und alle nicht übernommenen Klassen mit dem Präfix `EX:` markiert.

Komplettes Beispiel

Der Aufruf

```
java de.uni-koblenz.jgralab.utils.xml.SchemaGraph2XSD
    -s soamigschema.tg -n smg -o soamigschema.xsd
    -p "+frontend\..*" "-frontent\.common\.Token" -x -d "debug.txt"
```

erstellt eine XSD-Datei namens `soamigschema.xsd` aus der Datei `soamigschema.tg` mit dem XML-Namespace-Prefix `smg`. Darin enthalten sind alle Klassen des Pakets `frontend` außer der Knotenklasse `frontent.common.Token`. Die Option `-x` bewirkt, dass alle Spezialisierungen von `frontend.common.Token` nicht übernommen werden. Darüber hinaus werden alle Kantenklassen nicht übernommen, die zu `common.Token` oder einer der Spezialisierungen inzident sind.

A Metamodell von grUML-Schemas

Das Metamodell von grUML-Schemas ist in Abb. 3 aus Vollständigkeitsgründen aufgeführt. Das Metamodell von Domains steht in Abb. 4 zur Verfügung.

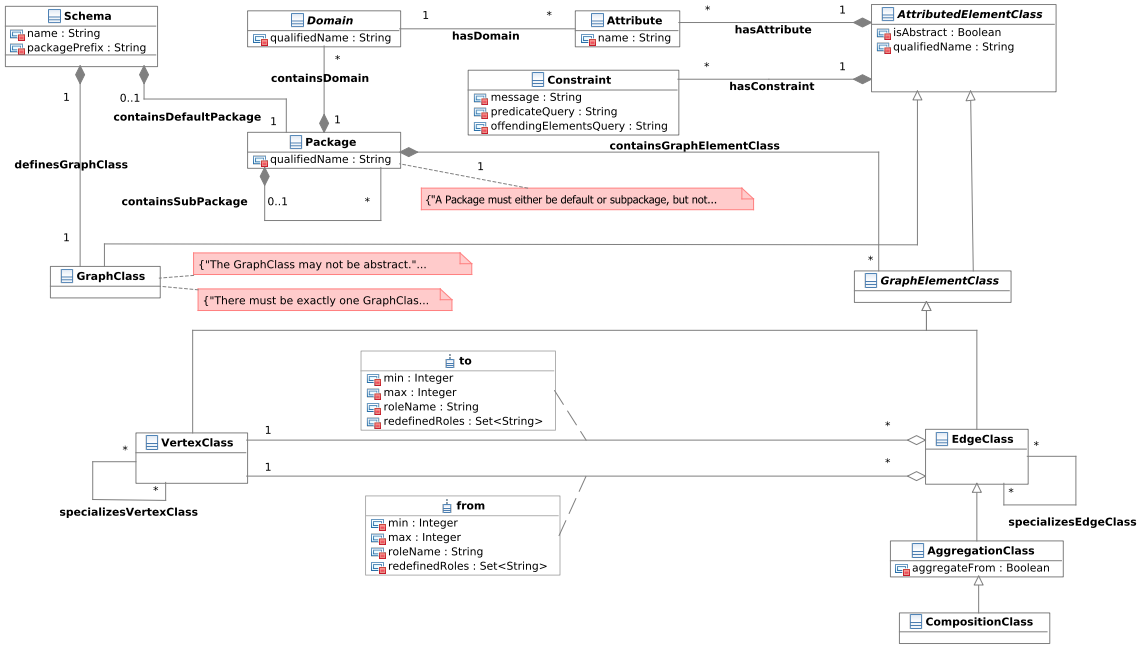

Abbildung 3: Metamodell zu grUML-Schemas.

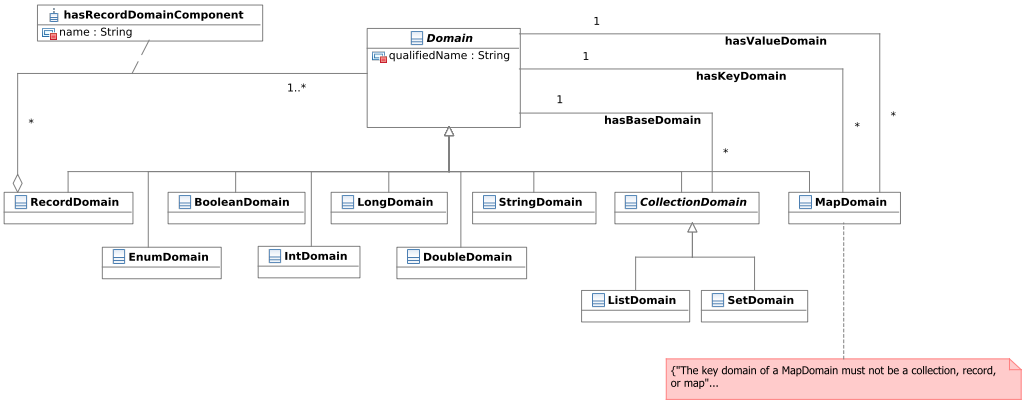

Abbildung 4: Metamodell zu grUML-Domains.

