Entwicklung eines Matching- und Mappingverfahrens zur Verbesserung der XML-Schemaevolution

Masterarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik

vorgelegt von:	Jan Deffke
Matrikelnummer:	8200361
geboren am:	01.01.1989 in Rostock
Erstgutachter:	PD Dr. -Ing habil. Meike Klettke
Zweitgutachter:	Prof. Dr. Kurt Sandkuhl
Betreuer:	Dipl. -Inf. Thomas Nösinger
Abgabedatum:	27.08.2013

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 24. August 2013

Zusammenfassung

Im Bereich der schemabasierten Matching- und Mappingverfahren existieren eine Vielzahl von Algorithmen und Umsetzungen in die Praxis, beispielsweise zum Finden von Mappings zwischen XML-Schemata. Die Verwertbarkeit dieser Ansätze zur Verbesserung der XML Schema Evolution ist ein offenes Forschungsfeld.
In dieser Masterarbeit wird ein stand-alone Tool entwickelt, welches die Schemaevolution auf Basis von automatisch generierten Mappings durchführt. Dazu wird eine Benutzeroberfläche zum Hinzufügen und zum Ändern der Mappings angeboten. Aus den Mappings können die entsprechenden Anpassungsschritte generiert und in *Evolution Language for XML-Schema* (ELaX) ausgedrückt werden, um das eine XML-Schema in das andere zu transformieren. Die Evaluation der Ergebnisse ergibt, dass die Erzeugung der Evolution Language for XML-Schema (ELaX)-Ausdrücke im stand-alone Tool annähernd an die Leistung des universitären Tools *Conceptual Design and Evolution for XML-Schema* (CodeX) herankommt. Dadurch kann das Ergebnis als Referenz benutzt werden, um die Schemaevolution zu verbessern, weshalb das stand-alone Tool in CodeX integriert wird.

Abstract

There are a variety of schema-based matching and mapping algorithms implemented in different tools, e.g. to discover mappings between XML schemas. The ability of these approaches to improving XML schema evolution is an open research field.
In this thesis a stand-alone tool is developed which performs the schema evolution on the basis of automatically generated mappings. To do so, a user interface for adding and changing the mappings will be offered. The relevant adaptation steps can be generated by the mappings and expressed in *Evolution Language for XML-Schema* (ELaX) to transform one XML schema to the other. The results of the evaluation indicates that the generated ELaX expressions by the stand-alone tool, approaching nearly to the performance of the university tool *Conceptual Design and Evolution for XML-Schema* (CodeX). Thereby, the result can be used as reference to improve the schema evolution, so the stand-alone tool is integrated into CodeX.

Schlüsselwörter

XML Schema, Schema-Evolution, Mapping, Matching, Evolutionssprache

Keywords

XML Schema, schema-evolution, mapping, matching, evolution language

Inhaltsverzeichnis

Abkürzungsverzeichnis

BSM Base Similarity Matcher . . . 26
CodeX Conceptual Design and Evolution for XML-Schema . . . 35
COMA Combining Matchers . . . 25
DAG Directed acyclic graph . . . 37
DSI Descendant's Similarity Inheritance . . . 27
DTD Document Type Definition . . . 9
EBNF Erweiterte Backus-Naur-Form . . . 36
ELaX Evolution Language for XML-Schema . . . 3
EMX Entity Model for XML-Schema . . . 36
HTML Hypertext Markup Language . . . 9
idf inverse document frequency . . . 13
LWC Linear Weighted Combination . . . 27
PCG Pairwise connectivity graph . . . 24
PSM Parametric String-based Matcher . . . 26
OII Open Information Integration . . . 28
SGML Standard Generalized Markup Language . . . 9
SSC Sibling's Similarity Contribution . . . 27
tf term frequency . . . 13
VMM Vector-based Multi-word Matcher . . . 26
XML eXtensible Markup Language . . . 9

Kapitel 1

Einleitung

Zur Informationsdarstellung im Internet wurde die Sprache Hypertext Markup Language (HTML) entwickelt, welche vor allem Style bzw. Layoutinformationen abbildet und keine Semantik (vgl. [147]). Die nutzerdefinierbaren Strukturierungselemente dieser Sprache sind übersichtlich gestaltet und für Menschen gut lesbar, wobei die maschinelle Verarbeitung nur auf diesen Vorrat beschränkt ist (vgl. [72]).
Dieser Mangel wird in der Ableitung einer neuen Sprache behoben, indem über individuell bestimmbare Strukturierungselemente die Semantik (Inhalt) ausgedrückt werden kann. Als Basissprache wird Standard Generalized Markup Language (SGML) verwendet, die im Bereich von Text- und Dokumentverarbeitungssystemen große Verbreitung fand (vgl. [131]). Ausgehend davon wurde die vielseitig einsetzbare Sprache eXtensible Markup Language (XML) entwickelt, um ein breites Spektrum von Anwendungen zu unterstützen. Sie kann als Grundlage für weitere Sprachen dienen, beispielsweise als Datenspeicherungs- oder Datenverarbeitungs- bzw. Datenaustauschformat. Trotz der Vielseitigkeit von XML kann die Sprache nicht als Ersatz von HTML angesehen werden (vgl. [72] und [13]).
Vor allem im Bereich des Datenaustausches besteht die Notwendigkeit XML-Dokumente zu validieren. Der Validierungsprozess schließt die Überprüfung auf die syntaktische Korrektheit (Wohlgeformtheit), sowie den Abgleich gegen eine definierte Grammatik ein. Wird der Prozess erfolgreich abgeschlossen, ist das XML-Dokument gültig (valide) (vgl. [26]). Die zuerst verwendete Grammatik (Schema) für XML-Dokumente wurde aus SGML übernommen und wird als Document Type Definition (DTD) bezeichnet. Dabei weist die Verwendung der DTD Grenzen bezüglich der nicht unterstützten XML-Syntax, sowie des unzureichenden Typkonzepts auf. Daraufhin wurden mehrere Schemabeschreibungssprachen entwickelt, bei denen sich XML-Schema als Standard durchgesetzt hat (vgl. [131] und [25]).
Das XML-Schema kann im Laufe der Zeit an neue Verhältnisse angepasst werden, wobei veränderte Anforderungen an Anwendungen oder ein fehlerhafter Entwurf des Schemas denkbar sind (vgl. [146]). Die damit verbundenen Einfüge-, Lösch- und/oder Updateoperationen können zu nicht mehr schemakonformen XML-Dokumenten führen, die dann ebenfalls angepasst werden müssen. Dieser Vorgang wird als XML-Schema Evolution bezeichnet.
Ein Kernstück der Evolution ist die Formulierung und Erfassung von Änderungen, wobei der Vergleich zweier XML-Schemata als eine Herangehensweise existiert. Der Vorgang des Schemaabgleichs wird als Matching bezeichnet, welcher Korrespondenzen (Mappings) zwischen Schemakomponenten der beiden XML-Schemata ermittelt. Aus den so ermittelten Mappings können die entsprechenden Anpassungsschritte generiert werden, um das eine XML-Schema in das andere zu transformieren (vgl. [123]).
Die Masterarbeit stellt eine Übersicht über die verfügbaren Matching- und Mappingansätze dar, wobei hauptsächlich Verfahren mit Berücksichtigung der Schemaebene ausgewählt werden. Anschließend wird ein Überblick gegeben auf welche Art und Weise die Matching- und Mappingverfahren in aktuellen Forschungsprototypen und kommerziellen Systemen umgesetzt werden und inwieweit sie den Anwender bei der Evolution unterstützen. Ziel der Masterarbeit soll die Entwicklung eines Matching- und Mappingverfahren zur Verbesserung der Schemaevolution sein.

Die Funktionsweise des eigenen Ansatzes soll zum Einen mit der Durchführung der XML-Schema Evolution anhand eines selbstgewählten Beispiels und zum Anderen gegen ein an der Universität Rostock entwickelten Programm evaluiert werden.
Der weitere Aufbau der Arbeit stellt sich wie folgt dar:

- **Kapitel 2: Schema Matching Klassifikation**
  Dieses Kapitel untersucht welche Matching- und Mappingverfahren existieren und schafft somit die Grundlage für die Umsetzung dieser in der Praxis.
- **Kapitel 3: Übersicht über Matching- und Mappingverfahren in der Praxis**
  Das dritte Kapitel beschäftigt sich damit wie die Matching- und Mappingverfahren in aktuellen Datenbanksystemen und Forschungsprototypen umgesetzt wird und bietet die Grundlage zur Umsetzung eines eigenen Ansatzes zur XML-Schema Evolution.
- **Kapitel 4: Matching- und Mappingintegration in CodeX**
  Im vierten Kapitel steht die Implementierung des eigenen Ansatz im Vordergrund, welcher anschließend in das universitäre Tool CodeXintegriert wird.
- **Kapitel 5: Selbstgewähltes Beispiel für die XML-Schema Evolution**
  Dieses fünfte Kapitel erklärt den Ablauf des Szenarios, welches anhand einer vorgestellten Kategorisierung von Schemaevolutionsschritten abgeglichen wird.
- **Kapitel 6: Evaluierung des vorgestellten Ansatzes**
  Das sechste Kapitel beschäftigt sich mit der Durchführung des selbstgewählten Beispiels auf den eigenen Ansatz, sowie die Evaluierung gegen die herkömmliche Vorgehensweise im universitären Tool CodeX.
- **Kapitel 7: Zusammenfassung**
  Im letzten Kapitel wird die Thematik zusammengefasst und ein Ausblick gegeben.

Kapitel 2

Schema Matching Klassifikation

Das Kapitel vermittelt die Grundlagen von Schema Mapping und Matchingverfahren. In der Literatur [123], [134], [133] und [17] werden verschiedene Klassifikationen für Schemata erstellt, dargestellt in Abbildung 2.1. Eine komplette Übersicht über die Schema Matching Klassifikation befindet sich im Anhang A.

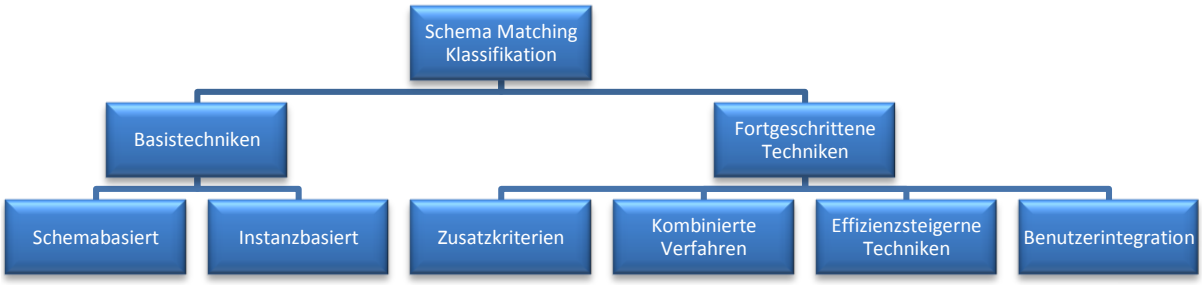

Abbildung 2.1: Schema Matching Klassifikation abgeleitet aus [123] und [17]

Demnach ist eine Unterscheidung in Basistechniken und fortgeschrittene Techniken notwendig. Unter Basistechniken werden etablierte und individuelle Matching- und Mappingverfahren verstanden. Diese gliedern sich in schemabasierte und instanzbasierte Ansätze auf und werden im Abschnitt 2.1 erläutert. Fortgeschrittene Techniken sind umfangreichere Matching- und Mappingverfahren, da sie zum Einen Basistechniken erweitern und zum Anderen neue Ansätze integrieren. Als grundlegende Erweiterungen sind Zusatzkriterien und kombinierte Verfahren zu nennen, die zusammen mit den Verfahren hinsichtlich der Effizienzsteigerung und der Benutzerintegration im Abschnitt 2.2 näher betrachtet werden.

2.1 Basistechniken

Die Basistechniken gliedern sich in schema- und instanzbasierte Ansätze auf, die detaillierter in den Abschnitten 2.1.1 und 2.1.2 erläutert werden.

2.1.1 Schemabasiert

Schemabasierte Ansätze sind nach [123] Methoden, welche Informationen über die Schemaelemente oder -struktur verarbeiten ohne Instanzdaten zu verwenden. Dabei wird jeweils nur ein einziger Aspekt behandelt, weshalb sie auch als Einzelmatcher bezeichnet werden (vgl. [94]).

Ausgehend von [123] sind die schemabasierten Klassifikationen [133] und [134] entstanden. Hierbei unterscheidet [133] in heuristische und formale Techniken. Die heuristischen Verfahren ermitteln elementbasierte und/oder strukturbasierte Gemeinsamkeiten, die am wahrscheinlichsten sind. Die formale Vorgehensweise benutzt dazu modelltheoretische Semantik in Form von Ontologien oder Aussagenlogik. Zusätzlich wird bei beiden Techniken geprüft, ob diese auf impliziten (syntaxbasierten) oder expliziten (semantischen) Informationen beruhen.
Bei [134] werden grundsätzlich drei verschiedene Ebenen (Layer) betrachtet. Die erste Ebene betrachtet, ob zunächst elementbasierte oder strukturierte Informationen von den schemabasierten Ansätzen verarbeitet werden. Ausgehend davon wird ermittelt wie die Techniken diese Informationen interpretieren, d.h. syntaktisch, extern oder semantisch. Die zweite Ebene, parallel zur ersten Ebene, untersucht auf welchen Daten die Algorithmen arbeiten, d.h. auf Zeichenketten (terminological), der Struktur (structural) oder Modellen (semantics). Die letzte Ebene stellt die Basistechniken dar, die in den zuvor beschriebenen Ebenen verwendet werden. Im Folgenden werden auf Grundlage dieser Basistechniken, die in Abbildung 2.2 dargestellten Algorithmen näher vorgestellt.

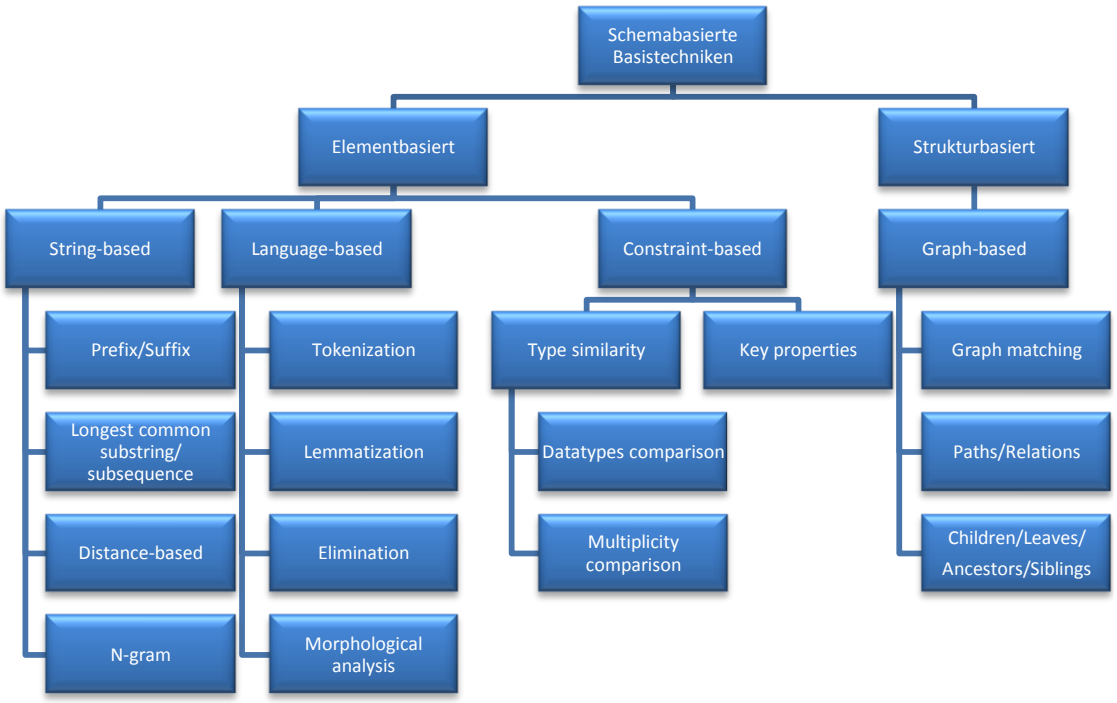

Abbildung 2.2: Schemabasierte Matchingalgorithmen abgeleitet aus [123], [133], [134] und [79]

Als Einstiegspunkt wird die Unterteilung der Basistechniken nach elementbasiert und strukturbasiert vorgenommen. Unter elementbasiert wird die Berechnung der Gemeinsamkeiten zwischen einzelnen Elementen verstanden, z.B. Typdefinitionen oder Attribute in den XML-Schemata. Strukturbasiert hingegen bedeutet, dass die Gesamtheit aller Elemente bzw. Teile davon miteinander in Beziehung gesetzt werden können auf Grundlage der Anordnung in der Hierarchie.
In Anlehnung an [134] sind bei den elementbasierten Verfahren die string-, language- und constraintbasierten, sowie bei den strukturbasierten Verfahren die graph-basierten Techniken von Bedeutung. Die dabei nicht dargestellten Verfahren „Upper level formal ontologies“, „Taxonomy-based“ und „Model-based“ sind für XML-Schemata nicht relevant, da sie in erster Linie Ontologien als Fokus haben. Die Verfahren „Linguistic resource“, „Alignment Reuse“ und „Repository of structures“ werden als fortgeschrittene Techniken eingestuft und deshalb im Abschnitt 2.2.1 näher untersucht.

String-based

Die string-basierten Techniken berechnen für zwei gegebene Zeichenketten einen Wert, der angibt inwiefern sie sich ähneln. Häufig bewegt sich dieser Wert in einer Skala von [0,1], wobei Null keine und Eins vollkommene Übereinstimmung bedeutet (vgl. [143]). Die ersten beiden Ansätze bestimmen gleiche Teilzeichenketten im vorderen bzw. hinteren Abschnitt eines Wortes (Prefix bzw. Suffix). Eine weitere Möglichkeit ist die Berechnung des Verhältnis der Anzahl gleicher Buchstaben einer Zeichenkette mit der einer anderen als Ganzes bzw. nach Reduzierung einiger Buchstaben (Longest common substring bzw. Longest common subsequence) (vgl. [79]).
Mit distance-basierten Ansätzen sind in erster Linie die Verfahren Edit-Distance, Token-basierte Distance und Hybrid Distance gemeint. Bei Edit-Distance wird die minimale Anzahl der Einfüge-, Lösch- und Ersetzoperationen gezählt, die nötig sind um eine Zeichenkette in die andere zu überführen. Dabei bewertet die Levenshtein Distance die drei Operationen mit Eins, wohingegen die Monger-Elkan Distance die Einfüge- und Löschoperation geringer gewichtet (vgl. [32]). Einen ähnlichen Ansatz verfolgen die Jaro-Distance und deren Erweiterung Jaro-Winkler-Distance. Bei der Jaro-Distance werden zunächst die Anzahl der gemeinsamen Zeichen und der Vertauschungen bestimmt, um diese Zeichen in der gleichen Reihenfolge zu erhalten. Die Jaro-Winkler-Distance berücksichtigt zusätzlich die Länge des gemeinsamen Präfixes, maximal die ersten vier Buchstaben (vgl. [51]). Die Token-basierte Distance betrachtet längere Zeichenketten bzw. Wortgruppen, die häufig im Information Retrieval eingesetzt werden. Im Kontext von XML-Schema können diese Distanzfunktionen zum Vergleich von Annotationen in XML Schemata angewandt werden (vgl. [4]). So vergleicht die Jaccard similarity zwei Wortmengen, indem der Durchschnitt der beiden Mengen durch dessen Vereinigung geteilt wird. Ein anderer Ansatz kombiniert die cosine similarity mit dem term frequency (tf)-inverse document frequency (idf)-Maß derart, dass identische, aber auch vertauschte Wörter, erkannt werden können (vgl. [33] und [113]). Eine Vereinfachung dessen wird in [46] beschrieben, wobei eventuelle Schreibfehler mitberücksichtigt werden. Die Hybrid Distance zerlegt beispielsweise zuerst die gegebenen Zeichenketten in Teilzeichenketten, um darauf z.B. die genannten distance-basierten Ansätze anzuwenden (vgl. [51]).
Außerdem ist die die Zerlegung der beiden Ausgangswörter in vorgegebene Teilzeichenketten der Länge n möglich (n-gram oder q-gram), z.B. das Trigram von „Rostock“ ist „Ros“, „ost“, „sto“, „toc“ und „ock“. Anschließend kann die Anzahl der übereinstimmenden Teilzeichenketten zwischen Quell- und Zielwort durch die n-gram-Anzahl des Quellwortes geteilt werden (vgl. [79] und [134]). Bei längeren Zeichenketten kann die Zuordnung fehlerhaft sein, weshalb [80] eine Erweiterung vorgeschlägt, die einzelne Abschnitte sinngemäß in Multimengen (cluster) zusammenfasst und dann darauf die n-grams ausführt.