B Einschränkungen der Transformation von grUML nach XSD

Der folgende Abschnitt zählt alle Einschränkungen auf, die bei der Transformation von grUML nach XSD zu beachten sind.

`Packages` werden nicht explizit aufgeführt, sondern implizit durch den Qualified Name ausgedrückt, den ein jedes `AttributedElement` enthält.

`Constraints` werden nicht in XSD abgebildet.

`from`- und `to`-Kanten werden nicht abgebildet, somit ihre Multiplizitäten, Rollennamen, etc. ebenfalls nicht.

`Aggregation` und `Composition` werden in XSD durch einen `Edge`-Typ abgebildet.

Mehrfachvererbung wird in XSD nicht unterstützt, sondern nur die einfache Vererbung. Aus dem Grund kann die Vererbung und im Speziellen die Mehrfachvererbung in grUML-Schemas nicht durch gegebene Sprachmittel in XSD ausgedrückt werden. Die Vererbung wird indirekt durch das erneute Definieren der geerbten Attribute realisiert.

C Vollständig transformiertes Beispielschema

Das Listing 15 listet den vollständig transformierten XSD-Beispielcode des Beispielschemas aus Abb. 1 auf Seite 2 auf, der einen Ausschnitt des SOAMIG-Java-Schemas zeigt. Das Listing 7 auf Seite 11 enthält eine Beispiel Instanz in XML, die valide gegenüber dem generierten XSD ist.

```
1  <?xml version="1.0" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:smg="http://soamig.de/
       schema/SoamigSchema-0-3" targetNamespace="http://soamig.de/schema/SoamigSchema-0-3">
3      <!--Default types-->
4      <xsd:complexType name="BT_AttributedElement" abstract="true"/>
5      <xsd:complexType name="BT_Graph" abstract="true">
6          <xsd:complexContent>
7              <xsd:extension base="smg:BT_AttributedElement">
8                  <xsd:attribute name="ID" type="xsd:ID" use="required"/>
9              </xsd:extension>
10         </xsd:complexContent>
11     </xsd:complexType>
12     <xsd:complexType name="BT_Vertex" abstract="true">
13         <xsd:complexContent>
14             <xsd:extension base="smg:BT_AttributedElement">
15                 <xsd:attribute name="ID" type="xsd:ID" use="required"/>
16             </xsd:extension>
17         </xsd:complexContent>
18     </xsd:complexType>
19     <xsd:complexType name="BT_Edge" abstract="true">
20         <xsd:complexContent>
21             <xsd:extension base="smg:BT_AttributedElement">
22                 <xsd:attribute name="FROM" type="xsd:IDREF" use="required"/>
23                 <xsd:attribute name="TO" type="xsd:IDREF" use="required"/>
24                <xsd:attribute name="FSEQ" type="smg:ST_INTEGER"/>
25                <xsd:attribute name="TSEQ" type="smg:ST_INTEGER"/>
```

```
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="ST_BOOLEAN">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="t"/>
            <xsd:enumeration value="f"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ST_STRING">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="&quot;.*&quot;"/>
            <xsd:pattern value="n"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ST_INTEGER">
        <xsd:restriction base="xsd:integer"/>
    </xsd:simpleType>
    <xsd:simpleType name="ST_LONG">
        <xsd:restriction base="xsd:long"/>
    </xsd:simpleType>
    <xsd:simpleType name="ST_DOUBLE">
        <xsd:restriction base="xsd:double"/>
    </xsd:simpleType>
    <xsd:simpleType name="ST_LIST">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\[.*\]"/>
            <xsd:pattern value="n"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ST_SET">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\{.*\}"/>
            <xsd:pattern value="n"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ST_MAP">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\{.*\}"/>
            <xsd:pattern value="n"/>
        </xsd:restriction>
    </xsd:simpleType>

    <!--Graph-type-->
    <xsd:element name="SoamigGraph" type="smg:GT_SoamigGraph"/>

    <xsd:complexType name="GT_SoamigGraph">
        <xsd:complexContent>
            <xsd:extension base="smg:BT_Graph">
                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                    <xsd:element name="frontend.java.Operation" type="smg:VT_frontend.java
                        .Operation"/>
                    <xsd:element name="frontend.java.HasOperand" type="smg:ET_frontend.
                        java.HasOperand"/>
                </xsd:choice>
```

```
79              </xsd:extension>
80          </xsd:complexContent>
81      </xsd:complexType>
82
83      <!--Vertex-types-->
84      <xsd:complexType name="VT_frontend.java.Operation">
85          <xsd:complexContent>
86              <xsd:extension base="smg:BT_Vertex">
87                  <xsd:attribute name="arity" type="smg:ST_INTEGER"/>
88                  <xsd:attribute name="operator" type="smg:ST_ENUM_frontend.java.TokenType"/
                        >
89                  <xsd:attribute name="options" type="smg:ST_SET"/>
90              </xsd:extension>
91          </xsd:complexContent>
92      </xsd:complexType>
93
94          <!--Edge-types-->
95      <xsd:complexType name="ET_frontend.java.HasOperand">
96          <xsd:complexContent>
97              <xsd:extension base="smg:BT_Edge"/>
98          </xsd:complexContent>
99      </xsd:complexType>
100
101     <!--Enumeration-types-->
102     <xsd:simpleType name="ST_ENUM_frontend.java.TokenType">
103         <xsd:restriction base="xsd:string">
104             <xsd:enumeration value="COMMENT"/>
105             <xsd:enumeration value="ANNOTATION"/>
106             <xsd:enumeration value="ERROR"/>
107             <xsd:enumeration value="ILLEGAL"/>
108            ...
109             <xsd:enumeration value="n"/>
110         </xsd:restriction>
111     </xsd:simpleType>
112     <xsd:simpleType name="ST_ENUM_frontend.java.Options">
113         <xsd:restriction base="xsd:string">
114             <xsd:enumeration value="CLASS_STATIC_INITIALIZER"/>
115             <xsd:enumeration value="METHOD_BODY"/>
116             <xsd:enumeration value="INNER_METHOD_BLOCK"/>
117            ...
118             <xsd:enumeration value="n"/>
119         </xsd:restriction>
120     </xsd:simpleType>
121 </xsd:schema>
```

Listing 15: Vollständig transformierter XSD-Beispielcode für einen Ausschnitt des SOAMIG-Java-Schemas.

D Vollständiger Beispielgraph für Inzidenzsequenzen

Der Beispielgraph im Listing 16 und in der Abb. 5 ist ein Graph für den Java-Ausdruck:

```
myDouble = myBool ? 18.4 : 11.4;
```

Es wird davon ausgegangen, dass der Ausdruck in der ersten Zeile und Spalte der Datei beginnt, was sich in den Positionsangaben in den `JavaTokens` zeigt.

```
<?xml version="1.0" encoding="UTF-8"?>
<smg:SoamigGraph
    xmlns:smg="http://soamig.de/schema/SoamigSchema-0-3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://soamig.de/schema/SoamigSchema-0-3 soamig.xsd"
    ID="g82f0c844-4077dbcc-dc06298-cd53151b">

    <frontend.java.SourceFile  ID="v1" convertedName="n" encoding="n"
                               originalName="n" processedName="n"/>
    <frontend.common.TokenList ID="v2"/>
    <frontend.java.Access      ID="v3"  name="&quot;myDouble&quot;"
                               operator="VAR" options="n"/>
    <frontend.java.JavaToken   ID="v4"  pos="(1 1 8 1 9 1)"
                               text="&quot;myDouble&quot;" type="VAR" visibility="n"/>
    <frontend.java.Access      ID="v5"  name="&quot;myBool&quot;"
                               operator="VAR" options="n"/>
    <frontend.java.JavaToken   ID="v6"  pos="(12 1 6 12 18 1)"
                               text="&quot;myBool&quot;" type="VAR" visibility="n"/>
    <frontend.java.Literal     ID="v7"  operator="DOUBLE"
                               options="n" value="&quot;18.4&quot;"/>
    <frontend.java.JavaToken   ID="v8"  pos="(21 1 4 21 25 1)"
                               text="&quot;18.4&quot;" type="DOUBLE" visibility="n"/>
    <frontend.java.Literal     ID="v9"  operator="DOUBLE"
                               options="n" value="&quot;11.4&quot;"/>
    <frontend.java.JavaToken   ID="v10" pos="(28 1 4 28 32 1)"
                               text="&quot;11.4&quot;" type="DOUBLE" visibility="n"/>
    <frontend.java.Operation   ID="v11" arity="3" operator="COND" options="n"/>
    <frontend.java.JavaToken   ID="v12" pos="(19 1 1 19 20 1)"
                               text="&quot;?&quot;" type="COND" visibility="n"/>
    <frontend.java.Operation   ID="v13" arity="2" operator="ASSIGN" options="n"/>
    <frontend.java.JavaToken   ID="v14" pos="(10 1 1 10 11 1)"
                               text="&quot;=&quot;" type="ASSIGN" visibility="n"/>
    <frontend.java.JavaToken   ID="v15" pos="(26 1 1 26 27 1)"
                               text="&quot;:&quot;" type="COLON" visibility="n"/>
    <frontend.java.JavaToken   ID="v16" pos="(32 1 1 32 33 1)"
                               text="&quot;;&quot;" type="SEMICOLON" visibility="n"/>
    <frontend.java.JavaToken   ID="v17" pos="(33 1 0 33 33 1)"