Language-based

Häufig werden language-basierte Verfahren angewandt um die in einer bestimmten Sprache gegebenen Zeichenketten in eine Normalform zu bringen, die dann beispielsweise von string-basierten Techniken weiterverwendet werden können (vgl. [134]).
Zunächst existiert die Möglichkeit der Tokenization, bei der ein gegebener Text bzw. Zeichenkette in einzelne Wörter zerlegt wird. Hierbei kann beispielsweise die Whitespace-Tokenization angewandt werden, die die Leerzeichen zwischen den Wörtern als Trennungskriterium verwendet. Aufbauend auf der Tokenization kann zum Einen die Lemmatization und zum Anderen die Elimination eingesetzt werden. Bei der Lemmatization wird die Grundform eines Wortes hergestellt wo hingegen bei einer Elimination Wörter gelöscht werden die keine Relevanz haben, wie beispielsweise Artikel und Konjunktionen (vgl. [79]).
Die Morphological analysis untersucht Zeichenketten bzw. Wörter auf verschiedene Varianten. Mögliche Varianten entstehen beispielsweise durch Neubildung oder Kürzung von Wörtern. Dabei geht das gegebene Wort vollständig, teilweise oder in einer gebeugten Form im Singular bzw. Plural ein (vgl. [141]). Weiterhin beschreibt [3] die Expansion, welche Abkürzungen und Akronyme wieder ausschreibt, die z.B. durch die Morphological analysis entstanden sind. Außerdem lässt sich die Aussprache von Wörtern (Soundex) vergleichen (vgl. [54]).

Constraint-based

Mit Hilfe von constraint-basierten Techniken lassen sich Bedingungen, die zum Beispiel an Element- und/oder Attributdeklarationen gebunden sind, miteinander vergleichen (vgl. [94]).
Werden als Grundlage Typdefinitionen genommen dann wird von Type similarity gesprochen. Die Datatypes comparism verwendet Datentypen als Basis, wobei zum Einen die möglichen Ausprägungen der Wertebereiche verglichen werden können. Zum Anderen kann auf Grundlage einer sogenannten inheritance hierarchy, bei der die Datentypen in einer Rangfolge aufgelistet sind, z.B. beim Typsystem eines XML-Schemas (vgl. [19]), die Ähnlichkeiten zwischen zwei verschiedenen Datentypen ermittelt werden. Für das Typsystem des XML-Schemas existiert eine Koeffiziententabelle, die die Ähnlichkeit zwischen den 44 built-in Datentypen im Bereich von [0,1] angibt (vgl. [143]). Zwei weitere Ansätze stellen eine ähnliche Koeffiziententabelle auf, indem [145] die Facets von Datentypen mitberücksichtigt und [86] die Datentypen gruppiert, um zwischen den Gruppen Ähnlichkeitskoeffizienten festzulegen. Bei der Multiplicity comparison hingegen werden Kardinalitätsbeschränkungen verglichen, die in Verbindung mit Typdefinitonen angegeben werden können (vgl. [134]). So ist nach [79] eine Kombination mit Datentypen denkbar, indem zusätzlich die inheritance hierarchy verwendet wird. In [82] werden für DTD und in [145] für XML-Schemata Gleichheitskoeffizienten im Intervall [0,1] aufgestellt, die sich im Vergleich der gängigen Angaben (?,+,*) ergeben können. Dabei geht [145] noch einen Schritt weiter, indem für beliebige minOccurs und maxOccurs Werte eine Metrik erstellt wurde.
Neben Type similarity können Key properties verwendet werden, die Schlüsseleigenschaften berücksichtigen, in relationalen Schemata häufig über Primär- und vor allem Fremdschlüsselbeziehungen (vgl. [123]).

Graph-based

Der graph-basierte Ansatz verfolgt im Gegensatz zu allen vorher vorgestellten Techniken die Idee, dass die Struktur der Schemata als Graphen dargestellt wird. Diese Repräsentation wird verwendet um über Nachbarschaftsbeziehungen ähnliche oder gleiche Schemakomponenten aufzuzeigen (vgl. [79]).
Das Graph matching setzt die über die elementbasierten Techniken gefundenen syntaktischen Gemeinsamkeiten in Bezug zum Kontext bzw. der gesamten Struktur (vgl. [74]). Im Bereich des Graph matching kann die Idee der Edit Distance aus den string-basierten Verfahren verwendet werden, indem analog die Anzahl Editieroperationen bestimmt wird, um einen Graphen in den anderen umzuformen (vgl. [102]). So werden in [142] die Schemaknoten durchlaufen, wobei die Kosten für Einfüge-, Lösch- und Updateoperationen bestimmt werden, auf Grundlage von constraintbasierten Techniken und zusätzlichen Kriterien. Dabei ist Graph matching ein kombinatorisches Problem, welches bei größeren Strukturen schnell zu hoher Komplexität führt, weshalb Optimierungsprobleme entstehen (vgl. [134]).
Die Techniken Children, Leaves, Ancestors, Siblings und Paths/Relations betrachten nicht die gesamte Struktur, sondern nur Teile davon. Bei Children werden die inneren Knoten eines Graphs auf strukturelle Gleichheit überprüft. Hierbei sind diese Knoten dann gleich, wenn ihre unmittelbaren Kindknoten gleich sind. Ähnlich funktioniert der Leaves Ansatz, wobei es hier nicht auf die unmittelbare Gleichheit der Kindknoten ankommt, sondern auf die Gleichheit der Menge der Blattknoten (vgl. [79]). Die Ancestors Vorgehensweise funktioniert in die andere Richtung als die beiden zuvor genannten Techniken, d.h. die Betrachtung der übergeordneten Knoten einschließlich des Wurzelknotens liegt im Vordergrund (vgl. [6]). Bei der Siblings Methode werden die Navigationsachsen preceding-sibling und following-sibling ausgehend von den betrachteten Knoten untersucht (vgl. [4]). Als Interior wird die Kombination des Sibling Ansatzes mit einer Erweiterung des Children Ansatzes verstanden, indem alle Nachfolgerknoten betrachtet werden (vgl. [125]). Der Relations Ansatz kann über hergestellte elementbasierte Beziehungen zwischen zwei Schemata überprüfen, ob die Struktur um diese Schemakomponente herum identisch bzw. ähnlich ist (vgl. [134]). Der ähnliche Paths Ansatz überprüft die Pfade vom Wurzelknoten zum betrachteten Knoten bzw. zur betrachteten Knotengruppe auf Übereinstimmungen (vgl. [82]). Solche Ähnlichkeiten können beispielsweise über die überwiegend richtige Reihenfolge der Pfadknoten von Quell- und Zielschema in der Gesamtheit oder eines Teilausschnitts, z.b. der Pfadanfänge, bestimmt werden (vgl. [24]).

2.1.2 Instanzbasiert

Instanzbasierte Ansätze geben nach [123] einen Einblick auf die konkreten Inhaltsausprägungen und Bedeutung von Schemakomponenten. Diese können einerseits dazu benutzt werden, um im dazugehörigen XML-Schema die Schemaelemente richtig zu interpretieren. Andererseits kann bei Fehlen von Schemadefinitionen trotzdem XML-Dokumente miteinander verglichen werden (vgl. [94]). Hierzu können einige aus den schemabasierten Verfahren bekannten Techniken, wie Edit Distance, aber auch Information Retrieval basierte Techniken und weitere verwendet werden (vgl. [123], [141] und [48]). Beispielsweise werden in [141] die Instanzinformationen dazu benutzt, um ähnliche XML-Dokumente in gemeinsame Cluster zu bündeln. In [95] hingegen werden die Instanzen von Web directories z.B. von Google und Yahoo ausgewertet. Bei [2] werden Beispielinstanzdaten dazu benutzt, um im Prozess der Datenintegration die richtigen Mappings und letztendlich Transformationen vorzunehmen. Dabei untersucht [70] die Verwertbarkeit von Instanzinformationen für das Schema Matching, wenn diese schwer zu interpretieren sind. Grundlegend ist zu beachten, dass die Instanzdaten meist nur einen Ausschnitt zeigen, weshalb es umso besser ist, je mehr Instanzdaten zu einem XML-Schema und/oder Domäne vorhanden sind (vgl. [94]).

2.2 Fortgeschrittene Techniken

Die fortgeschrittenen Techniken lassen sich in Zusatzkriterien, kombinierte Verfahren, effizienzsteigerne Techniken und Benutzerintegration unterteilen, welche in den Abschnitten 2.2.1, 2.2.2, 2.2.3 und 2.2.4 näher betrachtet werden.

2.2.1 Zusatzkriterien

Werden zusätzlich zu den in den Basistechniken benutzten Schemata bzw. Instanzdaten Informationen bzw. Kriterien verwendet, wird von Zusatzkriterien gesprochen (vgl. [94]). Hierbei können diese die Qualität des Mappingverfahrens grundsätzlich steigern (vgl. [123]). Im Folgenden sollen hierbei die in Abbildung 2.3 dargestellten Verfahren im Einzelnen beschrieben werden.

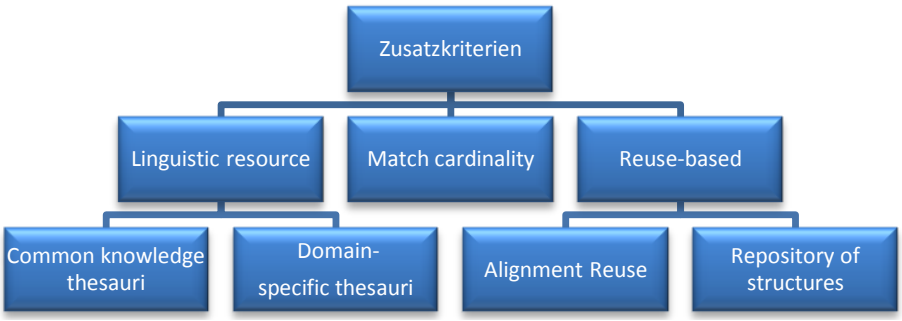

Abbildung 2.3: Zusatzkriterien abgeleitet aus [123], [134] und [17]

Linguistic resource

Linguistic resources verwenden als externe Informationsquellen zum Einen Lexika bzw. Common knowledge thesauri und zum Anderen Precompiled thesaurus bzw. Domain-specific thesauri (vgl. [133] und [134]). In einem Lexikon, z.B. WordNet, GermaNet oder OpenThesaurus, befindet sich eine große Menge an Wörtern, die über eine Taxonomie miteinander in Beziehung gesetzt werden. Darauf aufbauend können semantische Verbindungen, wie Synonyme oder Hyponyme ermittelt werden, aber auch die Länge des Pfades, um von einem Wort zu einem anderen zu gelangen.

Dabei bedeutet ein kurzer Pfad, dass sich die beiden Wörter stärker ähneln, als wenn die Pfadlänge größer ist (vgl. [74]). In [5] werden dafür der Wu&Palmer- und der Lin-Ansatz vorgestellt. Der Wu&Palmer bestimmt zunächst den kleinsten gemeinsamen Elternknoten zweier gegebener Wörter. Die Anzahl der Pfade von den gegebenen Wörtern zum gemeinsamen Elternknoten wird ins Verhältnis mit der Anzahl der Pfade vom Elternknoten zum Wurzelknoten gesetzt. Beim Lin-Ansatz wird zusätzlich die Wahrscheinlichkeit des Auftretens von Wörtern berücksichtigt. Weiterhin kann das Semantic tagging weitere semantische Informationen für den Mappingprozess beinhalten (vgl. [17]). Hier werden Korrespondenzen über mengenähnliche Beziehungen dargestellt, z.B. Gleichheit oder Teilmengenbeziehungen (vgl. [55] und [90]). Diese wiederrum können aus Lexika abgeleitet werden, z.B. werden nur bestimmte Wortbedeutungen, die in beiden Wörtern auftauchen, miteinander vereinigt (vgl. [56]).
Weiterhin können Domain-specific thesauri Verwendung finden, vor allem dann wenn die Common knowledge thesauri nicht mehr ausreichend sind. Hierbei können von Anwendungsbereich zu Anwendungsbereich gleiche Abkürzungen unterschiedliche Bedeutungen haben. So würde in der Informatik beispielsweise „DDR:double data rate“ einem Matcher signalisieren das DDR ein Synonym für double data rate ist, aber in der Politik für Deutsche Demokratische Republik steht (vgl. [79]).

Match cardinality

Mit der Match cardinality bzw. Matchkardinalität können Beziehungen zwischen Schemakomponenten zweier Schemata abgebildet werden (vgl. [94]). Hierzu werden in [84] verschiedene Mappingsituationen beschrieben, die eintreten können. Die erste Situation ist, dass keine Verbindungen zwischen Elementen existieren, auch als Missing correspondences bezeichnet. Daneben existieren die Single und Multiple correspondences, wobei Single correspondences 1:1 Beziehungen und Multiple correspondences 1:n, n:1 und n:m Korrespondenzen beschreiben. Direkte Zuordnungen (1:1 Beziehungen) werden von Matchingverfahren bevorzugt, weil dann automatisch der Mappingausdruck erzeugt werden kann, z.B. $SchemaA.Preis = SchemaB.Preis$. Im Gegensatz dazu erschweren 1:n, n:1 und n:m die automatische Erzeugung von Mappingausdrücken, weil hier häufig Expertenwissen nötig ist, z.B. SchemaA.Preis und SchemaA.MwSt ergeben $SchemaB.Kosten = SchemaA.Preis * (1 + SchemaA.MwSt/100)$(vgl. [123]). Ähnliche Mappingausdrücke können vorher definiert werden über sogenannte Conditional tagging (vgl. [17]). Hierbei werden kontextabhängige Bedingungen an Korrespondenzen formuliert, beispielsweise wenn ein Name einer Person, aber nicht eines Unternehmens, in Vor- und Nachname aufgespalten werden soll (vgl. [20]).

Reuse-based

Die reuse-basierten Ansätze verwenden als externe Informationen gespeicherte Mappings, die bei einem erneuten Matching wiederverwendet werden können. Grundsätzlich werden Alignment Reuse und Repository of structures unterschieden (vgl. [134]).
Hierbei bedeutet Alignment Reuse, dass zwei zu mappende Schemata, jeweils über Mappinginformationen zu einem dritten Schema besitzen. Diese können über die Transitivität ausgewertet und auf die zwei Eingabeschemata angewandt werden (vgl. [94] und [11]). In [148] und [153] werden im Zuge einer Schemaevolution bekannte Mappings zwischen einem zuvor evolutionierten Schema und dem Quellschema verwendet, um diese auf das Zielschema anzupassen. Dabei beinhaltet das Vorgehen ein zweifaches Matchingproblem, einerseits zwischen Quell- und Zielschema, andererseits zwischen den alten und neuen Mappings (vgl. [123]). Eine weitere Vorgehensweise stellt das dritte Schema als ein globales Schema dar, ein sogenanntes Pivot-Schema, welches im Idealfall alle Informationen bereits gemappter Schemata in sich vereinigt. Das ist dann sinnvoll wenn mehrere Schemata zugleich gemappt werden sollen, um so die Anzahl der direkten Mappings zu reduzieren (vgl. [94]).
Das Repository of structures enthält paarweise übereinstimmende Schemafragmente mehrerer Schemata, durch die Korrespondenzen zwischen zwei zu mappenden Schemata abgeleitet werden können (vgl. [134] und [127]).

Beispielsweise werden in [14] die Blattknoten-Paare abgespeichert, um über transitive Auswertungen bestehende Mappings wiederzuverwenden. Die Einträge werden häufig von Experten gepflegt und sind deshalb meist auf bestimmte Anwendungsbereiche beschränkt (vgl. [94]). Eine Erweiterung besteht darin, dass eine Vielzahl von Schemata einer Domäne lose gekoppelt vorliegen und deren Komponentenstatistiken ausgewertet werden (vgl. [88]). Des Weiteren können Repository of structures automatisch über Maschinelles Lernen aufgebaut werden (vgl. [94]). In [42] wird dazu eine Trainingsphase absolviert, in der bereitgestellte Instanzdaten, bei [44] Schemadaten, auf Ähnlichkeit geprüft und Gemeinsamkeiten verallgemeinert werden. In [39] wird dieser Ansatz erweitert, indem auch komplexere Ausdrücke wiederverwendbar sind (vgl. [17]).
Eine spezielle Form der Wiederverwendung von Mappinginformationen ist die Recovery, d.h. für bereits transformierte Schemata wird deren Ausgangsform wiederhergestellt (vgl. [9]). Die Verwendung von reuse-basierten Ansätzen in der Praxis kann in [122] nachgelesen werden.

2.2.2 Kombinierte Verfahren

Kombinierte Verfahren fassen mehrere Basistechniken zusammen, um dadurch die Qualität der Matchingresultate zu erhöhen (vgl. [94]). Eine spezielle Art der zusammengesetzten Verfahren ist die iterative Ausführung eines einzelnen Matchers (vgl. [116]). Nach [123] werden kombinierte Verfahren in Hybrid matcher und Composite matcher unterteilt, dargestellt in Abbildung 2.4.

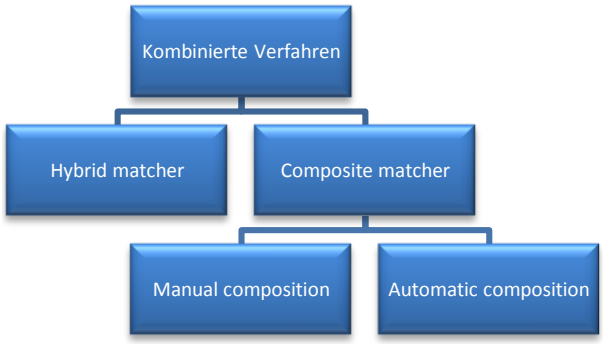

Abbildung 2.4: Kombinierte Verfahren abgeleitet aus [123]

Hybrid matcher

Ein Hybrid matcher kombiniert mehrere Basistechniken derart, dass beim Durchlauf zum Einen gleichzeitig mehrere Matchingkriterien überprüft werden können und zum Anderen können Teilergebnisse als Ausgangsdaten für nachfolgende Basistechniken in der fest vorgegebenen Reihenfolge dienen (vgl. [123]). In [31] wird ein Hybrid match Algorithmus für XML-Schemata vorgestellt, wobei eine Tiefensuche zum Einsatz kommt, die rekursiv Schemakomponenten vergleicht und gewichtete Quality of Match-Werte ausgibt. Zum Einsatz kommen string-, constraint- und graphenbasierte Ansätze, wobei Letzerer in [140] um einen Paths Ansatz erweitert wird. Ein ähnlicher Ansatz verwendet zum Durchlaufen der Graphenstruktur Spannbäume und wendet dann auf die Schemakomponenten verschiedene Heuristiken an, die dann in einer Matrix ausgewertet werden (vgl. [14]). In [75] und [76] wird das Matchingproblem in Teilprobleme unterteilt, d.h. die Phasen Word-to-Word, Node-to-Node, Path-to-Path und Tree-to-Tree werden durchlaufen, wobei zwischendurch Benutzerinteraktion optional möglich ist. Ein weiterer Ansatz kombiniert Linuistic resources (WordNet) mit distance-basierten Ansätzen, deren Beziehungen über die konjunktive Normalform angegeben werden (vgl. [28]).