                               text="&quot;n&quot;" type="EOF" visibility="n"/>

    <frontend.common.HasTokenList  FROM="v1"  FSEQ="1"  TO="v2"  TSEQ="1"/>
    <frontend.java.HasOperand      FROM="v11" FSEQ="1"  TO="v5"  TSEQ="1"/>
    <frontend.java.HasOperand      FROM="v11" FSEQ="2"  TO="v7"  TSEQ="1"/>
    <frontend.java.HasOperand      FROM="v11" FSEQ="3"  TO="v9"  TSEQ="1"/>
    <frontend.java.HasOperand      FROM="v13" FSEQ="1"  TO="v3"  TSEQ="1"/>
    <frontend.java.HasOperand      FROM="v13" FSEQ="2"  TO="v11" TSEQ="4"/>
    <frontend.java.BeginsAt        FROM="v13" FSEQ="3"  TO="v4"  TSEQ="1"/>
    <frontend.java.EndsAt          FROM="v13" FSEQ="4"  TO="v16" TSEQ="1"/>
    <frontend.java.HasToken        FROM="v13" FSEQ="5"  TO="v14" TSEQ="1"/>
    <frontend.java.BeginsAt        FROM="v3"  FSEQ="2"  TO="v4"  TSEQ="2"/>
    <frontend.java.EndsAt          FROM="v3"  FSEQ="3"  TO="v4"  TSEQ="3"/>
    <frontend.java.HasToken        FROM="v3"  FSEQ="4"  TO="v4"  TSEQ="4"/>
```

```
52      <frontend.java.BeginsAt        FROM="v11" FSEQ="5"  TO="v6"  TSEQ="1"/>
53      <frontend.java.EndsAt          FROM="v11" FSEQ="6"  TO="v16" TSEQ="2"/>
54      <frontend.java.HasToken        FROM="v11" FSEQ="7"  TO="v12" TSEQ="1"/>
55      <frontend.java.BeginsAt        FROM="v5"  FSEQ="2"  TO="v6"  TSEQ="2"/>
56      <frontend.java.EndsAt          FROM="v5"  FSEQ="3"  TO="v6"  TSEQ="3"/>
57      <frontend.java.HasToken        FROM="v5"  FSEQ="4"  TO="v4"  TSEQ="5"/>
58      <frontend.java.BeginsAt        FROM="v7"  FSEQ="2"  TO="v8"  TSEQ="1"/>
59      <frontend.java.EndsAt          FROM="v7"  FSEQ="3"  TO="v8"  TSEQ="2"/>
60      <frontend.java.HasToken        FROM="v7"  FSEQ="4"  TO="v8"  TSEQ="3"/>
61      <frontend.java.BeginsAt        FROM="v9"  FSEQ="2"  TO="v10" TSEQ="1"/>
62      <frontend.java.EndsAt          FROM="v9"  FSEQ="3"  TO="v10" TSEQ="2"/>
63      <frontend.java.HasToken        FROM="v9"  FSEQ="4"  TO="v10" TSEQ="3"/>
64      <frontend.common.ContainsToken FROM="v2"  FSEQ="2"  TO="v4"  TSEQ="6"/>
65      <frontend.common.ContainsToken FROM="v2"  FSEQ="3"  TO="v14" TSEQ="2"/>
66      <frontend.common.ContainsToken FROM="v2"  FSEQ="4"  TO="v6"  TSEQ="4"/>
67      <frontend.common.ContainsToken FROM="v2"  FSEQ="5"  TO="v12" TSEQ="2"/>
68      <frontend.common.ContainsToken FROM="v2"  FSEQ="6"  TO="v8"  TSEQ="4"/>
69      <frontend.common.ContainsToken FROM="v2"  FSEQ="7"  TO="v15" TSEQ="1"/>
70      <frontend.common.ContainsToken FROM="v2"  FSEQ="8"  TO="v10" TSEQ="4"/>
71      <frontend.common.ContainsToken FROM="v2"  FSEQ="9"  TO="v16" TSEQ="3"/>
72      <frontend.common.ContainsToken FROM="v2"  FSEQ="10" TO="v17" TSEQ="1"/>
73  </smg:SoamigGraph>
```

Listing 16: Beispielgraph in XML eines Javaausdrucks.