Die optimale Zusammenstellung von Matcher in einem Workflow bedarf meist großer Expertenerfahrung und kann zum Einen zeitaufwendig sein und zum Anderen können sich bei besonders bei größeren Schemata fehlerhafte Entscheidungen negativ auswirken, weshalb eine Automatisierung nötig wird (vgl. [122]). In [15] wird beispielsweise das System eTuner vorgestellt, das auf Grundlage von einem Matchertool und Trainingsdaten, in Form von Instanzdaten, iterativ die bestmöglichen Parameter ermittelt, z.B. einen optimierten Threshold, der die Relevanz einer speziellen Matchaufgabe angibt. Damit kann die Qualität des Matchergebnisses bis zu 15% erhöht werden, abhängig von der Qualität der Trainingsdaten (vgl. [83]).

Composite matcher

Der Composite matcher führt die Basistechniken und/oder Hybrid matcher unabhängig voneinander aus und kombiniert deren Ergebnisse zu einem Gesamtresultat. Durch die unabhängige Ausführung können einfach neue Matcher hinzugefügt werden. Das Gesamtresultat hingegen kann z.B. über die Mittelwertbestimmung mit gleichen (Average) oder manuell festgelegten (Weighted) Gewichten bestimmt werden (vgl. [117]). Eine Möglichkeit zur dynamischen Berechnung der Gewichtung verschiedener distancebasierter Ansätze wird in [85] vorgestellt. Hierbei werden die Gewichte aus zuvor bestimmten Label- und Strukturfaktoren berechnet.
Weiterhin wird zwischen Manual composition und Automatic composition unterschieden, wobei Ersteres ein manuelles und Letzteres ein automatisches Auswählen der simultanen Ausführung mehrerer Basistechniken bzw. Hybrid matcher erlaubt (vgl. [123]). Außerdem ist ebenfalls eine sequentieller Ausführung möglich, bei der eine Weitergabe von Teilergebnissen an die nachfolgenden Basistechniken unterstützt wird (vgl. [94]).
In [18] wird ein manuelles Auswählen der Ausführung vorgeschlagen, indem strategy scripts über eine graphische Benutzungsoberfläche zur Verfügung gestellt werden. Diese können neu erstellt werden oder liegen bereits in gespeicherter Form vor und können gegebenenfalls überarbeitet werden. In [115] wird ein Framework vorgestellt, welches über ein Process Designer manuell verschiedene Prozessreihenfolgen erstellen kann. Dabei kann nicht nur die Auswahl von Verfahren aus einer Sammlung von Matchern, sondern auch wie diese kombiniert und eingesetzt werden sollen, von einem Experten festgelegt werden. Ein weiterer Ansatz automatisiert die Ausführungsauswahl dahingehend, dass Entscheidungsbäume eingesetzt werden (vgl. [43]). Diese wählen automatisch einzelne Algorithmen aus einer Sammlung von Matchingverfahren, die dann in einer optimierten oder qualitativen Reihenfolge ausgeführt werden. Der Nutzer muss sich lediglich nur für eine initial angebotene Abarbeitung entscheiden, kann aber durchaus eine Überarbeitung vornehmen oder neue Entscheidungsbäume erstellen. Eine vollständige Automatisierung bietet das in [116] vorgestellte Matching System. Hier wird eine auf das konkrete Matchingproblem angepasste Ablaufreihenfolge zusammengestellt, indem ausgehend von den gegebenen Schemata Features ermittelt und durch vordefinierte Matching Rules ausgewertet werden. Ebenfalls findet in [150] eine Automatisierung statt, indem über mehrere Iterationen, unter Einbeziehung von benutzerdefinierten Trainingsdaten, die Mappings verfeinert werden. Dabei werden nur solche Mappings am Ende des Matchingprozesses vorgeschlagen, die mit einer genutzten Transformationssprache auch ausführbar sind.

2.2.3 Effizienzsteigerne Techniken

Effizienzsteigerne Techniken sollen zum Einen die entwickelten Matcher verbessern, ohne auf Qualität verzichten zu müssen, und zum Anderen mit größeren Matchingproblemen umgehen, z.B. zwischen großen XML-Schemata aus den E-Business Standards (vgl. [122]). Auf der einen Seite können durch verschiedene Vorverarbeitung der importierten Schemata Effizienzvorteile generiert werden, z.B. durch Node labeling Techniken oder Indexing, um unnötige Navigation durch Graphstrukturen zu vermeiden (vgl. [135] und [128]). Hierbei können in [7] und [3] die XML-Schemakomponenten durch die Nutzung von Prüfer Code als consolidated Prüfer sequence dargestellt werden.
Andererseits kann die Performance bei der Ausführung von paarweisen Schemamatchingmethoden durch Parallel matching und/oder Reduction of search space gesteigert werden, dargestellt in Abbildung 2.5.

Für das gleichzeitige Matchen von n-Schemata existieren Holistic-Ansätze, die in [65], [67], [139], [66], [129] und [122] nachgelesen werden können. Außerdem kann die Wiederverwendung eine entscheidene Rolle zur Effizienzsteigerung beitragen, indem z.B. vorhandene Mappings wiederverwendet werden können und sich somit Matchaufgaben reduzieren lassen. Dabei wird in [62] eine effiziente Möglichkeit der Speicherung von Mappings in einer neuartigen Speicherstruktur vorgestellt, dem block tree. Weiterhin können Effizienzvorteile durch die Wahl der Navigation durch XML-Dokumente generiert werden, beispielsweise durch Twig patterns in Kombination mit ID/IDREF (vgl. [151]).

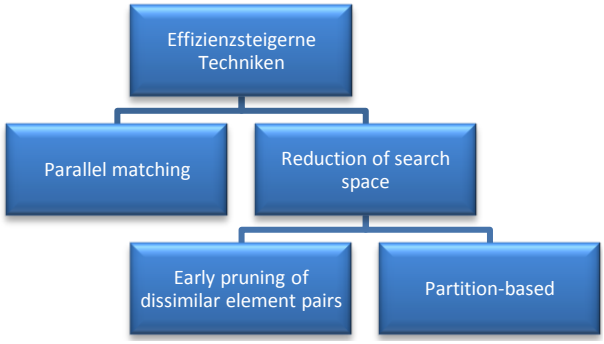

Abbildung 2.5: Effizienzsteigerne Techniken abgeleitet aus [122] und [130]

Parallel matching

Beim Parallel matching werden unabhängige Teilaufgaben gleichzeitig ausgeführt. Dies kann zunächst ähnlich zum composite matcher geschehen, indem mehrere voneinander unabhängige Matcher gleichzeitig ausgeführt werden, auch als Inter-matcher parallelization bezeichnet (vgl. [122]). Eine weitere Möglichkeit ist der Einsatz von Intra-matcher parallelization, wobei einzelne Matcher intern in unabhängige Teile gegliedert und parallel ausgeführt werden können. Somit ist auch eine Parallelisierung für Matchaufgaben in einer Sequenz möglich (vgl. [130]). Die Implementierung der Kombination von Inter- und Intra-matchter parallelization wurde in [63] vorgestellt. Jedoch ist zu beachten, dass die parallele Ausführung nur so schnell ist, wie der Task mit der höchsten Ausführungsdauer (vgl. [122]). Somit macht die Parallelisierung Sinn, wenn die Ausführungszeiten aller Tasks identisch bzw. ähnlich sind, um keine zusätzlichen Effizienznachteile zu produzieren.

Reduction of search space

Die Reduction of search space reduziert den Suchraum in den untersuchten Schemata derart, dass Effizienzvorteile durch den Einsatz von Early pruning of dissimilar element pairs oder partition-basierte Ansätze entstehen können. Bei der Early pruning of dissimilar element pairs werden frühzeitig gering übereinstimmende Schemakomponenten aus der weiteren Betrachtung im Matchingprozess ausgeschlossen (vgl. [122]). In [45] werden alle Schemakomponenten aussortiert, die nicht gleiche Namensbezeichnungen oder Strukturen besitzen. Auf den übrig gebliebenen Schemakomponenten werden dann weiterführende Matcher eingesetzt. Ein anderer Ansatz reduziert durch Kombination definierter Features, die anfänglich oder zwischendurch Informationen aus den gegebenen Schemata sammeln, mit Matchingregeln den Suchraum und liefert einen optimalen Matchingworkflow (vgl. [114] und [116]).
Der partition-basierte Ansatz verfolgt das Ziel das Quellschema in Regionen zu zerlegen, um diese mit einer Teilmenge von Regionen, im Idealfall nur mit einer Region des Ziel-Schemas, zu vergleichen. Die Performance kann durch parallele Ausführung noch weiter erhöht werden (vgl. [122]). In [68] beispielsweise werden die Partitionen (Blocks) nach strukturellen Gesichtspunkten eingeteilt. Die miteinander zu vergleichenden Blöcke werden durch zuvor ermittelte Namensgleichheiten in den jeweiligen Blöcken erkannt.

Somit bestimmt die höchste Anzahl gleicher Namen in Quell- und Zielschema welche Regionen miteinander gematcht werden. Ein ähnlicher Ansatz bestimmt in beiden Schemata Subgraphen (approximate common substructures), die verglichen werden, um potentielle Übereinstimmungen zu finden. Dabei wird gleichzeitig eine Optimierung des graphbasierten Ansatzes mit der Edit-Distance vorgenommen, weil dort die Anzahl der Editieroperationen sehr hoch sein kann, obwohl die Substrukturen ähnlich bzw. identisch sind (vgl. [86]). In [27] hingegen wird davon ausgegangen, dass das Zielschema ein Update des Quellschemas ist und somit ähnliche bzw. identische Fragmente vermutet werden. Diese Fragemente werden mithilfe einer Dekomposition ermittelt und anschließend gematcht. Die Vorgehensweise in [73] teilt zuerst das Zielschema in Regionen (target parts), um ausgehend davon passende Regionen im Quellschema (source parts) zu finden. Dabei enthalten die target parts den jeweiligen Elternknoten, ausgenommen der Wurzelknoten, und die dazugehörigen Kindknoten, es sei denn die Kindknoten sind selbst wieder Elternknoten. Anschließend werden gefundene Mappings zwischen source und target parts zu einem Gesamtmapping vereinigt und als Ergebnis ausgegeben. Ebenfalls wird in [135] zunächst das Zielschema in Regionen bzw. Cluster unterteilt, unter der Berücksichtigung der Knotenarten im zugrundeliegenden Quellschema. Somit muss das Quellschema nicht gegen das gesamte Zielschema geprüft werden.

2.2.4 Benutzerintegration

Die Einbeziehung des Nutzers spielt vor allem dann eine Rolle, wenn im Matchingprozess ein Feedback benötigt wird, sodass insgesamt die Matchqualität und gegebenenfalls die Performance steigt (vgl. [17]). Nach Abbildung 2.6 werden zwischen GUI support, Top-k matching, Collaborative user involvement und usage-basierten Ansätzen unterteilt, die nachfolgend genauer erläutert werden.

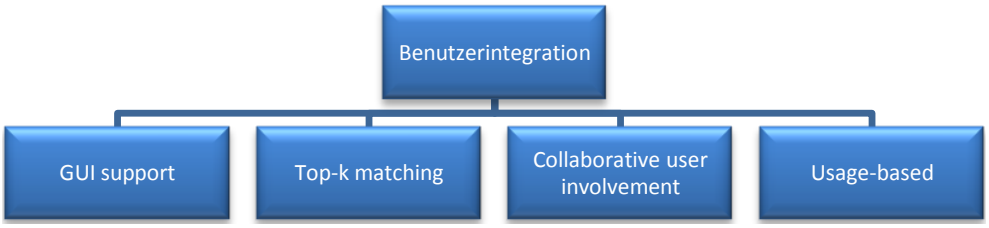

Abbildung 2.6: Verfahren zur Integration des Benutzers abgeleitet aus [17] und [122]

GUI support

Um den Benutzer wirkungsvoll in den Matchingprozess zu integrieren, können Benutzeroberflächen eingesetzt werden, die eine visuelle Darstellung erlauben, welche beispielsweise Mappingkorrekturen oder Workflowanpassungen seitens des Benutzers unterstützen (vgl. [17] und [122]). Quell- und Zielschema können beispielsweise in einer Baumstruktur, ähnlich einem Explorer, oder in einem Formular dargestellt werden (vgl. [50] und [23]). Mappings können dann direkt zwischen den Schemata beispielsweise mit durchgezogenen oder gestrichelten Linien oder in Schriftform visualisiert werden. Dabei können die Linien über Farben oder Bezeichnungen Informationen über den Ähnlichkeitsgrad zweier Schemakomponenten besitzen. Außerdem ist ein Hervorheben von einzelnen oder mehrerer ausgewählten Mappings möglich, um sie von anderen abzuheben (vgl. [126]). Weiterhin lassen sich in [121] Matchingkardinalitäten, sowie verschiedene Linienarten, z.B. zwischen Elementen oder zwischen Attributen, visualisieren. Bei großen Schemata ist der Einsatz der Fisch-Perspektive möglich, d.h. es werden angeklickte Schemakomponenten hervorgehoben und Nachbarkomponenten verkleinert dargestellt (vgl. [50]).
Der Ansatz in [1] kann bei gefundenen Mappings, die nicht sicher zugeordnet werden konnten, direkt über Ja/Nein-Fragen Feedback anfordern. Dabei werden auf Grundlage von Instanzdaten Beispiele erzeugt, die dem Nutzer beim Matchingprozess unterstützen sollen.

In [23] können einzelne Schemakomponenten ausgewählt werden, wodurch automatisch generierte Mappings über Pfeile angezeigt werden. Die zutreffenden Zielkomponenten können mithilfe eines Aufklappmenüs verworfen oder angepasst werden. Eine ähnliche Vorgehensweise präsentiert [138], wobei alle Mappings nicht auf einmal sondern kompakt in einer Tabelle aufgelistet werden. Einzelne Datensätze werden dann visuell dargestellt und Feedback angefordert. Dieser Ansatz wird in einer Benutzerstudie mit der herkömmliche Vorgehensweise, wo alle Mappings gleichzeitig dargestellt werden, verglichen. Demnach ist der neuartige Ablauf intuitiver und besser, vor allem bei Designern mit geringen Kenntnissen. Ebenfalls mit geringen Kenntnissen seitens des Nutzers ist der sogenannte Sample-Driven, vorgestellt in [119]. Hierbei werden Kenntnisse des Zielschemas und Beispielinstanzen vorausgesetzt. Die Beispielinstanzen erlauben den Vergleich mit Instanzen des Quellschemas und können so Äquivalenzen ableiten, um das eine Schema in das andere zu überführen.

Top-k matching

Das Top-k matching reduziert die Anzahl aller möglichen Mappings einer Schemakomponente auf eine vielversprechende Teilmenge mit k Elementen (vgl. [17]). Dabei zeigt [52], dass Top-K matching unter Umständen falsche Mappings generiert, z.B. wenn die Auswahl zwischen gleichgewichteten Mappings erfolgen soll. Deshalb wird eine Heuristik vorgeschlagen, in der der Nutzer einen bestimmten Threshold definiert, ab dem keine Mappings mehr zugelassen werden sollen. In [53] wird der Ansatz erweitert, indem komplexere Beziehungen auf Basis mehrerer Top-k Matchings dargestellt werden können. Hierbei werden die Thresholds über die Qualität vorher bestimmter Cluster gebildet, wobei zwischen einem bestimmten Bereich die Autoren Nutzerfeedback für sinnvoll halten. Ein weiterer Ansatz verwendet das Top-k matching, um die beste Transformationsstrategie auszuwählen, die das Quell- in das Zielschema übersetzt (vgl. [120]). Außerdem existiert das Verfahren des Incremental matching, wo der Nutzer eine Schemakomponente oder mehrere auswählt und dazu das beste oder die besten (Top-k) Matching(s) präsentiert bekommt (vgl. [16]). So kann der Nutzer schrittweise Mappings überprüfen und gegebenenfalls anpassen, ohne dabei den Überblick zu verlieren. In [152] wird der Top-k Algorithmus in Verwendung mit string-basierten Verfahren der Edit Distance und n-gram benutzt, um zu einer Zeichenkette passende Strings aus einer Sammlung auszuwählen. Dabei werden in [62] zum Einen Standardverfahren, z.B. Murty vorgestellt, die Mappings mit der höchsten Summe von Ähnlichkeitsgewichten auswählen. Zum Anderen werden effizientere Verfahren erläutert, die auf Grundlage von Partitionen und Heaps die Performance erhöhen können.

Collaborative user involvement

Collaborative user involvement bezieht mehrere Benutzer in den Matchingprozess mit ein, um zum Einen den Aufwand des Einzelnen zu reduzieren und zum Anderen die Matchqualität des Tools zu erhöhen (vgl. [50]). So wird in [154] eine webbasierte Community aufgebaut, die Schemata verknüpfen und hochladen. Dabei können die Mappings, die von einzelnen Teilnehmern erstellt werden, sowohl von anderen Teilnehmern oder von einem Tool wiederverwendet werden. Ein weiterer Ansatz verwendet zielgerichtete Fragen, z.B. ob ein bestimmter Built-In Datentyp mit einem benutzerdefinierten übereinstimmt oder ob eine Schemakomponente mit einer anderen übereinstimmt (vgl. [97]). Dabei werden die Antwortmöglichkeiten auf die Zustimmung, Ablehnung oder Überspringen bzw. Unwissenheit reduziert. Die Ergebnisse werden dann wiederrum der Community zur Verfügung gestellt und können weiterverwendet werden.

Usage-based

Die usage-basierten Verfahren stellen im Gegensatz zu den schemabasierten und instanzbasierten Ansätzen laut [47] eine neue Kategorie dar. Hierbei werden im Allgemeinen verwertbare Informationen aus von Nutzern gestellten Query logs gewonnen, um daraus Matchkandidaten zu finden (vgl. [122]). In [47] werden dazu als Grundlage SQL query logs verwendet, jedoch sind auch XQuery bzw. XPath-Anfragen denkbar.

Voraussetzung für die Verwertbarkeit ist, dass die verglichenen Query logs dieselben Informationen extrahieren, aber aus unterschiedlichen Schemata. So können beispielsweise über Join-Bedinungen äquivalente Attribute in beiden Schemata ermittelt und gematcht werden. Im Bereich von Suchmaschinen verwendet ein weiterer Ansatz Clicklogs, die die URL und das Suchwort des Nutzers speichert (vgl. [108]). Daraus können Ähnlichkeiten in einer Taxonomie abgeleitet werden, z.B. Synonyme für gleiche Bereiche. Jedoch besteht bei usage-basierten Ansätzen das Problem der Verfügbarkeit und Verwertbarkeit solcher Query logs, deutlich schwieriger als bei Instanzdaten, weshalb der Einsatz nur begrenzt möglich ist (vgl. [122]).

Kapitel 3

Übersicht über Matching- und Mappingverfahren in der Praxis

Die beiden nachfolgenden Abschnitte 3.1 und 3.2 geben einen Überblick wie die Matching und Mappingverfahren in den Forschungsprototypen bzw. kommerziellen Systemen umgesetzt werden. Anschließend wird im Abschnitt 3.3 die Verwertbarkeit der vorgestellten Ansätze untersucht.

3.1 Forschungsprototypen

Die hier vorgestellten bzw. ausgewählten Forschungsprototypen werden häufig in der Literatur zitiert und zu Evaluationen herangezogen, wenn beispielsweise ein neues Matchingsystem implementiert wird (vgl. [134], [102], [7], [125], [3], [76], [17], [122] und [98]). Deshalb werden im Folgenden die Matchingansätze von Cupid, Similarity Flooding, COMA 3.0, AgreementMaker, ++Spicy und OII Harmony vorgestellt.

3.1.1 Cupid

Das in [89] vorgestellte hybride, generische Matchingtool Cupid verbindet element- und strukturbasierte Verfahren. Diese können einfach auf andere Datenmodelle und/oder Anwendungsdomänen adaptiert werden. In Cupid selbst werden relationale Datenmodelle oder XML-Schemata als Quell- und Zielschemata bereitgestellt, die jeweils als Graph (schema tree) kodiert werden. Anschließend werden gewichtete Koeffizienten (wsim) zwischen den Schemakomponenten ermittelt, die sich im Intervall von [0,1] bewegen: $wsim = w_{struct} * ssim + (1 - w_{struct}) * lsim$. Dabei gibt w_{struct} im gleichen Intervall an, ob das Linguistic matching (lsim) oder Structual matching (ssim) mehr gewichtet wird.
Beim lsim werden Schemaelemente zunächst über eine Normalization vorverarbeitet, was vergleichbar ist mit den language-basierten Basistechniken. Hierbei werden Tokenization, Expansion, Elimination und Tagging größtenteils basierend auf einem Thesaurus durchgeführt, wobei unter Tagging die Zuordnung von einem Konzept bzw. Oberbegriff zu einem oder mehreren Token verstanden wird. Außerdem werden alle Namenstoken in fünf verschiedene Typen eingeteilt, beispielsweise number oder special symbol. Aufbauend auf diesen Tokentypen und dem Tagging werden Kategorien erstellt, um die Anzahl der Elementvergleiche zu reduzieren, indem nur kompatible Kategorien miteinander verglichen werden. Der lsim ergibt sich schließlich aus der Ähnlichkeit der Token und den Kategorien (vgl. [87]).
Der ssim basiert auf der Ähnlichkeit von Kontext bzw. Umgebung in der die Schemakomponenten eingebettet sind. Dabei werden hauptsächlich die Blattknoten miteinander verglichen bzw. die übereinstimmenden Blattknotenmengen, weil angenommen wird, dass sich diese Struktur weniger ändert als die der inneren Knoten. Deshalb wird ssim höher gewichtet, wenn die Ähnlichkeit der Blattknoten einen bestimmten Threshold übersteigt (vgl. [123]).

Abschließend werden die Mappingpaare ausgewählt, bei denen der wsim-Wert einen bestimmten Threshold überschreitet. Dadurch können Quell-Schemakomponenten zu genau einer oder zu mehreren Ziel-Schemakomponenten gehören, d.h. Matchkardinalitäten von 1:1 und 1:n sind möglich (vgl. [89]).

3.1.2 Similarity Flooding

Similarity Flooding ist ein generischer und struktureller Algorithmus mit der Annahme, dass zwei Schemakomponenten ähnlich sind, wenn sich ihre benachbarten Komponenten gleichen. Der Algorithmus kann verschiedene Datenstrukturen, z.B. XML-Schemata, in gerichtete und beschriftete Graphen konvertieren (vgl. [104]). Dabei werden zwei Repräsentationsmöglichkeiten für XML-Schemata vorgestellt. Die erste Repräsentation OEM/Lore verwendet Elementtags als Beschriftungen, während die zweite Repräsentation DOM hierarchische Beziehungen über die Bezeichnungen child darstellt. Dabei ist zu beachten, dass die Transformation in diesen Graphen vorher implementiert sein muss (vgl. [103]).
Aus dem erzeugten Graphen wird ein sogenannter Pairwise connectivity graph (PCG) abgeleitet. Dieser besteht aus den Verknüpfungen von Quell- und Zielschemakomponenten, die über eine gemeinsame Beschriftung verfügen. Danach werden die im PCG beschrifteten Kanten zunächst um eine Kante in Gegenrichtung ergänzt und anschließend werden anstelle der Beschriftungen Propagierungskoeffizienten eingefügt, die sich im Intervall von [0,1] bewegen. Der resultierende Graph wird als induced propagation graph bezeichnet (vgl. [104]). Die Berechnung der Koeffizienten kann beispielsweise in Abhängigkeit von den aus- bzw. eingehenden Kanten erfolgen. Somit wären bei zwei ausgehenden Kanten die jeweiligen Koeffizientenwerte 0,5 denkbar. Weitere Berechnungsalternativen können in [103] nachgelesen werden.
Anschließend werden die Mappingpaare aus dem induced propagation graph extrahiert und ein Initialmapping durchgeführt, d.h. das string-basierte Verfahren Prefix/Suffix wird verwendet. Aufbauend auf dem Initialwert und den Propagierungskoeffizienten wird über eine Fixpoint Berechnung übereinstimmende Schemakomponenten in Quell- und Zielschema berechnet. Die Berechnung iteriert solange bis ein bestimmter Wert ϵ unterschritten oder eine festgesetzte Anzahl von Iterationen erreicht wird (vgl. [104]). Danach werden für jedes Element eine Menge von Matchkandidaten erzeugt, wobei bei n Matchingpaaren 2^n Untermengen existieren. Damit der Nutzer nicht mit einer Vielzahl von Matchmöglichkeiten konfrontiert wird, können Filter eingesetzt werden, die die Anzahl der Mappings reduziert. So können verschiedene Kandidaten durch Matchkardinalitäten und/oder einem Threshold ausgewählt werden (vgl. [103]). Als Ergebnis können 1:1 und n:m Mappings erzeugt werden (vgl. [123]). Der Algorithmus wird in [105] verwendet, indem z.B. zwei XML-Schemata ausgewertet werden, dargestellt in Abbildung 3.1.

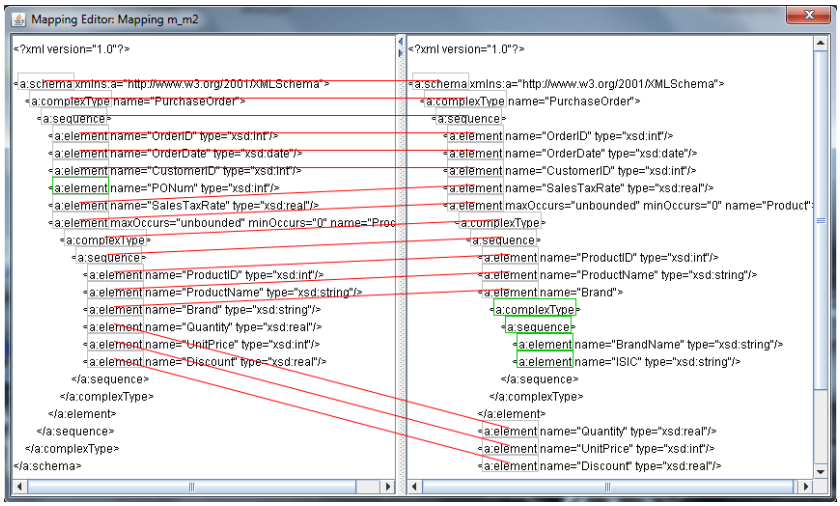

Abbildung 3.1: Benutzeroberfläche von Rondo

3.1.3 COMA 3.0

In [96] wird das generische Tool und composite matcher Combining Matchers (COMA) in der Version 3.0 beschrieben, welches aus den Vorgängern COMA (vgl. [41]) und COMA++ (vgl. [12] und [48]) entstanden ist. Dabei wird COMA sowohl in einer Community Edition, d.h. Open-Source-Projekt (vgl. [10]), als auch in einer Business Edition angeboten, wobei Letztere nur auf Anfrage erhältlich ist.
Aus COMA++ werden bekannte Aspekte übernommen, wie der Import von Schemata, Ontologien, Zusatzinformationen und Mappings, aber auch der Export zu Ausgabemodellen und Mappings. Außerdem wird das Repository übernommen, welches Matchkonfigurationen, importierte und exportierte Informationen beinhaltet und aus dem die Mappingergebnisse wiederverwendet werden können. Weiterhin können durch eine Execution Engine definierte Matchstrategien ausgeführt werden, d.h. die Abarbeitung verschiedener Matchingverfahren, sowie Vor- und Nachbearbeitung durch Einsatz von Bibliotheken (vgl. [96]).
Die Neuerungen in beiden Editionen sind zum Einen die Configuration Engine mit der automatischen Konfigurationsmöglichkeit und zum Anderen die Überarbeitung des User Interfaces. Beim Letzeren wird eine neue GUI entwickelt (Abbildung 3.2), sowie der Zugriff auf die Funktionalitäten von COMA entweder über eine API oder über eine Web Edition bzw. Web Service ermöglicht (vgl. [96]).

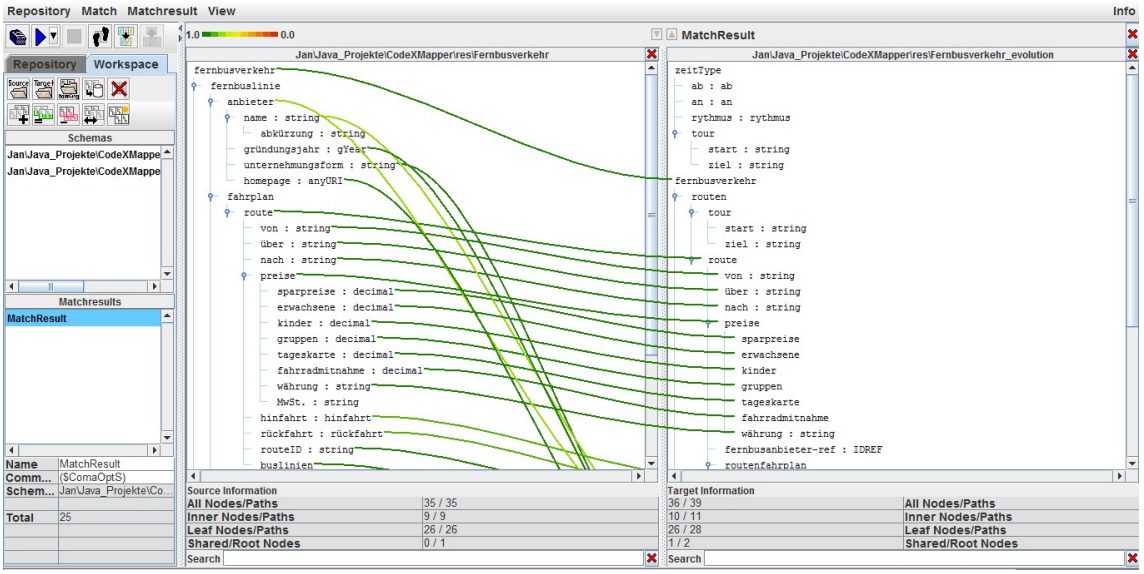

Abbildung 3.2: Benutzeroberfläche von COMA 3.0

Die Configuration Engine unterstützt einerseits die manuelle (Expert Mode) und andererseits die automatische (simple mode) Konfiguration des Ablaufs von Matchingverfahren (Workflow). Dabei wird die entstehende Workflow-Konfiguration validiert. Die automatische Konfiguration wählt zunächst die Default-Matchstrategie als Startpunkt aus. Anschließend werden die Eingabeschemata auf die Verfügbarkeit von Instanzen, Kommentaren und Datentypen überprüft, um Matchingverfahren auszuwählen, die diese Informationen verarbeiten können. Außerdem werden wiederverwendbare Mappings auf Verwendbarkeit geprüft. Weiterhin können durch linguistische und strukturelle Ähnlichkeiten element- und/oder strukturbasierte Matchingverfahren ausgewählt werden. Wenn die Eingabegrößen der Schemata eine bestimmte Größe erreichen wird gegebenenfalls eine partion-basierte Vorgehensweise ausgewählt, das fragment matching (vgl. [96]). Dabei werden Fragmente eines speziellen Typs erzeugt, d.h. benutzerdefinierte, Subschemata oder automatisch durch einen Clusteralgorithmus. Diese werden oberflächlich miteinander verglichen, z.B. nur die Wurzelknoten der Fragmente, um dann ähnliche Fragmente im Detail zu untersuchen. Die Teilergebnisse werden dann zu einem Gesamtergebnis vereinigt (vgl. [124] und [122]). Bisher nur in der Business Edition sind die Enrichment, Merge und Transformation Engines verfügbar.

Dabei ermöglicht die Enrichment Engine die semiautomatische oder manuelle Erweiterung von 1:1 Beziehungen zu komplexeren Beziehungen, z.B. 1:n oder n:1. Dabei werden zum Erkennen komplexer Beziehungen die Instanzdaten der betrachteten Schemata herangezogen. In der Merge Engine wird das Ontology Merging unterstützt, wobei in der Transformation Engine ausführbare Mappings für die Datentransformation erzeugt werden. Der letztere Ansatz ist aus dem Matchingsystem Spicy entnommen, welcher genauer in 3.1.5 betrachtet wird (vgl. [96]).

3.1.4 AgreementMaker

Das in [37] vorgestellte Tool ist praxisorientiert für die Verarbeitung von Geodaten entwickelt worden (vgl. [36]). Als Eingabeschemata werden Ontologien in verschiedenen Formaten akzeptiert, z.B. in XML. Diese werden in einer Baumstruktur innerhalb einer Benutzeroberfläche dem Domänexperten angezeigt, dargestellt in Abbildung 3.3.

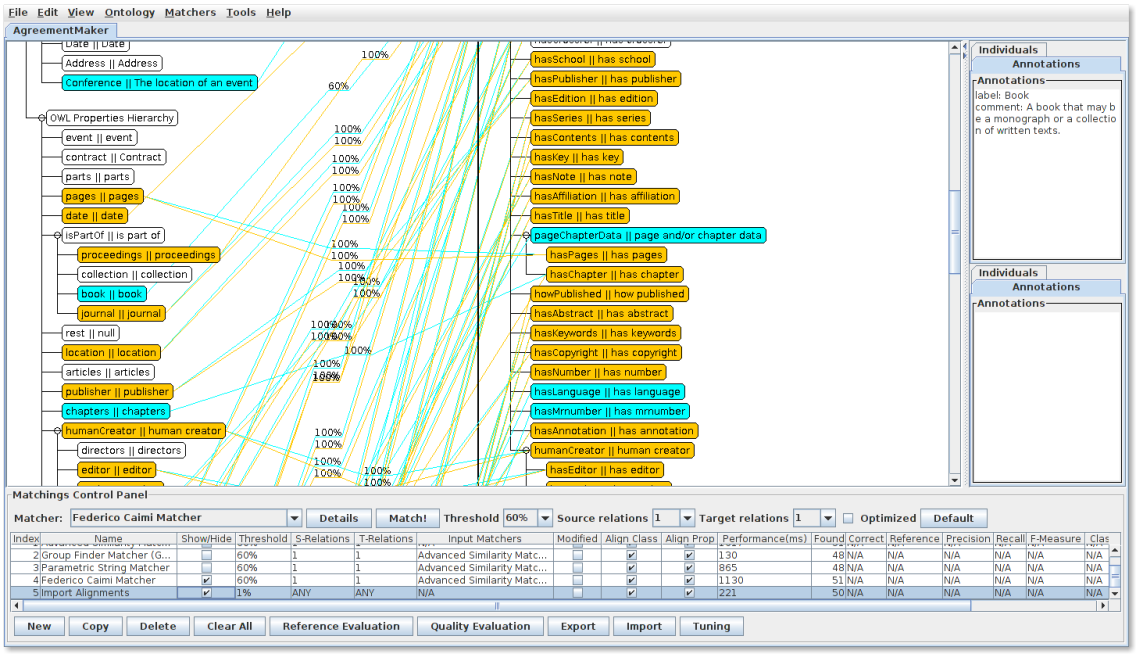

Abbildung 3.3: Benutzeroberfläche von AgreementMaker entnommen aus [137]

Die Mappings können manuell oder automatisch definiert werden. Beim Letzteren werden drei Matchingschritte (Layer) durchlaufen, wobei kombinierte Verfahren (hybrid, composite) erlaubt sind (vgl. [37]). Die Matchingverfahren im ersten Layer verwenden elementbasierte Informationen und werden zum Base Similarity Matcher (BSM) zusammengefasst. Dieser besteht aus den string- und language-basierten Verfahren Parametric String-based Matcher (PSM) und Vector-based Multi-word Matcher (VMM) und verwendet daneben noch die Linguistic resource WordNet. Bei PSM kann der Nutzer zunächst zwischen verschiedenen string-basierten Verfahren, wie Jaro-Winkler und Edit-Distance auswählen. Weiterhin kann der Normalisierungsprozess mit language-basierten Verfahren, wie Stammwortreduktion und Stoppworteliminierung, definiert werden. Die Ähnlichkeit wird dann als gewichteter Durchschnitt berechnet, wobei der Nutzer das Gewicht festlegt. Beim VMM werden Container mit Termen, die Informationen zu Konzepten enthalten, zu term frequency (tf)-inverse document frequency (idf)-Vektoren transformiert und anschließend mit der cosine similarity verglichen (vgl. [34]).

Der zweite Layer verfügt über die strukturbasierten Verfahren Descendant's Similarity Inheritance (DSI) und Sibling's Similarity Contribution (SSC). Dabei untersucht DSI ob die beiden verglichenen Konzepte als Nachkommen ähnlich sind, d.h. ob sie ähnliche Elternkonzepte und Abstände zum Wurzelknoten des jeweiligen Baumes besitzen. Bei SSC hingegen werden die Anzahl und die übereinstimmenden Nachbarkonzepte mit in die Betrachtung einbezogen (vgl. [35]).
Im letzten Layer werden in einem ersten Schritt die Matchergebnisse der vorherigen Matcher über Linear Weighted Combination (LWC) miteinander kombiniert. Dabei können manuelle oder automatische Gewichte festgelegt werden. Beim Letzteren werden Evaluationsmethoden eingesetzt, wobei häufig die Mappings mit bereits von Domänexperten getätigten Mappings (gold standard) verglichen werden (vgl. [37]). Sind solche gold standards nicht vorhanden werden zum Einen Mappings höher gewichtet, die die Ordnung von Konzepten nicht grundlegend verändern, sowie Distanzänderungen zwischen Konzepten möglichst gering halten (global-selection measures). Zum Anderen können die ausgewählten Gleichheitswerte in der Ähnlichkeitsmatrix durch den Durchschnitt der anderen Gleichheitswerte in der Reihe reduziert werden und als Gewicht verwendet werden (local-similarity quality measure). Anschließend wird in einem zweiten Schritt aus der kombinierten Ähnlichkeitsmatrix diejenigen Mappings ausgewählt, die zum Einen einen bestimmten Threshold besitzen und zum Anderen eine bestimmte Matchkardinalität. Dabei können Kardinalitäten von 1:1, 1:n, n:1 und n:m unterstützt werden (vgl. [37]).
Die Ergebnisse der jeweiligen Layer werden in der Benutzeroberfläche durch verschiedene Farben dargestellt. Außerdem kann der Nutzer sich für Ergebnisse eines oder mehrerer Layer entscheiden, sodass die Übersichtlichkeit gewahrt bleibt (vgl. [35]). Ein Nachteil des AgreementMakers ist die Notwendigkeit eines Domänexperten, um beispielsweise nicht vom Tool entdeckte Mappings zu ergänzen (vgl. [130]).

3.1.5 ++Spicy

Das in [93] vorgestellte Open-Source-Tool ++Spicy ist aus den Vorgängern Spicy (vgl. [21] und [22]) und +Spicy (vgl. [101], [99], [92] und [100]) entstanden. Das Ziel von ++Spicy ist das Finden einer optimalen Transformationslösung aus verschiedenen Mappings für ein Matchingproblem. Dabei vermag die aktuellste Version neben relationalen Schemata auch mit XML-Schemata umzugehen (vgl. [93]).
Das passende SQL- bzw. XQuery-Skript wird durch Hochladen von Quell- und Zielschema mit mindestens einer Quellinstanz erzeugt. Zuvor können Mappings entweder durch externe Tools zur Verfügung gestellt oder durch das Tool selbst erzeugt werden (Abbildung 3.4), wobei dann eine Zielinstanz nötig ist.