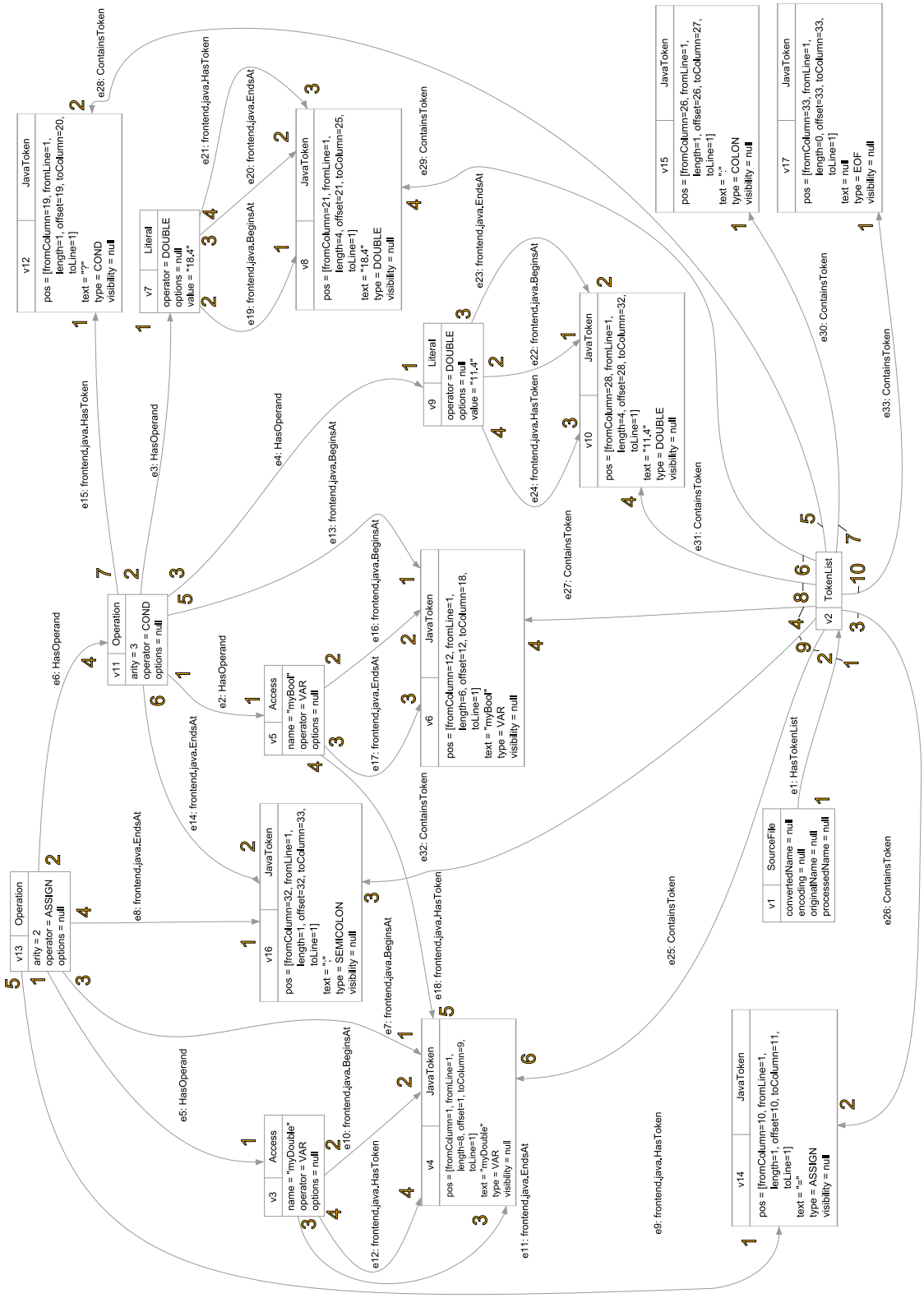

Abbildung 5: Visualisierter Beispielgraph eines Java-Ausdrucks: `myDouble = myBool ? 18.4 : 11.4;`

Literatur

[BHR+09] BILDHAUER, Daniel ; HORN, Tassilo ; RIEDIGER, Volker ; SCHWARZ, Hannes ; STRAUSS, Sascha: grUML - A UML based modelling language for TGraphs. (2009)

[ERW08] EBERT, J. ; RIEDIGER, V. ; WINTER, A.: Graph Technology in Reverse Engineering, The TGraph Approach. In: GIMNICH, R. (Hrsg.) ; KAISER, U. (Hrsg.) ; QUANTE, J. (Hrsg.) ; WINTER, A. (Hrsg.): *10th Workshop Software Reengineering (WSR 2008)* Bd. 126, GI, 2008 (GI Lecture Notes in Informatics). – ISBN 978–3–88579–220–8, S. 67–81

[EWD+96] EBERT, Jürgen ; WINTER, Andreas ; DAHM, Peter ; FRANZKE, Angelika ; SÜTTENBACH, Roger: Graph Based Modeling and Implementation with EER/GRAL. In: THALHEIM, B. (Hrsg.): *ER'96 - Proceedings of the 15th International Conference on Conceptual Modeling*. Berlin : Springer Verlag, 1996, S. 163–178