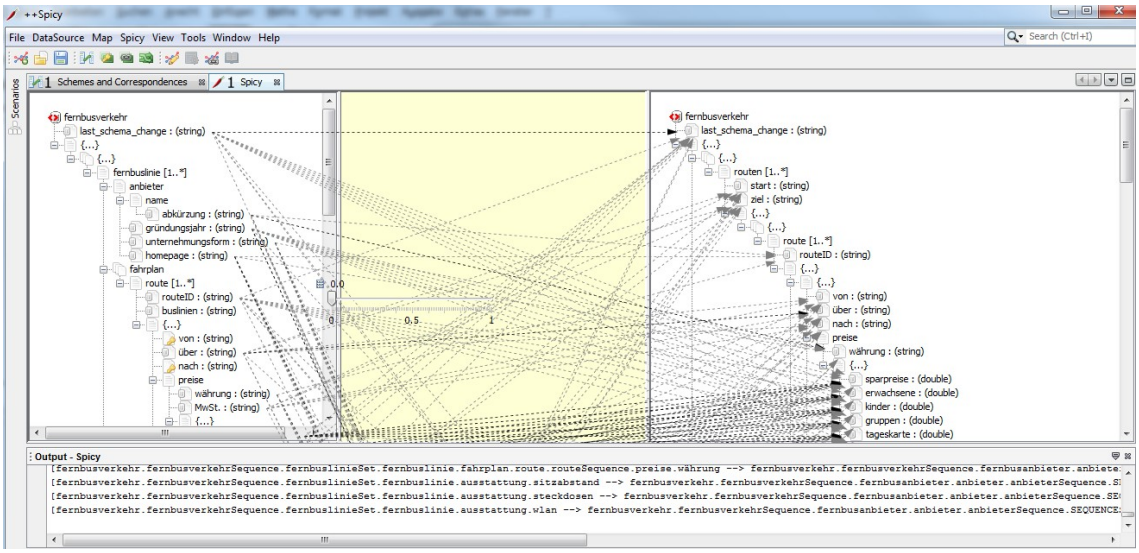

Abbildung 3.4: Benutzeroberfläche von ++Spicy

Die Mappings werden zunächst über einen dynamischen Threshold (0,6-0,99) ausgewählt, um sie dann durch die Rewrite Engine auf formaler Ebene umzuformen. Damit lassen sich z.B. funktionale Abhängigkeiten auflösen (vgl. [101]). Darauf aufbauend wird über die Chase Engine aus der Quellinstanz eine Zielinstanz erzeugt, die canonical solution (vgl. [99]). Diese canonical solution wird zur Mappingverifikation herangezogen, indem bestehende Mappings verfeinert oder fehlerhafte entfernt werden können. Dazu werden beispielsweise Mappingausdrücke variiert und die daraus resultierenden Zielinstanzen mit der canonical solution verglichen. Somit können Redundanzen vermieden, Transformationsschritte optimiert und ein effizientes Erzeugen von Transformationsskripten gewährleistet werden. Dieses Vorgehen wird als structural analysis bezeichnet, welches die Schemata und Instanzen auf Basis der Logik von elektrischen Schaltkreisen (electrical circuits) analysiert (vgl. [21]).

3.1.6 OII Harmony

In [132] wird das Open-Source-Projekt Open Information Integration (OII) vorgestellt, mit dem Ziel eine einheitliche, öffentlich zugängliche und erweiterbare Plattform für die Informationsintegration zu schaffen. Sie besteht aus den drei Kernkomponenten Yggdrasil (Repository), Importer/Exporter (z.B. von XML Schema oder Excel-Tabellen), sowie den Komponententools. Unter Letzterem fällt das hier betrachtete hybride, schemabasierte Matchingtool Harmony (vgl. [132]), dargestellt in Abbildung 3.5.

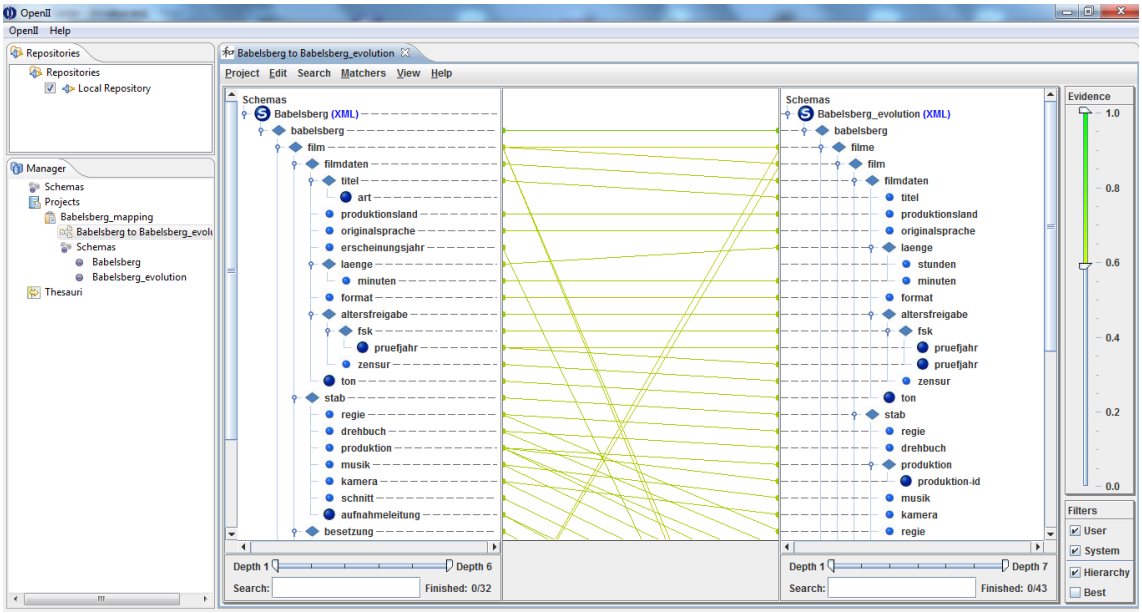

Abbildung 3.5: Benutzeroberfläche von OII Harmony

OII Harmony normalisiert die Eingabeschemata zunächst in eine kanonische Graphrepräsentation. Darauf aufbauend werden zunächst language-basierte Verfahren, wie Tokenization oder Stammwortreduktion durchgeführt. Anschließend führen benutzerdefinierte Matchingverfahren (match voters) string-basierte Verfahren, z.B. Edit Distance, oder Linguistic resources, wie beispielsweise ein Akronymmatcher in Verbindung mit einem Domain-specific thesauri aus. Jeder match voter liefert für jedes Mappingpaar ein Wert im Intervall von [-1,+1], wobei -1 für keine, +1 für eine sichere und 0 für eine unsichere Korrespondenz stehen. Diese Werte werden zu einem Ergebnis vereinigt, d.h. Werte im Bereich von +1 werden höher gewichtet als diejenigen mit 0 oder -1. Danach wird eine Version des Similarity Floodings ausgeführt, welche die zusammengefassten Ergebnisse auf Grundlage von strukturellen Informationen anpasst. So können z.B. zwei Attribute nicht gematcht werden, wenn ihre Elternknoten nicht übereinstimmen.

Letztendlich werden die übriggebliebenen Mappingpaare graphisch als farblich unterlegte Linien dargestellt, um beispielsweise benutzerdefinierte von automatisch erzeugten Mappings auseinander zu halten (vgl. [106]).
Weitere interessante Komponententools sind RMap bzw. XMap, welche die Mappings in die ausführbaren Transformationsskripte SQL bzw. XQuery übersetzen (vgl. [132]). Aber auch Unity, welches Mappings zwischen vorhandenen Schemata zu einem Standardschema erzeugt und verwandte Konzepte als synsets im Repository abspeichert. Dieses kann dann als zusätzliches Thesaurus zur Verbesserung der Matchqualität in Harmony verwendet werden (vgl. [136]).

3.2 Kommerzielle Systeme

Die nachfolgenden komerziellen Matchingsysteme werden in der Literatur erwähnt, weshalb im Folgenden die Matchingansätze von Altova und IBM näher untersucht werden (vgl. [126], [29], [3], [17], [122], [93] und [64]).

3.2.1 Altova DiffDog/MapForce 2013

DiffDog und MapForce sind jeweils von Altova entwickelte Werkzeuge, die sich mit dem Vergleich und Mapping von (XML)-Dokumenten beschäftigen.
In [57] wird DiffDog als XML-Vergleichstool vorgestellt, welches zwischen XML-Dokumenten oder XML-Schemata Unterschiede ermittelt. Weiterhin können auch Textdateien, z.B. Microsoft Word-Dokumente, Datenbankschemata, sowie Datenbankdaten verglichen werden. Für den Vergleich zweier XML-Schemata stellt das Programm die Schemata graphisch nebeneinander dar, dargestellt in Abbildung 3.6.

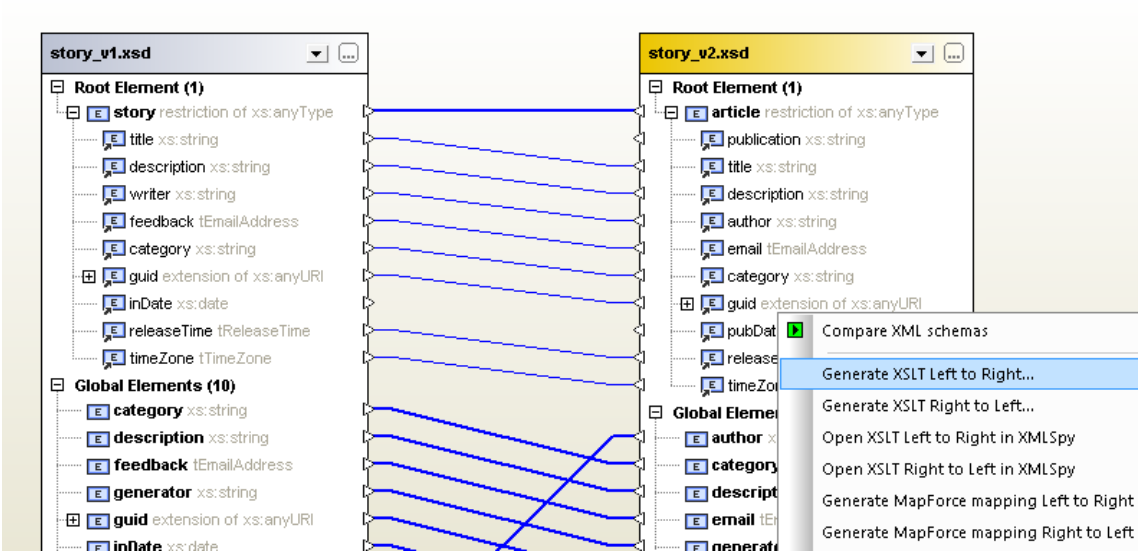

Abbildung 3.6: Benutzeroberfläche von Altova DiffDog entnommen aus [59]

Anschließend werden vom anpassbaren Root-Element automatisch alle gleichnamigen, untergeordneten Elemente über Mappings verknüpft. Dabei kann der Nutzer jederzeit die Mappings manuell anpassen bzw. hinzufügen. Diese können dann für einen späteren Gebrauch gesichert aber auch weiterverwendet werden. Im letzteren Fall können aus den Mappings zum Einen ein XSLT-Skript generiert werden. Zum Anderen lassen sich die Mappings auch in MapForce-Mappings übersetzen, die von MapForce weiterverarbeitet werden können (vgl. [57]).

MapForce wird in [58] als visuelles Mapping Tool beschrieben (Abbildung 3.7, welches einerseits das Mappen zwischen z.B. XML-Schemata ermöglicht und andererseits das Generieren von Transformationsskripten wie XQuery und XSLT und deren Ausführung erlaubt.

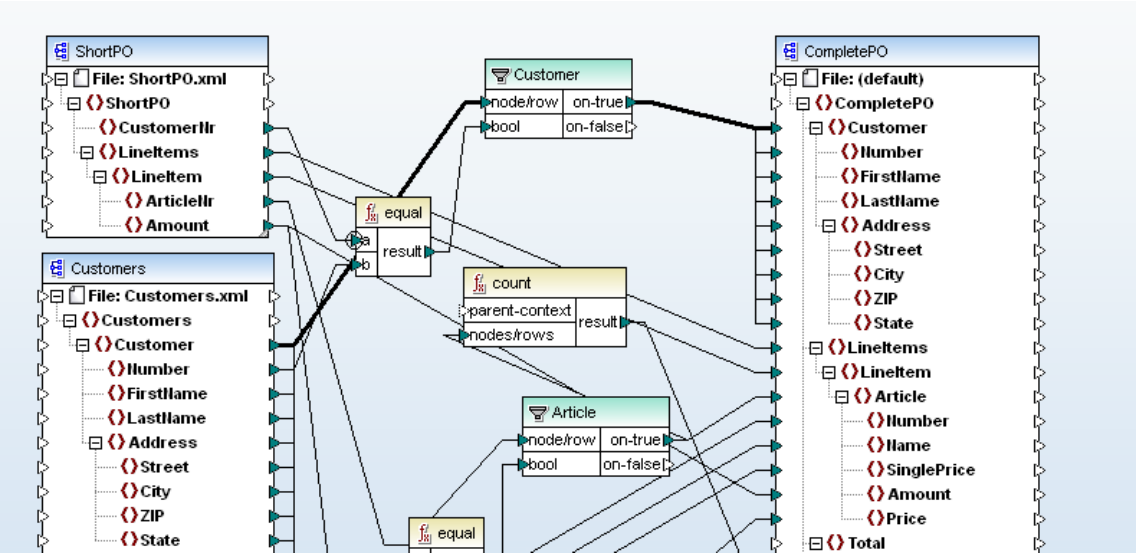

Abbildung 3.7: Benutzeroberfläche von Altova MapForce entnommen aus [60]

Dabei wird nicht von einer Schemaevolution zwischen den Schemata ausgegangen, sondern von komplett heterogenen Schemata (vgl. [64]). Deshalb werden die Mappings in der Regel manuell erstellt, d.h. Quell- und Zielkomponenten können direkt (1:1), über Funktionen (z.B. concat) und/oder über Filter (z.B. aktzeptiert nur true von wahrheitswertigen Funktionen) miteinander verbunden werden. Bei aktivierter Autovervollständigung wird die automatische Verbindung zwischen gleichen Unterkomponenten ermöglicht. Sollen mehrere Quellkomponenten mit einer einzigen Zielkomponente verbunden werden, müssen zunächst Duplikate der Quell- und Zielkomponenten erzeugt werden, um darauf aufbauend die Mappings durchzuführen. Die Erzeugung des Transformationsskriptes kann auf den drei Verbindungsarten Source-driven, Target-driven und/oder Copy-all basieren. Bei Source-driven können automatisch Text- und Kindknoten in der gleichen Reihenfolge gemappt werden, wie diese in der Quellinstanz auftreten, z.B. Mixed Content. Die Standardeinstellung Target-driven hingegen berücksichtigt die Reihenfolge der Kindknoten in Abhängigkeit vom Zielschema, wobei Mixed Content Textknoten nicht unterstützt werden. Das Copy-all erlaubt beim Verbinden zweier Elternknoten die automatische Verknüpfung der gleichnamigen Kindknoten ohne Beachtung des Typs. Außerdem können zum Einen Mappings validiert werden, z.B. werden Fehler durch fehlende Mappings beim Input/Output von Funktionen generiert. Zum Anderen ist die Validierung der Mappingausgabe möglich, d.h. die potentielle Zielinstanz wird erzeugt und gegen das Zielschema validiert (vgl. [58]). Für die aktuellsten Spezifikationen, wie XML Version 1.1 oder XSLT 3.0, hat Altova 2013 RaptorXML als Nachfolger des Prozessors AltovaXML veröffentlich (vgl. [61]).

3.2.2 IBM InfoSphere Data Architect 9.1

Die von IBM bereitgestellte Entwicklungsumgebung InfoSphere Data Architect 9.1 (Rational Data Architect) umfasst eine Reihe von Tools zur Datenmodellierung, zur Integration und zur Verknüpfung von Datenbeständen sowie zur Entwicklung von Datenbankanwendungen (vgl. [149]). Unter anderem wurde das semiautomatische Mappingtool Clio (vgl. [118], [30] und [107]) in den InfoSphere Data Architect integriert, welches neben dem Mapping zwischen relationalen und/oder XML-Schemata das anschließende Transformationsskript in SQL, SQL/XML, XQuery oder XSLT anbietet (vgl. [49] und [81]).

Über einen sogenannten Zuordnungseditor werden auf der linken Seite die Quell- und auf der rechten Seite die Zielschemata dargestellt, während in der Mitte die verschiedenen Zuordnungen (Mappings) angezeigt werden können, dargestellt in Abbildung 3.8. Die Mappings können manuell oder automatisch

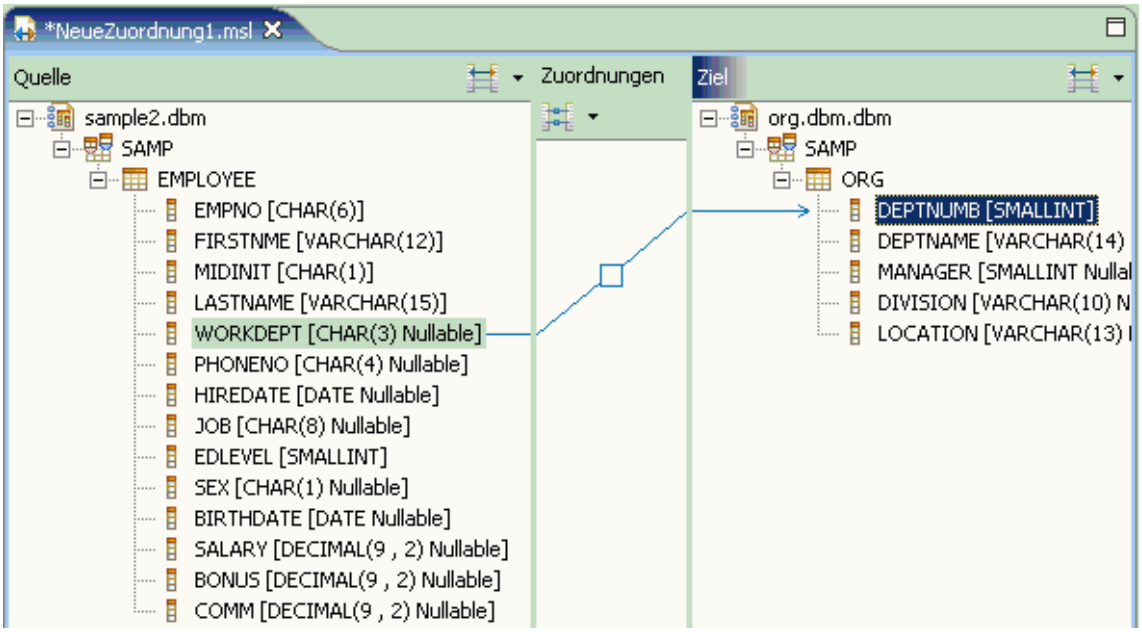

Abbildung 3.8: Benutzeroberfläche von IBMInfoSphere Data Architect 9.1 entnommen aus [69]

erstellt werden. Bei der automatischen Generierung werden die besten 1:1-Mappings dem Benutzer vorgeschlagen, die manuell aktzeptiert oder abgelehnt werden müssen. Diese Zuordnungen werden aufgespürte Zuordnungen genannt und werden über die Namensgleichheit entdeckt, die zum Einen Schreibfehler und zum Anderen externe Thesauri mitberücksichtigt (vgl. [122]). Daneben existieren die Zuordnungen, welche manuell erstellt oder durch ein vorheriges Mapping übernommen wurden. Wenn mehrere Elemente eines XML-Schemas gemappt werden, dann wird eine Zuordnungsgruppe definiert. Außerdem besteht die Möglichkeit der Zuweisung einer Konstanten zu einer Zielkomponente über eine Konstantenzuordnung. Ungültige Zuordnungen entstehen zwischen bereits bestehenden Mappings, wenn beispielsweise Namens- und/oder Typänderungen der gemappten Schemakomponenten vorgenommen werden (vgl. [149]).
Für die anschließende Transformation können die Zuordnungen angepasst werden, indem Transformationsregeln erstellt werden. Hierbei werden zunächst Filter unterstützt, die nur bestimmte Anteile aus dem Quellschema selektieren. Weiterhin können durch Angabe von Sortierbedingungen, z.B. die zu mappenden Elemente zuvor in aufsteigender Reihenfolge sortiert werden bevor sie transformiert werden. Unter Verwendung von Umsetzungen können n:1, aber aber auch 1:n Zuordnungen definiert werden. Letzteres wird im Tool durch mehrere einzelne 1:1 Mappings umgesetzt. Hierbei bedeuten Umsetzungen, dass die Schemakomponente bzw. Schemakomponenten aus dem Quellschema durch Anpassungen in die gewünschte Form gebracht werden. Außerdem ist die Verknüpfung mehrerer Quellkomponenten zu einer Zielkomponente über Joinbedingungen möglich (vgl. [149]).