[HSESW06] HOLT, Richard C. ; SCHÜRR, Andy ; ELLIOTT SIM, Susan ; WINTER, Andreas: GXL: A graph-based standard exchange format for reengineering. In: 60 (2006), Nr. 2, S. 149–170

[W3C04] W3C: *XML Schema Part 0: Primer Second Edition*. `http://www.w3.org/TR/xmlschema-0/`. Version: October 2004

[W3C09a] W3C: *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. `http://www.w3.org/TR/xmlschema11-1/`. Version: April 2009

[W3C09b] W3C: *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. `http://www.w3.org/TR/xmlschema11-2/`. Version: April 2009

Bisher erschienen

Arbeitsberichte aus dem Fachbereich Informatik
(http://www.uni-koblenz.de/fb4/publikationen/arbeitsberichte)

Eckhard Großmann, Sascha Strauß, Tassilo Horn, Volker Riediger, Abbildung von grUML nach XSD soamig, Arbeitsberichte aus dem Fachbereich Informatik 15/2009

Kerstin Falkowski, Jürgen Ebert, The STOR Component System Interim Report, Arbeitsberichte aus dem Fachbereicht Informatik 14/2009

Sebastian Magnus, Markus Maron, An Empirical Study to Evaluate the Location of Advertisement Panels by Using a Mobile Marketing Tool, Arbeitsberichte aus dem Fachbereich Informatik 13/2009

Sebastian Magnus, Markus Maron, Konzept einer Public Key Infrastruktur in iCity, Arbeitsberichte aus dem Fachbereich Informatik 12/2009

Sebastian Magnus, Markus Maron, A Public Key Infrastructure in Ambient Information and Transaction Systems, Arbeitsberichte aus dem Fachbereich Informatik 11/2009

Ammar Mohammed, Ulrich Furbach, Multi-agent systems: Modeling and Virification using Hybrid Automata, Arbeitsberichte aus dem Fachbereich Informatik 10/2009

Andreas Sprotte, Performance Measurement auf der Basis von Kennzahlen aus betrieblichen Anwendungssystemen: Entwurf eines kennzahlengestützten Informationssystems für einen Logistikdienstleister, Arbeitsberichte aus dem Fachbereich Informatik 9/2009

Gwendolin Garbe, Tobias Hausen, Process Commodities: Entwicklung eines Reifegradmodells als Basis für Outsourcingentscheidungen, Arbeitsberichte aus dem Fachbereich Informatik 8/2009

Petra Schubert et. al., Open-Source-Software für das Enterprise Resource Planning, Arbeitsberichte aus dem Fachbereich Informatik 7/2009

Ammar Mohammed, Frieder Stolzenburg, Using Constraint Logic Programming for Modeling and Verifying Hierarchical Hybrid Automata, Arbeitsberichte aus dem Fachbereich Informatik 6/2009

Tobias Kippert, Anastasia Meletiadou, Rüdiger Grimm, Entwurf eines Common Criteria-Schutzprofils für Router zur Abwehr von Online-Überwachung, Arbeitsberichte aus dem Fachbereich Informatik 5/2009

Hannes Schwarz, Jürgen Ebert, Andreas Winter, Graph-based Traceability – A Comprehensive Approach. Arbeitsberichte aus dem Fachbereich Informatik 4/2009

Anastasia Meletiadou, Simone Müller, Rüdiger Grimm, Anforderungsanalyse für Risk-Management-Informationssysteme (RMIS), Arbeitsberichte aus dem Fachbereich Informatik 3/2009

Ansgar Scherp, Thomas Franz, Carsten Saathoff, Steffen Staab, A Model of Events based on a Foundational Ontology, Arbeitsberichte aus dem Fachbereich Informatik 2/2009

Frank Bohdanovicz, Harald Dickel, Christoph Steigner, Avoidance of Routing Loops, Arbeitsberichte aus dem Fachbereich Informatik 1/2009

Stefan Ameling, Stephan Wirth, Dietrich Paulus, Methods for Polyp Detection in Colonoscopy Videos: A Review, Arbeitsberichte aus dem Fachbereich Informatik 14/2008

Tassilo Horn, Jürgen Ebert, Ein Referenzschema für die Sprachen der IEC 61131-3, Arbeitsberichte aus dem Fachbereich Informatik 13/2008

Thomas Franz, Ansgar Scherp, Steffen Staab, Does a Semantic Web Facilitate Your Daily Tasks?, Arbeitsberichte aus dem Fachbereich Informatik 12/2008

Norbert Frick, Künftige Anfordeungen an ERP-Systeme: Deutsche Anbieter im Fokus, Arbeitsberichte aus dem Fachbereicht Informatik 11/2008

Jürgen Ebert, Rüdiger Grimm, Alexander Hug, Lehramtsbezogene Bachelor- und Masterstudiengänge im Fach Informatik an der Universität Koblenz-Landau, Campus Koblenz, Arbeitsberichte aus dem Fachbereich Informatik 10/2008

Mario Schaarschmidt, Harald von Kortzfleisch, Social Networking Platforms as Creativity Fostering Systems: Research Model and Exploratory Study, Arbeitsberichte aus dem Fachbereich Informatik 9/2008

Bernhard Schueler, Sergej Sizov, Steffen Staab, Querying for Meta Knowledge, Arbeitsberichte aus dem Fachbereich Informatik 8/2008

Stefan Stein, Entwicklung einer Architektur für komplexe kontextbezogene Dienste im mobilen Umfeld, Arbeitsberichte aus dem Fachbereich Informatik 7/2008

Matthias Bohnen, Lina Brühl, Sebastian Bzdak, RoboCup 2008 Mixed Reality League Team Description, Arbeitsberichte aus dem Fachbereich Informatik 6/2008

Bernhard Beckert, Reiner Hähnle, Tests and Proofs: Papers Presented at the Second International Conference, TAP 2008, Prato, Italy, April 2008, Arbeitsberichte aus dem Fachbereich Informatik 5/2008

Klaas Dellschaft, Steffen Staab, Unterstützung und Dokumentation kollaborativer Entwurfs- und Entscheidungsprozesse, Arbeitsberichte aus dem Fachbereich Informatik 4/2008

Rüdiger Grimm: IT-Sicherheitsmodelle, Arbeitsberichte aus dem Fachbereich Informatik 3/2008

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik 2/2008

Markus Maron, Kevin Read, Michael Schulze: CAMPUS NEWS – Artificial Intelligence Methods Combined for an Intelligent Information Network, Arbeitsberichte aus dem Fachbereich Informatik 1/2008

Lutz Priese,Frank Schmitt, Patrick Sturm, Haojun Wang: BMBF-Verbundprojekt 3D-RETISEG Abschlussbericht des Labors Bilderkennen der Universität Koblenz-Landau, Arbeitsberichte aus dem Fachbereich Informatik 26/2007

Stephan Philippi, Alexander Pinl: Proceedings 14. Workshop 20.-21. September 2007 Algorithmen und Werkzeuge für Petrinetze, Arbeitsberichte aus dem Fachbereich Informatik 25/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS – an Intelligent Bluetooth-based Mobile Information Network, Arbeitsberichte aus dem Fachbereich Informatik 24/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS - an Information Network for Pervasive Universities, Arbeitsberichte aus dem Fachbereich Informatik 23/2007

Lutz Priese: Finite Automata on Unranked and Unordered DAGs Extented Version, Arbeitsberichte aus dem Fachbereich Informatik 22/2007

Mario Schaarschmidt, Harald F.O. von Kortzfleisch: Modularität als alternative Technologie- und Innovationsstrategie, Arbeitsberichte aus dem Fachbereich Informatik 21/2007

Kurt Lautenbach, Alexander Pinl: Probability Propagation Nets, Arbeitsberichte aus dem Fachbereich Informatik 20/2007

Rüdiger Grimm, Farid Mehr, Anastasia Meletiadou, Daniel Pähler, Ilka Uerz: SOA-Security, Arbeitsberichte aus dem Fachbereich Informatik 19/2007

Christoph Wernhard: Tableaux Between Proving, Projection and Compilation, Arbeitsberichte aus dem Fachbereich Informatik 18/2007

Ulrich Furbach, Claudia Obermaier: Knowledge Compilation for Description Logics, Arbeitsberichte aus dem Fachbereich Informatik 17/2007

Fernando Silva Parreiras, Steffen Staab, Andreas Winter: TwoUse: Integrating UML Models and OWL Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 16/2007

Rüdiger Grimm, Anastasia Meletiadou: Rollenbasierte Zugriffskontrolle (RBAC) im Gesundheitswesen, Arbeitsberichte aud dem Fachbereich Informatik 15/2007

Ulrich Furbach, Jan Murray, Falk Schmidsberger, Frieder Stolzenburg: Hybrid Multiagent Systems with Timed Synchronization-Specification and Model Checking, Arbeitsberichte aus dem Fachbereich Informatik 14/2007

Björn Pelzer, Christoph Wernhard: System Description:“E-KRHyper“, Arbeitsberichte aus dem Fachbereich Informatik, 13/2007

Ulrich Furbach, Peter Baumgartner, Björn Pelzer: Hyper Tableaux with Equality, Arbeitsberichte aus dem Fachbereich Informatik, 12/2007

Ulrich Furbach, Markus Maron, Kevin Read: Location based Informationsystems, Arbeitsberichte aus dem Fachbereich Informatik, 11/2007

Philipp Schaer, Marco Thum: State-of-the-Art: Interaktion in erweiterten Realitäten, Arbeitsberichte aus dem Fachbereich Informatik, 10/2007