3.3 Abschließende Bewertung

Die in den vorherigen Abschnitten vorgestellten Forschungsprototypen und kommerziellen Systemen werden zusammenfassend in Abbildung 3.9 dargestellt.

Forschungsprototypen/ kommerzielle Systeme			**Cupid**	**Similarity Flooding**	**COMA 3.0**	**Agreement-Maker**	**++Spicy**	**OII Harmony**	**Altova DiffDog/ MapForce**	**IBM Info-Sphere Data Architect 8.5**
Schematypen			XML, relational	XML, relational	XML, relational, Ontologie	XML, Ontologie	XML, relational	XML, relational, Ontologie	XML	XML, relational
Basistechniken	Schemabasiert	String-based	√	√	√	√	-	√	√	√
		Language-based	-	-	√	√	-	√	-	-
		Constraint-based	√	-	√	√	-	√	-	-
		Graph-based	√	√	√	√	√	√	-	-
	Instanzbasiert		-	-	√	√	√	-	(√)	-
Fortgeschrittene Techniken	Zusatzkriterien	Linguistic resource	√	-	√	√	-	√	-	√
		Match cardinality	1:1 und n:1	1:1 und n:m	1:1, 1:n und n:1	1:1, 1:n und n:1	1:1	1:1, 1:n und n:1	1:1	1:1
		Reuse-based	-	-	√	-	-	-	-	-
	Kombinierte Verfahren		√ (hybrid)	√ (hybrid)	√ (hybrid, composite)	√ (hybrid, composite)	√ (hybrid)	√ (hybrid, composite)	?	?
	Effizienz-steigerne Techniken	Parallel matching	-	-	(√)	-	-	-	-	-
		Reduction of search space	-	-	√	-	-	-	-	-
	Benutzer-integration	GUI support	√	-	√	√	√	√	√	√
		Top-k matching	-	-	-	-	-	-	-	-
		Collaborative user involvement	-	-	-	-	-	-	-	-
		Usage-based	-	-	-	-	-	-	-	-
Verfügbarkeit			?	akademische Zwecke	Open Source AGPL V3.0	akademische Zwecke	Open Source	Open Source Apache License V2.0	Kommerziell	Kommerziell
√ = vorhanden, (√) = teilweise Unterstützung, - = nicht vorhanden, ? = keine Angabe										

Abbildung 3.9: Übersicht der Matching- und Mappingverfahren abgeleitet aus [123], [40], [3], [17] und [130]

Demnach werden von allen Implementierungen XML-Schema als Schematypn akzeptiert. Darüber hinaus unterstützen einige Tools relationale Schemata und/oder Ontologien, wie beispielsweise AgreementMaker, ++Spicy oder OII Harmony. Hinsichtlich der unterstützten Basistechniken, können lediglich die Forschungsprototypen COMA 3.0 und AgreementMaker alle schema- und instanzbasierten Techniken abdecken, wobei OII Harmony auf instanzbasierte Verfahren verzichtet. Dabei werden von COMA 3.0 eine Vielzahl von Matchern in einer eigens dafür angelegten Matcherbibliothek zur Verfügung gestellt. Im Vergleich dazu verwenden die kommerziellen Systeme von Altova und IBM string-basierte Verfahren, um Namensgleichheiten aufzudecken, wobei IBM zusätzlich die Einbeziehung externer Thesauren erlaubt. Ähnliche Einbeziehungen dieser Art von Zusatzkriterien werden ebenfalls von Cupid, COMA 3.0, AgreementMaker und OII Harmony unterstützt.
Die automatisch erzeugbaren Matchingkardinalitäten reichen von 1:1 bei ++Spicy, Altova und IBM bis hin zu n:m bei Similarity Flooding. Dabei können manuell komplexere Beziehungen dargestellt werden, die z.B. bei IBM über mehrere 1:1 Mappings dargestellt werden. Sollen bereits in der Vergangenheit getätigte Mappings wiederverwendet werden, so bietet nur COMA 3.0 die Möglichkeit dazu.
Die Forschungsprotypen unterstützen alle hybride Verfahren, wobei COMA 3.0, AgreementMaker und OII Harmony zusätzlich composite erlauben. Somit verfügen die zuletzt genannten verschiedene kombinierte Verfahren, die vom Nutzer ausgewählt werden können. Zusätzlich bietet COMA3.0 die manuelle Anpassung des Workflows an, d.h. Bestimmung der Matcher-Reihenfolge, die Anpassung von Thresholds, sowie die Reihenfolge von Kombinationsmöglichkeiten, wie z.B. Average.
Im Bereich der effizienzsteigernden Techniken bietet nur COMA 3.0 das Parallel Matching (vgl. [130]) an, sowie die Reduction of search space mithilfe des Fragment Matchings. Aus der fortgeschrittenen Technik der Benutzerintegration wird der GUI support von allen Tools über eine graphische Benutzeroberfläche ermöglicht, ausgenommen Similarity Flooding. Dabei können automatisch generierte Mappings editiert werden. Dabei punkten die kommerziellen Systeme mit einer Menge von Funktionsbibliotheken, die beispielsweise mehrere Mappings bündeln und zu einem Mapping verknüpfen. Die Forschungsprototypen beschränken sich auf das Hinzufügen und Löschen von Mappings.

Bei der Verfügbarkeit können Similarity Flooding und AgreementMaker nur für akademische Zwecke verwendet werden, wobei COMA 3.0, ++Spicy und OII Harmony unter Open Source Lizenzen angeboten werden. Dadurch können die zuletzt genannten Tools erweitert werden, was bei Altova und IBM nicht der Fall ist, da sie kommerziell sind. Aus allen Vergleichspunkten geht hervor, dass COMA 3.0 alle Basistechniken und eine Vielzahl von fortgeschrittenen Techniken umgesetzt hat, welche frei verfügbar sind. Deshalb wird im nachfolgenden Kapitel 4 ausgehend von COMA 3.0 zunächst eine stand-alone Anwendung entwickelt, die anschließend in *Conceptual Design and Evolution for XML-Schema* (CodeX) integriert wird.

Kapitel 4

Matching- und Mappingintegration in CodeX

Ausgehend von den Überlegungen in Abschnitt 3.3 wird die Matching- und Mappingfunktionalität von COMA in *Conceptual Design and Evolution for XML-Schema* (CodeX) integriert bzw. implementiert, siehe Abschnitt 4.3. Das Tool COMA wurde bereits im Kapitel 3 näher vorgestellt, sodass im Abschnitt 4.1 die Grundlagen zu Conceptual Design and Evolution for XML-Schema (CodeX) und der Evolutionssprache *Evolution Language for XML-Schema* (ELaX) gelegt werden. Anschließend wird im Abschnitt 4.2 die allgemeine Architektur der zu integrierenden und umzusetzenden Funktionalitäten vorgestellt.

4.1 Grundlagen

In den beiden folgenden Abschnitten 4.1.1 und 4.1.2 werden zum Einen der CodeX-Editor und zum Anderen die Evolutionssprache ELaX näher vorgestellt.

4.1.1 CodeX

Der an der Universität Rostock entwickelte Forschungsprototyp *Conceptual Design and Evolution for XML-Schema* (CodeX) wird im Rahmen der Arbeiten [78], [77] und [110] ausführlich vorgestellt. Der CodeX-Editor bietet eine Reihe von Funktionen, dargestellt in Abbildung 4.1.

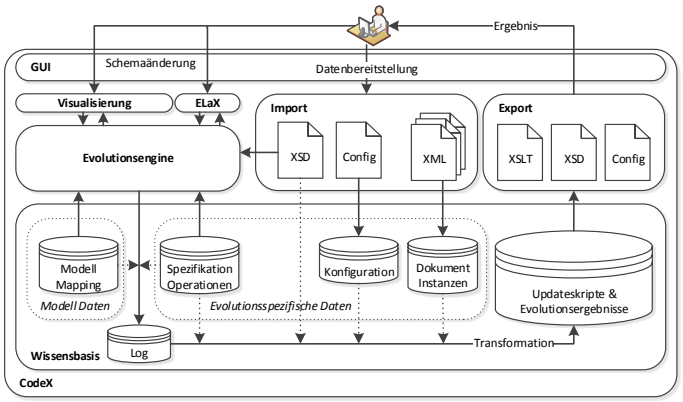

Abbildung 4.1: Komponentenmodell von CodeX entnommen aus [110]

Das Komponentenmodell zeigt, dass zur Schemaevolution ein XML-Schema benötigt wird. Dieses wird vom Nutzer mit den dazugehörigen XML-Dokumenten bereitgestellt. Aus dem XML-Schema wird dann das konzeptionelle Modell *Entity Model for XML-Schema* (EMX) generiert. Mit dem Entity Model for XML-Schema (EMX) arbeitet der Nutzer interaktiv, indem er Änderungsoperationen durchführt. Dabei werden automatische Vervollständigungen mit Hilfe von Default-Werten von CodeX angeboten.
Alle getätigten Änderungen werden in ELaX-Operationen übersetzt und in einer Log-Datei protokolliert. Nach Abschluss aller Modelländerungen wird die Log-Datei ausgewertet, um daraus die relevanten Schemaevolutionsschritte abzuleiten. Das XML-Schema wird daraufhin angepasst und eine Überprüfung (inkrementelle Validierung) der Schemaevolutionsschritte wird vorgenommen. Damit wird die Notwendigkeit der Anpassung eventuell nicht mehr schemakonformer XML-Dokumente ermittelt. Für die Anpassung werden XML-Updates generiert, die mit Hilfe von XSLT-Skripten umgesetzt werden (vgl. [38]).
Außerdem wird im Tool eine Kostenabschätzung der durchzuführenden Änderungsoperationen mit einem benutzerdefinierten Grenzwert abgeglichen. Wird dieser Wert überschritten, dann wird eine Versionierung des XML-Schemas vorgeschlagen, anstelle der Erzeugung und Durchführung eines XSLT-Skriptes. Die Import- und Export-Bausteine garantieren die Datenbereitstellung, sowie den Erhalt der Ergebnisse der Schemaevolution (vgl. [110]).

4.1.2 ELaX

Die Evolutionssprache *Evolution Language for XML-Schema* (ELaX) wird in [111] und [112] näher vorgestellt. Sie kann zum Einen zum formalen Ausdrücken und zum Anderen zum Durchführen von Anpassungsschritten bei der XML-Schemaevolution eingesetzt werden. Außerdem werden die ELaX-Operationen über eine Schnittstelle nach außen angeboten, d.h. die Ausdrücke können zum Einen exportiert und zum Anderen importiert werden, um die Schemaevolutionsschritte wiederzuverwenden. Aktuell wird die Funktionalität in das CodeX-Tool integriert.
Dabei werden XML-Schemata verwendet, die sich im Modellierungsstil *Garden of Eden* befinden, um die globalen Element- und Attributdeklarationen, sowie globalen Typdefinitionen auszunutzen. Um XML-Schema, die sich nicht in dieser Normalform befinden, wird eine Transformation durchgeführt, die in [71] ausführlich beschrieben wird. Aufbauend auf dieser Normalform werden die grundlegenden Operationen *add*, *delete* und *update* in einer an die Erweiterte Backus-Naur-Form (EBNF) angelehnte Notation definiert. Dabei passen die Einfüge-, Lösch- und Änderungsoperationen die verschiedenen Knotentypen im XML-Schema an, z.B. für Elemente, Attribute oder Inhaltsmodelle (Modellgruppen), siehe Theorem 4.1.

$$
\begin{aligned}
&elax ::= ((< add > | < delete > | < update >)";")+;\\
&add ::= "add"(< addannotation > | < addattributegroup > | < addgroup > | < addst >\\
&\qquad | < addct > | < addelement > | < addmodule > | < addconstraint >);\\
&delete ::= "delete"(< delannotation > | < delattributegroup > | < delgroup > | < delst >\\
&\qquad | < delct > | < delelement > | < delmodule > | < delconstraint >);\\
&update ::= "update"(< updannotation > | < updattributegroup > | < updgroup > | < updst >\\
&\qquad | < updct > | < updelement > | < updmodule > | < updconstraint > | < updschema >);
\end{aligned}
\tag{4.1}
$$

Ein ELaX-Ausdruck beginnt immer mit einem der Schlüsselwörter „add“, „delete“ oder „update“, gefolgt vom jeweiligen Knotentyp mit den jeweiligen knotentypischen Informationen. Am Ende eines ELaX-Ausdrucks steht dann ein „;“. Weiterhin zu beachten ist, dass wenn beispielsweise Element- oder Attributdeklarationen eingefügt werden, referenzierte, noch nicht vorhandene einfache oder komplexe Typdefinitionen vorher hinzugefügt werden müssen bzw. ein entsprechender ELaX-Ausdruck erzeugt werden muss. Eine komplette Übersicht der ELaX-Regeln befindet sich in [109].

4.2 Allgemeine Architektur

Zur Vorbereitung auf den Abschnitt 4.3 wird die allgemeine Architektur des entwickelten stand-alone Tools *CodeXMapper* vorgestellt, dargestellt in Abbildung 4.2.

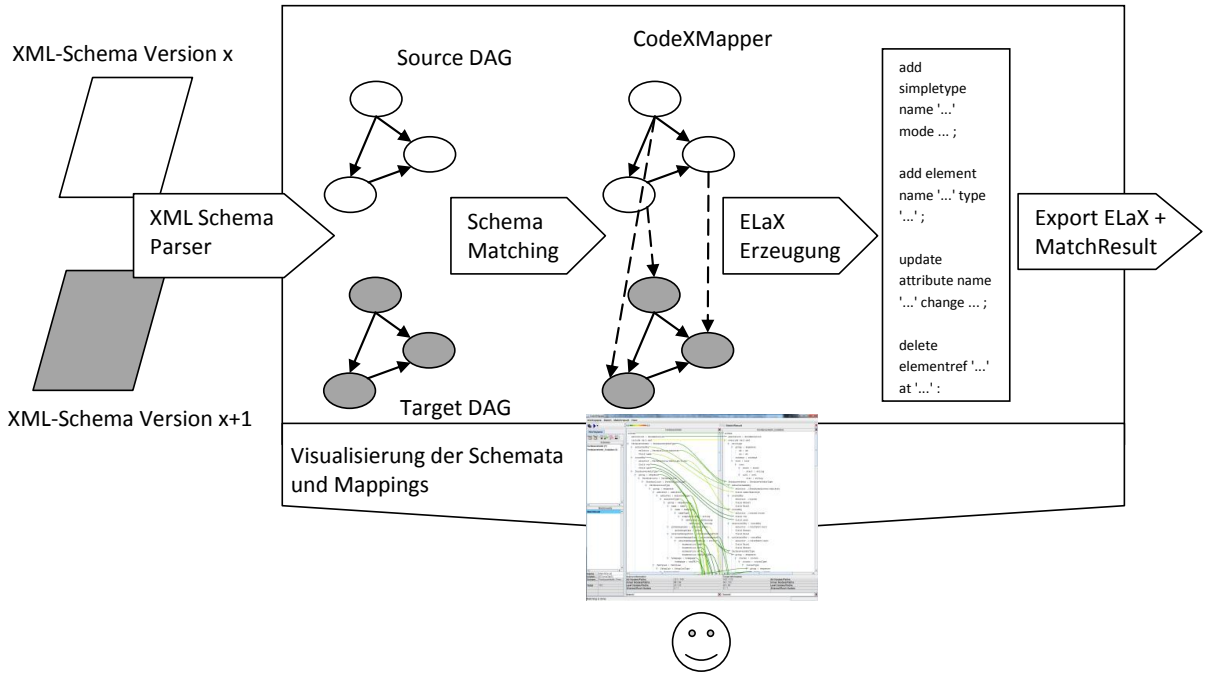

Abbildung 4.2: Allgemeine Architektur von CodeXMapper

Das Tool setzt die Schemaevolution mittels zweier Schemata um (vgl. [38]). Daraus folgt, dass dem Tool zunächst ein Quellschema und eine modifizierte Version dessen (Zielschema) zur Verfügung gestellt werden muss. Danach wird über einen XML-Schema Parser, die interne Datenstruktur der beiden importierten Schemata jeweils als Directed acyclic graph (DAG) erzeugt. Diese Datenstruktur wird verwendet um eine visuelle Baumstruktur der Schemata zu erzeugen, indem auf der linken Seite das Quell- und auf der rechten Seite das Zielschema dargestellt wird. Anschließend kann der Nutzer manuell oder über die Auswahl von verschiedenen Matchingalgorithmen automatisch Mappings zwischen den dargestellten Schemata erzeugen. Dabei lässt sich die Reihenfolge der Abarbeitung der Matchingalgorithmen (Workflow) durch den Nutzer manuell anpassen.
Auf Basis der erzeugten Mappings werden die Schemaevolutionsschritte entwickelt und in ELaX ausgedrückt, die für die Transformation des Quellschemas in das Zielschema notwendig sind. Anschließend können sowohl das Mappingergebnis (MatchResult), als auch das erzeugte ELaX-Skript jeweils in eine Textdatei exportiert werden. Weiterhin besteht die Möglichkeit vorhandene ELaX-Ausdrücke in das Tool zu importieren. Diese können dazu verwendet werden, um sie mit dem aktuell erzeugten Skript zu vergleichen, um so potentielle Unterschiede zwischen den verschiedenen Matching-Workflows zu ermitteln. Außerdem lassen sich Vergleiche zwischen mehreren Mappingergebnissen durchführen werden, die während mehrerer Durchläufe im Tool generiert wurden, z.B. den Durchschnitt zweier Mappingergebnisse.

4.3 Implementierung

Auf Basis der im Abschnitt 4.2 beschriebenen Architektur wird ein stand-alone Tool entwickelt, welches anschließend in CodeX integriert wird. Dabei wird zunächst in Abschnitt 4.3.1 die Implementierung des Matchingprozesses bzw. die dafür benötigte Integration von COMA näher erläutert. Anschließend wird in 4.3.2 die Erzeugung der ELaX-Ausdrücke aus den Matchingergebnissen erklärt. Letzendlich werden die notwendigen Anpassungsschritte in 4.3.3 erläutert, um das stand-alone Tool in CodeX zu integrieren. Als Programmiersprache wurde Java gewählt und die Entwicklungsumgebung Eclipse Juno (4.2) verwendet. Dabei unterscheiden sich die Implementierungen im stand-alone Tool und die Integration dessen im CodeX-Editor nicht sonderlich, weshalb hauptsächlich das stand-alone Tool *CodeXMapper* erwähnt wird. Dabei wird an gegebener Stelle die nötigen Erweiterungen aufgezeigt, um die Funktionalität des *CodeXMapper* in CodeX verfügbar zu machen.

4.3.1 Integration von COMA 3.0

Zur Umsetzung der Matchingfunktionalität wird das an Universität Leipzig entwicklete Matchingtool COMA 3.0 verwendet, aufgrund der im Abschnitt 3.3 genannten Gründen. Dabei wird nur eine Teilmenge der Funktionalitäten von COMA verwendet, z.B. werden Instanzbetrachtungen während des Matchingprozesses nicht weiter berücksichtigt, aufgrund der schemabasierten Fokussierung im Matchingprozess. Somit reduziert sich die Auswahl der benötigten Klassen auf ein Minimum, dargestellt in Abbildung 4.3.

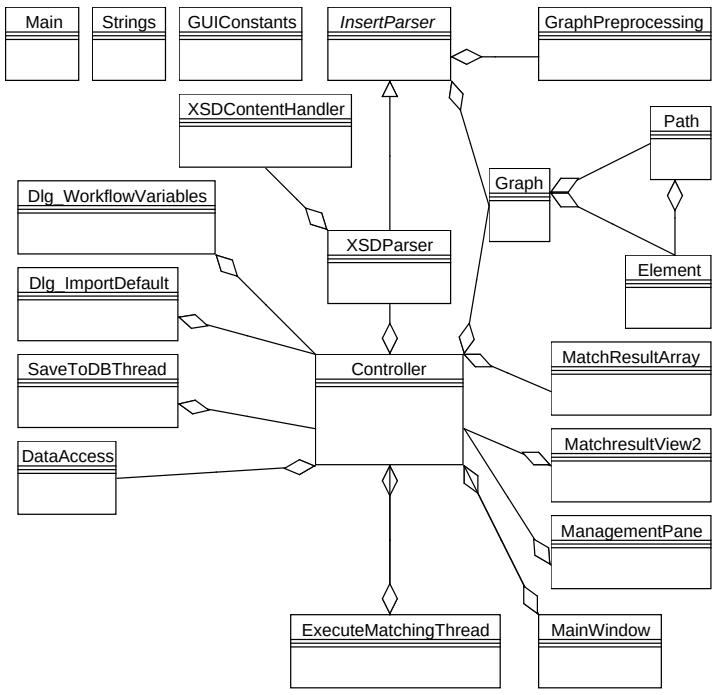

Abbildung 4.3: Klassendiagramm von CodeXMapper nach COMA-Integration

Das Klassendiagramm zeigt vor allem die Klassen, die zum Einen für die Funktionalität des *CodeXMapper* benötigt werden und zum Anderen im Zuge der Integration angepasst werden mussten. Als Einstiegspunkt wird in der Main-Klasse ein Objekt der Klasse Controller als Startinstanz erzeugt. Dadurch wird eine GUI angezeigt, die hauptsächlich über die MainWindow- und ManagementPane-Klasse generiert wird.

Ein versuchter Aufbau zu einer Datenbank wird unterbunden, da der *CodeXMapper* unabhängig von einer Datenbank laufen soll, um so die Portabilität zu erhöhen. In der MainWindow- und ManagementPane-Klasse werden Änderungen bezüglich der GUI vorgenommen, z.B. Weglassen eines Repository-Panels, welches die bereits gespeicherten Matchingergebnisse oder importierten Schemata anzeigt. Demnach ist ein Anpassen des Importdialogs (Dlg_ImportDefault-Klasse) notwendig, welcher das Quell- und/oder Zielschema sofort in den dafür vorgesehenden Bereich in der GUI darstellt. Für den Import von Quell- und/oder Zielschemata werden zusätzlich zum Einen die Klasse XSDParser und zum Anderen die GraphPreprocessing-Klasse angepasst.
Die Anpassungen im XSDParser sind notwendig, aufgrund einiger nicht berücksichtigter Schemakomponenten, wie z.B. Inhaltsmodelle (Modellgruppen), Constraints (Key, Unique, Keyref) oder das Schema-Element. Die Informationserweiterung der internen DAG-Darstellung führt dazu, dass die standardmäßige Graphdarstellung in der GUI angepasst werden muss, um die erweiterten Konzepte auch darstellen zu können. Hierzu wird auf Grundlage der im GraphPreprocessing vorhandenen Resolved-Darstellung eine überarbeitete Reduced-Repräsentation entwickelt. Die Resolved-Darstellung erzeugt aus dem geparsten XML-Schema für jede global vorkommende Schemakomponente eine Baumstruktur, dessen Wurzelknoten die globale Komponente darstellt. Dadurch wird eine sehr große und unübersichtliche Darstellung erzeugt, weshalb eine Reduzierung der Informationen erfolgt. Dafür wird das globale Element mit der größten Typhierarchie ausgewählt und als Baumstruktur erzeugt. Anschließend werden die verbliebenden globalen Schemakomponenten gegen die Baumstruktur geprüft, ob diese bereits vorhanden sind. Kommen die Komponenten nicht vor, so werden diese ebenfalls in die Baumstruktur eingefügt, sodass die gewünschte Anzeige in der Benutzeroberfläche erreicht wird, dargestellt in Abbildung 4.4.

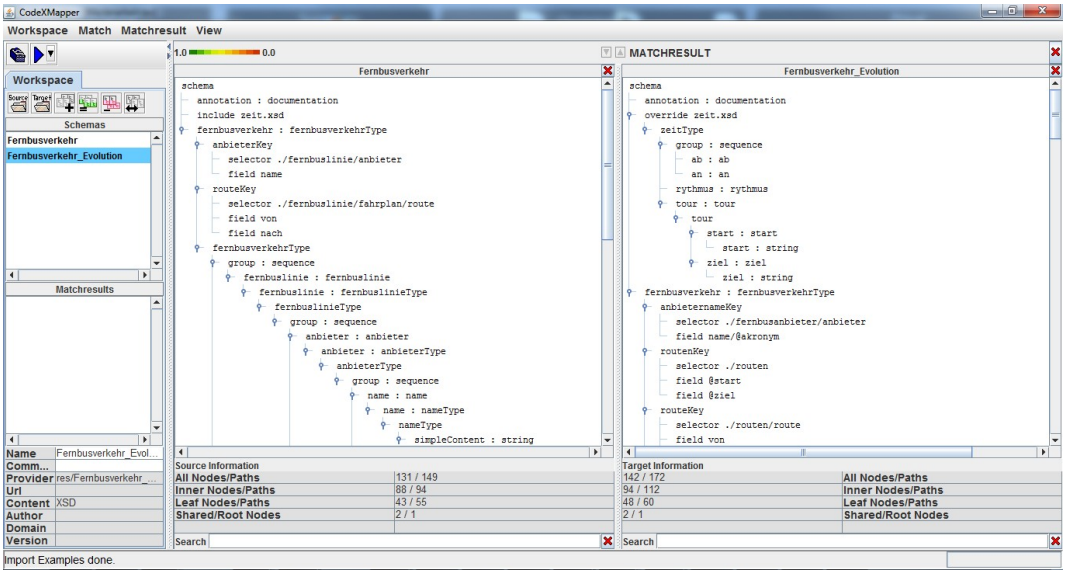

Abbildung 4.4: Angepasste COMA-GUI für CodeXMapper Teil 1/2

Nachdem die Graphen dem Nutzer korrekt angezeigt werden, kann die in COMA genutzte Matchingalgorithmen genutzt werden. Hierzu lassen sich verschiedene Workflows bzw. Strategien ausführen, die nach Benutzerwahl ausgeführt werden können. In der COMA-Version werden die vordefinierten Workflow-Variablen aus der Datenbank geladen, weshalb diese im *CodeXMapper* in die GUIConstants-Klasse geschrieben werden. Dabei wird der Dialog Dlg_WorkflowVariables dahingehend angepasst, dass zum Einen die Variablen aus den GUIConstants geladen werden und zum Anderen diese vom Nutzer auch geändert werden können.

Hierbei wird darauf hingewiesen, dass Änderungen auch beim nächsten Start der Anwendung wieder zur Verfügung stehen, jedoch jederzeit eine Rücksetzung auf den Ausgangszustand besteht. Standardmäßig ist die von COMA optimierte Strategie „ComaOptS“ ausgewählt, die eine Kombination von stringbasierten (Trigram) und graphbasierten (Paths, Leaves, Ancestor) Techniken ausführt. Über die ExecuteMatchingThread-Klasse wird der ausgewählte Matchingworkflow auf dem Quell- und Zielschema angewandt. Danach wird das Mappingergebnis über eingefärbte Linien dargestellt, welches die Klasse MatchresultView2 realisiert. Außerdem wird in der MatchResultView2-Klasse die Darstellung zum Einen um die Anzeige der Attribute der Schemakomponenten und zum Anderen um die Anzeige der später erzeugten ELaX-Ausdrücke erweitert, dargestellt in Abbildung 4.5.

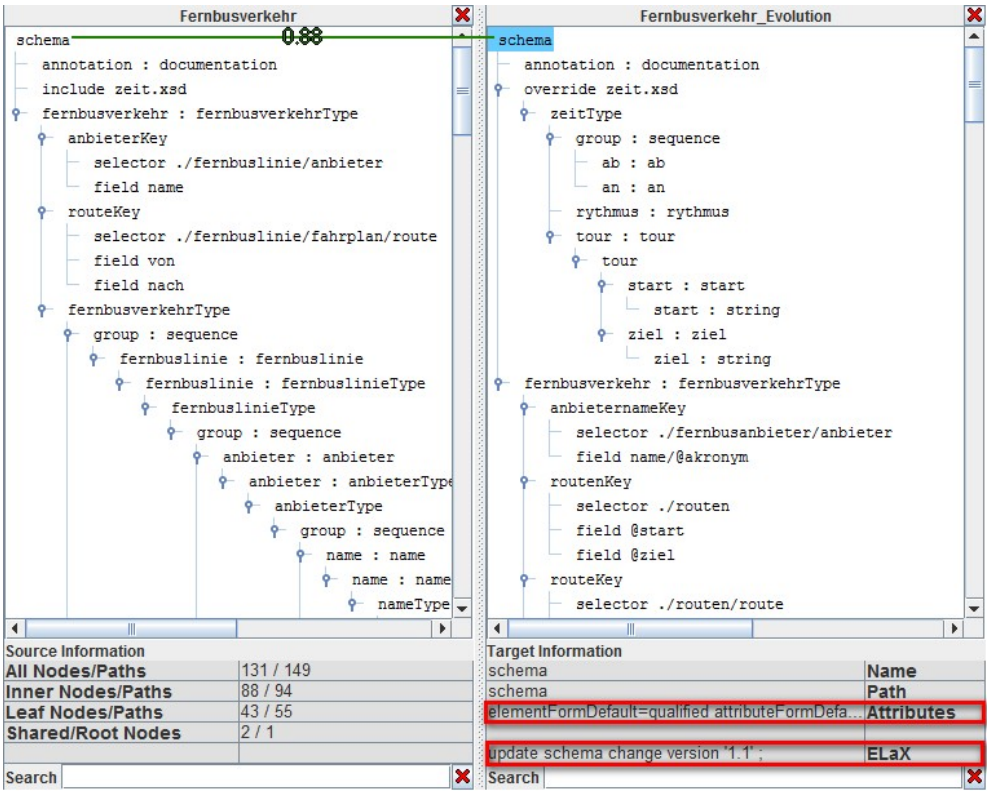

Abbildung 4.5: Angepasste COMA-GUI für CodeXMapper Teil 2/2

4.3.2 Übersetzung der Mappingergebnisse in ELaX-Ausdrücke

Im vorherigen Abschnitt 4.3.1 wurde die Grundlage geschaffen, um auf Basis eines Mappingergebnisses (MatchResult) die entsprechenden ELaX-Operationen zu erzeugen. Um die Funktionalität abzudecken werden die vier Klassen Attribute, ElaxPreprocessing, ElaxArray und CreateElax hinzugefügt, dargestellt in Abbildung 4.6.

Dabei erweitert die Attribute-Klasse den Informationsgehalt von Objekten der Element-Klasse. Diese Erweiterung ist notwendig, damit die getätigten Änderungen in der XSDParser-Klasse bzw. die benötigten Schemainformationen jeder Schemakomponente hinzugefügt und abgerufen werden können. Diese Art der Attribute enthalten neben den nach [144] gängigen Attribute einer jeden Schemakomponente, zusätzlich einen XPath-Ausdruck, der angibt an welcher Position sich die Schemakomponente befindet.

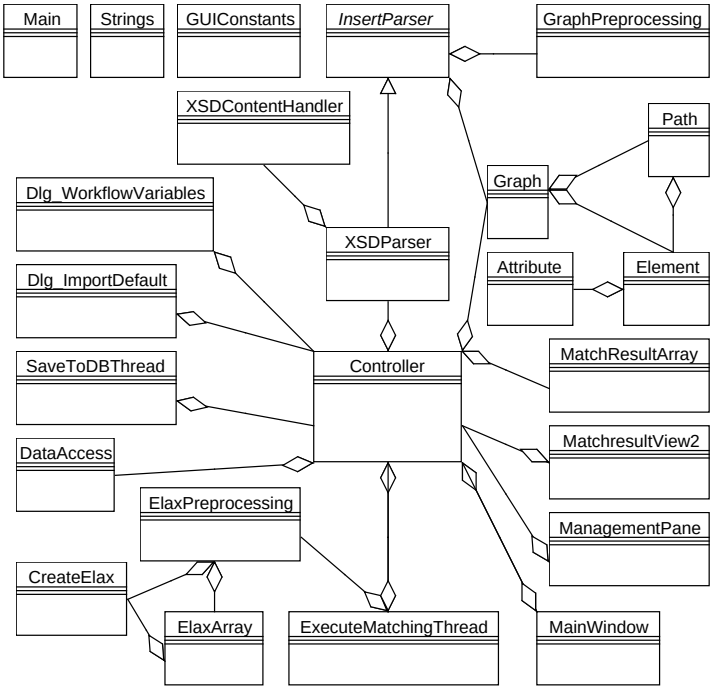

Abbildung 4.6: Klassendiagramm von CodeXMapper nach der ELaX-Erweiterung

Dabei orientiert sich der Pfadausdruck an dem *locator*-Ausdruck, siehe Theorem 4.2.

$$\begin{aligned} &locator ::= < xpathexpr > |emxid; \\ &xpathexpr ::= (\text{"/"}(\text{"."}|(\text{"}node()\text{"}|(\text{"}node()[@name = \text{"}NCNAME\text{"}]\text{"}))(\text{"}[\text{"}INT\text{"}]\text{"})?))+; \end{aligned} \tag{4.2}$$

Somit wird im *CodeXMapper* beispielsweise für eine Moduldeklaration, welche immer direkt unter dem Schemaelement (xs:schema) eingefügt werden muss, folgender XPath-Ausdruck erzeugt:

$$/node()/node()[1] \tag{4.3}$$

Dabei ermöglicht die Positionsangabe in Theorem 4.3 eine sichere Unterscheidung zwischen mehreren gleichnamigen Moduldeklarationen, z.B. zwischen zwei include-Modulen. Solch eine Positionsangabe wird neben Moduldeklarationen für die Facetten „enumeration“, „pattern“ und „assertion“, für die Assert-Elemente, sowie für die Field- und Selector-Elemente in den Constraintdeklarationen unterstützt. Die restlichen Facetten dürfen laut Definition [19] nicht mehr als einmal im Schema auftreten, sodass diese eindeutig sind. Aufgrund des Modellierungsstils *Garden of Eden* sind die globalen Element-, Attribut-, Attributgruppen-, Typ- und Constraintdeklarationen eindeutig über ihren Namen definiert, sodass sich folgender XPath-Ausdruck ergibt:

$$/node()/node()[@name = \text{"}Deklarationsname\text{"}] \tag{4.4}$$

Die restlichen Schemakomponenten, z.B. Anmerkungen werden über den folgenden XPath-Ausdruck adressiert:

$$/node()/. \tag{4.5}$$

Dabei beginnt die absolute Adressierung immer vom Schemaelement (xs:schema) aus und endet in der jeweiligen Schemakomponente. Außerdem enthalten die Attribute Informationen, z.B. über die Art (restriction, unique, list) und/oder vorkommende Facetten in einer einfachen Typdefinition. Damit können später die korrekten ELaX-Ausdrücke erzeugt werden.
Die Klasse ElaxPreprocessing wird benötigt, um aus dem erzeugten MatchResult die benötigten Quell- und Zielschemakomponenten zu extrahieren, die notwendig für die ELaX-Erzeugung sind. Dazu werden zunächst in der ExecuteMatchingThread-Klasse alle Quell- und Zielschemakomponenten des aktuell erzeugten MatchResults an die ElaxPreprocessing-Klasse übergeben. Anschließend wird über den von der ExecuteMatchingThread-Klasse ausgehenden Befehl „createElax“ ein Prozess der ElaxPreprocessing-Klasse gestartet, dargestellt in Abbildung 4.7.

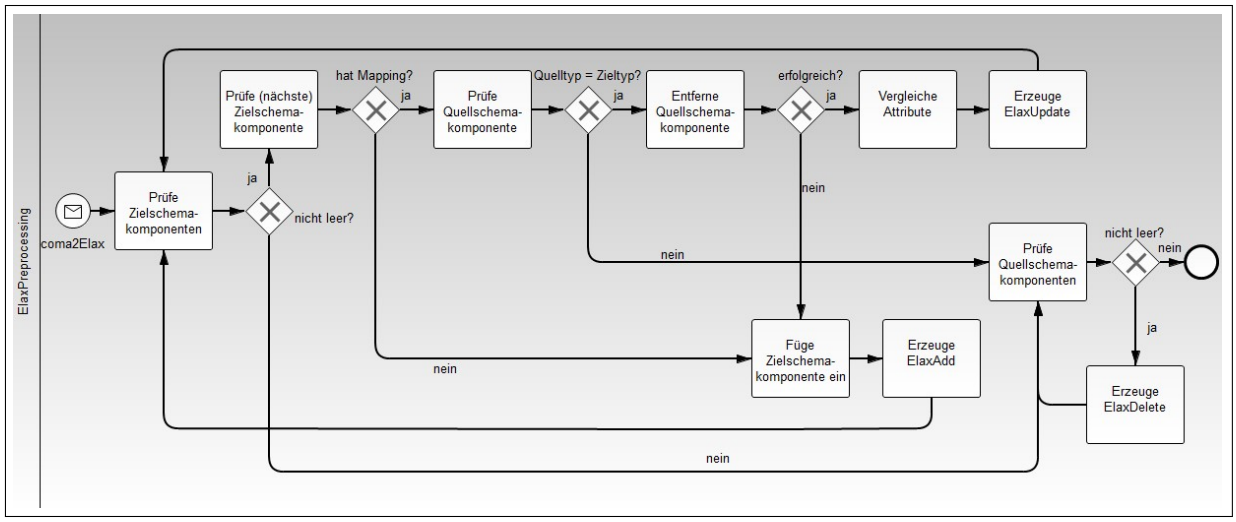

Abbildung 4.7: Ablaufprozess der ElaxPreprocessing-Klasse

Dieser Prozess wertet zunächst zu jeder Zielschemakomponente (Target) die gemappten Quellschemakomponenten (Sources) aus. Die Menge der Sources wird dann weiter untersucht, d.h. als Erstes wird geprüft, ob gleiche Schemakomponenten miteinander gematcht werden, z.B. nur Module oder Constraints untereinander. Anschließend wird die betrachtete Schemakomponente aus der Menge aller Quellschemakomponenten entfernt und bei erfolgreichem Durchführen werden die Attribute der Quellschemakomponente mit dem Target abgeglichen. Dabei werden identische Attribute nicht weiter betrachtet bzw. gelöscht. Wenn die Attributwerte sich unterscheiden werden ELaX-Update-Operationen erzeugt mit Hilfe der CreateElax-Klasse. Diese Operationen werden dann einer Instanz der ElaxArray-Klasse hinzugefügt.
Wenn eine Zielschemakomponente auf eine bereits schon gelöschte Quellschemakomponente verweist, dann wurde diese Quellschemakomponente schon mit einem ELaX-Ausdruck versehen und kann kein zweites Mal in eine Transformation eingebettet sein. Deshalb wird die referenzierende Zielschemakomponente als neue Komponente dem Quellschema hinzugefügt, indem eine entsprechende ELaX-Add-Operation erzeugt wird, die ebenfalls obiger, beim Update erwähnter Instanz der ElaxArray-Klasse hinzugefügt wird. Verweist eine Zielschemakomponente auf gar keine Quellschemakomponente, so wird ebenfalls eine Einfügeoperation erzeugt und der ElaxArray-Instanz hinzugefügt. Zum Schluss werden für alle übriggebliebenden nicht gemappten Quellschemakomponenten ELaX-Delete-Operationen erzeugt, die dann der ElaxArray-Instanz hinzugefügt werden.
Des Weiteren wird in der MatchResultView2-Klasse die Logik für die manuelle Erzeugung und Entfernung von Korrespondenzen hinzugefügt, die neue bzw. bestehende ELaX-Operationen hinzufügt und/oder löscht. Wenn der Nutzer ein neues Mapping hinzufügt, wird zwischen diesen beiden Objekten ein erneuter ELaX-Übersetzungsvorgang ausgelöst und die ELaX-Ausdrücke überarbeitet.