Ulrich Furbach, Claudia Obermaier: Applications of Automated Reasoning, Arbeitsberichte aus dem Fachbereich Informatik, 9/2007

Jürgen Ebert, Kerstin Falkowski: A First Proposal for an Overall Structure of an Enhanced Reality Framework, Arbeitsberichte aus dem Fachbereich Informatik, 8/2007

Lutz Priese, Frank Schmitt, Paul Lemke: Automatische See-Through Kalibrierung, Arbeitsberichte aus dem Fachbereich Informatik, 7/2007

Rüdiger Grimm, Robert Krimmer, Nils Meißner, Kai Reinhard, Melanie Volkamer, Marcel Weinand, Jörg Helbach: Security Requirements for Non-political Internet Voting, Arbeitsberichte aus dem Fachbereich Informatik, 6/2007

Daniel Bildhauer, Volker Riediger, Hannes Schwarz, Sascha Strauß, „grUML – Eine UML-basierte Modellierungssprache für T-Graphen“, Arbeitsberichte aus dem Fachbereich Informatik, 5/2007

Richard Arndt, Steffen Staab, Raphaël Troncy, Lynda Hardman: Adding Formal Semantics to MPEG-7: Designing a Well Founded Multimedia Ontology for the Web, Arbeitsberichte aus dem Fachbereich Informatik, 4/2007

Simon Schenk, Steffen Staab: Networked RDF Graphs, Arbeitsberichte aus dem Fachbereich Informatik, 3/2007

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik, 2/2007

Anastasia Meletiadou, J. Felix Hampe: Begriffsbestimmung und erwartete Trends im IT-Risk-Management, Arbeitsberichte aus dem Fachbereich Informatik, 1/2007

„Gelbe Reihe“
(http://www.uni-koblenz.de/fb4/publikationen/gelbereihe)

Lutz Priese: Some Examples of Semi-rational and Non-semi-rational DAG Languages. Extended Version, Fachberichte Informatik 3-2006

Kurt Lautenbach, Stephan Philippi, and Alexander Pinl: Bayesian Networks and Petri Nets, Fachberichte Informatik 2-2006

Rainer Gimnich and Andreas Winter: Workshop Software-Reengineering und Services, Fachberichte Informatik 1-2006

Kurt Lautenbach and Alexander Pinl: Probability Propagation in Petri Nets, Fachberichte Informatik 16-2005

Rainer Gimnich, Uwe Kaiser, and Andreas Winter: 2. Workshop "Reengineering Prozesse" – Software Migration, Fachberichte Informatik 15-2005

Jan Murray, Frieder Stolzenburg, and Toshiaki Arai: Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification, Fachberichte Informatik 14-2005

Reinhold Letz: FTP 2005 – Fifth International Workshop on First-Order Theorem Proving, Fachberichte Informatik 13-2005

Bernhard Beckert: TABLEAUX 2005 – Position Papers and Tutorial Descriptions, Fachberichte Informatik 12-2005

Dietrich Paulus and Detlev Droege: Mixed-reality as a challenge to image understanding and artificial intelligence, Fachberichte Informatik 11-2005

Jürgen Sauer: 19. Workshop Planen, Scheduling und Konfigurieren / Entwerfen, Fachberichte Informatik 10-2005

Pascal Hitzler, Carsten Lutz, and Gerd Stumme: Foundational Aspects of Ontologies, Fachberichte Informatik 9-2005

Joachim Baumeister and Dietmar Seipel: Knowledge Engineering and Software Engineering, Fachberichte Informatik 8-2005

Benno Stein and Sven Meier zu Eißen: Proceedings of the Second International Workshop on Text-Based Information Retrieval, Fachberichte Informatik 7-2005

Andreas Winter and Jürgen Ebert: Metamodel-driven Service Interoperability, Fachberichte Informatik 6-2005

Joschka Boedecker, Norbert Michael Mayer, Masaki Ogino, Rodrigo da Silva Guerra, Masaaki Kikuchi, and Minoru Asada: Getting closer: How Simulation and Humanoid League can benefit from each other, Fachberichte Informatik 5-2005

Torsten Gipp and Jürgen Ebert: Web Engineering does profit from a Functional Approach, Fachberichte Informatik 4-2005

Oliver Obst, Anita Maas, and Joschka Boedecker: HTN Planning for Flexible Coordination Of Multiagent Team Behavior, Fachberichte Informatik 3-2005

Andreas von Hessling, Thomas Kleemann, and Alex Sinner: Semantic User Profiles and their Applications in a Mobile Environment, Fachberichte Informatik 2-2005

Heni Ben Amor and Achim Rettinger: Intelligent Exploration for Genetic Algorithms – Using Self-Organizing Maps in Evolutionary Computation, Fachberichte Informatik 1-2005