Somit werden die Lösch- und Einfügeoperationen der betreffenden Komponenten aus dem ELaX-Skript entfernt und die neue Update-Operation hinzugefügt. Analog wird beim Löschen eines oder aller Mappings, die entsprechend gespeicherten ELaX-Ausdrücke entfernt.
Die ElaxArray-Klasse erzeugt eine Instanz, die wie angedeutet alle Elax-Ausdrücke sammelt, die beim Übersetzungsvorgang des Mappingergebnisses erzeugt werden. Ist der Übersetzungsprozess abgeschlossen wird die ElaxArray-Instanz ausgewertet. Dabei wird eine bestimmte Reihenfolge erstellt, beginnend mit dem Einfügen aller built-in Typdefinitionen, die nicht im Quellschema vorkommen, gefolgt von allen einfachen Typdefinitionen. Anschließend werden die komplexen Typdefinitionen eingefügt, aufgrund der Tatsache, dass diese auf bereits bestehende einfache Typdefinitionen referenzieren könnten. Danach werden alle restlichen Einfügeoperationen eingefügt, da nunmehr gewährleistet ist, dass z.B. Element- und Attributdeklarationen auf bereits bestehende Typdefinitionen verweisen. Dann folgen die Menge aller Update-Operationen und zum Schluss alle Löschoperationen. Letztendlich wird in der MainWindow-Klasse ein neuer Menüpunkt „Matchresult ELaX“ unter dem Menüeintrag „View“ eingefügt, welcher die erzeugten ELaX-Ausdrücke in der so generierten Reihenfolge enthält.
Die Logik zur Erzeugung der Add-, Update- und Delete-Operationen befindet sich in der CreateElax-Klasse. Hier werden die ELaX-Statements für jede einzelne Schemakomponente erzeugt. Dabei werden initial die built-in Typdefinitionen eingefügt, die nicht im Quell- aber Zielschema enthalten sind. Diese sind für die interne Datenverwaltung von CodeX wichtig, weil meistens nicht alle built-in Datentypen benötigt werden. Anschließend werden je nach übergebenen Modus und Schemakomponente die jeweiligen Einfüge-, Anpassungs- und Löschoperationen erzeugt. Eine komplette Übersicht der ELaX-Übersetzungsregeln befindet sich wegen des großen Umfangs im Anhang C. Die dabei aufgestellten Mappingbeziehungen $V \leftrightarrow V$, $\neg V \leftarrow V$ und $V \rightarrow \neg V$ bedeuten, dass die Schemakomponenten und/oder Attribute in Quell-und Zielschema, nur im Zielschema oder nur im Quellschma vorhanden sind. Dabei sortiert der in Abbildung 4.7 vorgestellte Prozess bereits die Schemakomponenten in die eingeführten Mappingbeziehungen bzw. gibt die ELaX-Erzeugungsart an. Deshalb wird sich in den nachfolgenden Unterabschnitten auf die Erläuterung der einzelnen Schemakomponenten, sowie deren Matchkardinalitäten beschränkt. Grundsätzlich werden die 1:1- und die n:m-Matchkardinalitäten angeben, mit der Berücksichtigung der 1:n, sowie n:1 Fälle in den n:m-Kardinalitäten.

Schemadeklaration

Die Schemadeklaration (xs:schema) kommt in der Quell- als auch im Zielschema nur ein einziges Mal vor, sodass die Matchkardinaliät immer bei 1:1 liegt. Somit wirken sich die Einfüge- und Löschoperationen nur auf die Attribute aus, weshalb sich innerhalb der Schemadeklaration alles mit der Update-Operation ausdrücken lässt, dargestellt in Abbildung 4.8.

Schema-komponente	Mapping-Beziehung	Match-kardinalität	Mapping-Regel	ELaX-Operation
1. xs:schema	1.1 V_s ↔ V_t	1:1	1.1.1 targetNamespace_s != targetNamespace_t	update schema change targetnamespace '*targetNamespace_t*';
			1.1.2 targetnamespaceprefix_s != targetnamespaceprefix_t	update schema change targetnamespaceprefix '*targetnamespaceprefix_t*';
			1.1.3 xml:lang_s != xml:lang_t	update schema change language '*xml:lang_t*';
			1.1.4 version_s != version_t	update schema change version '*version_t*';
			1.1.5 elementFormDefault_s != elementFormDefault_t	update schema change elementform '*elementFormDefault_t*';
			1.1.6 attributeFormDefault_s != attributeFormDefault_t	update schema change attributeform '*attributeFormDefault_t*';
			1.1.7 finalDefault_s != finalDefault_t	update schema change finaldefault '*finalDefault_t*';
			1.1.8 id_s != id_t	update schema change id '*id_t*';
			1.1.9 defaultAttributes_s != defaultAttributes_t	update schema change defaultattribute '*defaultAttributes_t*';
			1.1.10 xpathDefaultNamespace_s != xpathDefaultNamespace_t	update schema change xpathdefaultnamespace '*xpathDefaultNamespace_t*';

Abbildung 4.8: ELaX-Übersetzungsregeln für Schemadeklaration abgeleitet aus C

Demnach können mit ELaX die Schemaattribute „targetNamespace“, „xml:lang“, „version“, „elementFormDefault“, „attributeFormDefault“, „finalDefault“, „id“, „defaultAttributes“ und „xpathDefaultNamespace“ hinzugefügt, gelöscht und/oder geändert werden (siehe Operationen *1.1.1* und *1.1.3-1.1.10*). Zusätzlich besteht die Möglichkeit bei Festlegung eines Präfixes für das targetNamespace-Attribut dieses Präfix über „targetnamespaceprefix“ zu ändern bzw. hinzuzufügen oder zu löschen (siehe Operation *1.1.2*).

Moduldeklarationen

Die Moduldeklarationen teilen sich auf in import, redefine, include und override. Für alle vier Arten lassen sich die schemaLocation- und id-Attribute ändern (siehe Operationen *2.1.2* und *2.1.3*). Bei dem import-Modul kann zuätzlich das namespace-Attribut verändert werden (siehe Operation *2.1.1*), sodass sich die in in Abbildung 4.9 dargestellten ELaX-Update-Operationen ergeben.

<table>
<tr><th>Schema-komponente</th><th>Mapping-Beziehung</th><th>Match-kardinalität</th><th colspan="3">Mapping-Regel</th><th colspan="2">ELaX-Operation</th></tr>
<tr><td rowspan="7">2. xs:import, xs:redefine, xs:include, xs:override</td><td rowspan="7">2.1 V_s ↔ V_t</td><td rowspan="3">1:1</td><td colspan="3">2.1.1 import_s_namespace != import_t_namespace</td><td colspan="2">update module at 'import_s_locator' change mode import with namespace 'import_t_namespace';</td></tr>
<tr><td colspan="3">2.1.2 (import_s|redefine_s|include_s|override_s)_schemaLocation != (import_t|redefine_t|include_t|override_t)_schemaLocation</td><td colspan="2">update module at '(import_s|redefine_s|include_s| override_s)_locator' change from '(import_t|redefine_t| include_t|override_t)_schemaLocation';</td></tr>
<tr><td colspan="3">2.1.3 (import_s|redefine_s|include_s|override_s)_id != (import_t|redefine_t|include_t|override_t)_id</td><td colspan="2">update module at '(import_s|redefine_s|include_s| override_s)_locator' change id '(import_t|redefine_t| include_t|override_t)_id';</td></tr>
<tr><td rowspan="4">n:m</td><td rowspan="4">2.1.4 M[n] = {(import_s|redefine_s|include_s|override_s)_1,..., (import_s|redefine_s|include_s|override_s)_n}, N[m] = {(import_t|redefine_t|include_t|override_t)_1,..., (import_t|redefine_t|include_t|override_t)_m}</td><td rowspan="4">2.1.4 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]</td><td>2.1.4.1 n = m</td><td colspan="2">loop n: (import_s|redefine_s|include_s|override_s)_n ↔ (import_t|redefine_t|include_t|override_t)_n (analog zu 2.1.1, 2.1.2 und 2.1.3)</td></tr>
<tr><td>2.1.4.2 n > m</td><td colspan="2">analog zu 2.1.4.1 (P[m]!=∅, loop m, O[n]\{(import_s|redefine_s|include_s|override_s)_m})
loop n: delete module at '(import_s|redefine_s|include_s| override_s)_n_locator';</td></tr>
<tr><td rowspan="2">2.1.4.3 n < m</td><td colspan="2">analog zu 2.1.4.1 (O[n]!=∅, P[m]\{(import_t| redefine_t| include_t|override_t)_n})</td></tr>
<tr><td>loop n: add module from '(import_t|redefine_t| include_t|override_t)_n_schemaLocation' [...] id = '(import_t|redefine_t| include_t| override_t)_n_id';</td><td>[...]: mode import with namespace 'import_t_n_namespace'
[...]: mode redefine (<addst>|<addct>|<addattributegroupdef>)
[...]: mode include
[...]: mode override(<addst>|<addct>|<addattributegroupdef>|<addelementdef>|<addattribute>)</td></tr>
</table>

Abbildung 4.9: ELaX-Übersetzungsregeln für Moduldeklarationen abgeleitet aus C

Werden mehrere Mappingbeziehungen ausgewertet, so werden zunächst über die Mengen O[n] und P[m] die vorhandenen identischen Moduldeklarationen in Quell- und Zielschemata entfernt (siehe Operation *2.1.4*). Wenn die resultierenden Mengen die gleiche Anzahl von Moduldeklarationen haben, werden die bereits angesprochenden Update-Operationen durchgeführt (siehe Operation *2.1.4.1*). Ansonsten werden nicht im Zielschema vorkommende Moduldeklarationen im Quellschema gelöscht (siehe Operation *2.1.4.2*), sowie diejenigen nur im Zielschema vorkommenden Moduldeklarationen eingefügt (siehe Operation *2.1.4.3*). Beim Einfügen beispielsweise des override-Moduls findet eine Schachtelung von ELaX-Ausdrücken statt, sodass die CreateElax-Klasse sich selbst nochmal aufruft, um die nötigen Teilausdrücke zu erzeugen. Hierfür werden sowohl im redefine-, als auch im override-Modul zusätzliche Attributwerte gehalten, wie das Vorkommen von einfachen und komplexen Typdefinitionen. Der folgende Ausdruck 4.6 veranschaulicht die zusätzlichen Informationen am Beispiel eines override-Moduls.

$$schemaLocation = zeit.xsd\ mode = override\ complexType.1 = name\_zeitType \tag{4.6}$$

Hierbei wird deutlich, dass im Falle einer Einfügeoperation die SchemaLocation und der Mode zum Einen das einzubettende Schema und zum Anderen den Modultyp angeben. Der complexType.1 wird einer neuen CreateElax Instanz übergeben, indem ein Split am „.“ durchgeführt wird, um den Typ des zu erzeugenden ELaX-Teilausdrucks festzulegen. Der Attributwert wird am „_“ aufgetrennt, sodass der Name mit dem Wert zeitType der ELaX-Add-Operation übergeben werden kann. Damit ausgeschlossen wird, dass kein Schlüsselwort „add“ im Teilausdruck vorkommt wird ein entsprechender Nullwert übergeben, der dann das Vorkommen eines solchen Schlüsselwortes unterbindet.
Daneben können Moduldeklarationen im Quell- und Zielschema vorkommen, die sich aber vom Typ her unterscheiden. Grundsätzlich wird in solchen Fällen eine Update-Operation generiert, sofern derselbe Inhalt (Content) zur Verfügung steht. Mit Content sind hier die strukturellen Übereinstimmungen nach [144] gemeint, sodass bei einem Konflikt das Quellmodul gelöscht und das neue eingefügt wird.

Anmerkungen

Anmerkungen können überall im Schema vorkommen, wobei der Kommentar im *CodeXMapper* bedingt durch COMA im Attribut „Comment“ der Element-Klasse festgehalten wird. Die möglichen Update-Operationen werden in Abbildung 4.10 zusammengefasst.

Schema-komponente	Mapping-Beziehung	Match-kardinalität	Mapping-Regel			ELaX-Operation
3. xs:annotation	3.1 V_s ↔ V_t	1:1	3.1.1 appinfo_s != appinfo_t			update annotation at '*annotation_s_locator*' change appinfo '*appinfo_t*';
			3.1.2 documentation_s != documentation_t			update annotation at '*annotation_s_locator*' change documentation '*documentation_t*';
			3.1.3 id_s != id_t			update annotation at '*annotation_s_locator*' change id '*id_t*';
			3.1.4 locator_s != locator_t			update annotation at '*annotation_s_locator*' change move into '*annotation_t_locator*';
		n:m	3.1.5 M[n]={annotation_s_1,..., annotation_s_n}, N[m]={annotation_t_1,..., annotation_t_m}	3.1.5 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	3.1.5.1 n = m	loop n: annotation_s_n ↔ annotation_t_n (analog zu 3.1.1-3.1.4)
					3.1.5.2 n > m	analog zu 3.1.5.1 (P[m]!=∅, loop m, O[n]\{annotation_s_m})
						loop n: delete annotation at '*annotation_s_n_locator*';
					3.1.5.3 n < m	analog zu 3.1.5.1 (O[n]!=∅, P[m]\{annotation_t_n})
						loop n: add annotation appinfo '*appinfo_t_n*' documentation '*documentation_t_n*' id '*id_t_n*' in '*annotation_t_n_locator*';

Abbildung 4.10: ELaX-Übersetzungsregeln für Anmerkungen abgeleitet aus C

Dabei lässt sich neben der id zum Einen die appInfo, sowie documentation und zum Anderen die Position verändern (siehe Operationen *3.1.1-3.1.4*). Werden die appInfo und documentation verändert bedeutet dies immer, dass der Kommentar bzw. der Elementinhalt verändert wird. Unterscheidet sich die Position der Quell- und Zielannotation, dann kann mit Hilfe eines bestimmten ELaX-Konstruktes die Annotation verschoben werden, siehe Theorem 4.7.

$$update\ annotation\ at\ '/node()/.'\ change\ appinfo\ 'Stand:\ 2013'\ move\ into\ '/node()/node()/.'\ ; \quad (4.7)$$

In diesem Beispiel wird zum Einen der Kommentarinhalt der Quellannotation geändert und zum Anderen wird über „move into“ die neue Position der Annotation angegeben. Sind mehrere Anmerkungen bzw. Annotationen miteinander verknüpft, so werden die genannten Update-Operationen ausgeführt, sofern diese zahlenmäßig in Quell- und Zielschema übereinstimmen. Ansonsten werden fehlende Annotationen im Quell- oder Zielschema in das evolutionierte Schema hinzugefügt bzw. aus dem Quellschema entfernt.

Einfache Typdefinitionen

Die einfachen Typdefinitionen treten im Schema immer global auf, aufgrund des Modellierungsstils *Garden of Eden*, weshalb die in Abbildung 4.11 verwendeten Update-Operationen möglich sind.
Als Erstes lassen sich in den Operationen *4.1.1*, *4.1.5* und *4.1.6* die name-, final- und/oder id-Attribute ändern. Weiterhin können in den Operationen *4.1.2-4.1.4* die Typen Liste (list), Vereinigung (union) und Einschränkung (restriction) angepasst werden. Demnach besteht für eine list die Möglichkeit die verwendete einfache Typdefinition im itemType anzupassen. Ähnlich verhält sich die Anpassung bei der union, bei der die einzelnen einfachen Typdefinitionen der Vereinigungsmenge (memberTypes) hinzugefügt oder entfernt werden können. Bei der restriction können neben des Basistyps, die Menge der verwendeten Facetten in der Quell- und Zielrestriction abgeglichen und gegebenenfalls angepasst werden. Dazu sind drei Varianten möglich, wobei das Modifizieren, Einfügen und Löschen gewährleistet wird (siehe Operationen *4.1.4.2.1-4.1.4.2.3*). Die folgenden Ausdrücke 4.8 und 4.9 veranschaulichen zum Einen die dafür erforderlichen Informationen und zum Anderen die resultierende ELaX-Update-Operation.

$$\begin{aligned} name = buslinieType\ restriction = xs:string\ enumeration.1 = value\_Wörlitz\ Tourist \\ |locator\_/node()/node()[@name = "buslinieType"]/node()/node()[1] \end{aligned} \quad (4.8)$$

$$update\ simpletype\ name\ 'buslinieType'\ change\ mode\ restriction\ of\ 'xs:string'\ modify\ enumeration\ 'Wörlitz-Tourist'\ at'/node()/node()[@name = "buslinieType"]/node()/node()[1]'; \quad (4.9)$$

Schema-komponente	Mapping-Beziehung	Match-kardinalität	Mapping-Regel				ELaX-Operation
4. xs:simple Type	4.1 V_s ↔ V_t	1:1	4.1.1 name_s != name_t				update simpletype name '*name_s*' change name '*name_t*';
			4.1.2 list_s != list_t				update simpletype name '*simpletype_s_name*' change mode list '*list_t_itemType*';
			4.1.3 M[n]={union_s_1,..., union_s_n}, N[m]={union_t_1,..., union_t_n}	4.1.3 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]			O[n]!=∅: update simpletype name '*simpletype_s_name*' change mode union *loop n:* remove '*union_s_n_memberTypes*';
							P[m]!=∅: update simpletype name '*simpletype_s_name*' change mode union *loop n:* insert '*union_t_m_memberTypes*';
			4.1.4 restriction_s != restriction_t	4.1.4.1 base_s != base_t			update simpletype name '*simpletype_s_name*' change mode restriction of '*base_t*';
				4.1.4.2 M[n] = {facet_s_1,..., facet_s_n}, N[m] = {facet_t_1,..., facet_t_n}	4.1.4.2 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	4.1.4.2.1 n = m	update simpletype name '*simpletype_s_name*' change mode restriction of '*base_s*' *loop n:* modify '*facet_t_n*' at '*facet_s_n_locator*';
						4.1.4.2.2 n > m	analog zu 4.1.4.2.1 (P[m]!=∅, loop m, O[n]\{facet_s_m})
							update simpletype name '*simpletype_s_name*' change mode restriction of '*base_s*' *loop n:* remove '*facet_s_n*' at '*facet_s_n_locator*';
						4.1.4.2.3 n < m	analog zu 4.1.4.2.1 (O[n]!=∅, P[m]\{facet_t_n})
							update simpletype name '*simpletype_s_name*' change mode restriction of '*base_s*' *loop n:* insert '*facet_t_n*' at '*facet_t_n_locator*';
			4.1.5 final_s != final_t				update simpletype name '*simpletype_s_name*' change final '*final_t*';
			4.1.6 id_s != id_t				update simpletype name '*simpletype_s_name*' change id '*id_t*';
		n:m	4.1.7 M[n]={simpletype_s_1,..., simpletype_s_n}, N[m]={simpletype_t_1,..., simpletype_t_m}	4.1.7 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]		4.1.7.1 n = m	loop n: simpletype_s_n ↔ simpletype_t_n (analog zu 4.1.1-4.1.6)
						4.1.7.2 n > m	analog zu 4.1.7.1 (P[m]!=∅, loop m, O[n]\{simpletype_s_m})
							loop n: delete simpletype name '*simpletype_s_n_name*';
						4.1.7.3 n < m	analog zu 4.1.7.1 (O[n]!=∅, P[m]\{simpletype_t_n})
							loop n: add simpletype name '*simpletype_t_n_name*' mode '*simpletype_t_n_mode*' final = *final_t_n* id = '*id_t_n*';

Abbildung 4.11: ELaX-Übersetzungsregeln für einfache Typdefinitionen abgeleitet aus C

Demnach sind für einfache Datentypen jeweils die Typdefinition mit den entsprechenden Werten vorzuhalten, z.B. „restriction“ mit dem base-Attributwert „xs:string“. Weiterhin werden die Facetten und Assertions mit dem Facetten-Namen beschrieben, ergänzt um eine Zahl, die für die interne Unterscheidung benötigt wird. Außerdem wird im Attributwert der jeweiligen Facette oder Assertion zum Einen die Facetten-Attribute mit ihren Werten festgehalten und zum Anderen der locator, um die entsprechende Facette oder Assertion im Quellschema anzusprechen bzw. beim Einfügen die Position im Zielschema anzugeben. Dabei entspricht die locator-Angabe der Definition 4.2.
Weiterhin existieren die Möglichkeiten zur Überführung von einfachen Typdefinitionen, wenn sich ihre Typen unterscheiden, z.B. von einer Liste zu einer Vereinigung. Dabei soll dann eine Aktualisierung vorgenommen werden, wenn zum Einen der Inhalt (Content) und zum Anderen der Wertebereich (Range) übereinstimmt. Dabei spiegelt der Content die strukturellen Ähnlichkeiten (vgl. [144]) wieder. Dabei wird angenommen, dass der Content ebenfalls zwischen den Attributwerten der verschiedenen Typen angenommen wird bzw. base-, itemType- und memberTypes-Attribute äquivalent zueinander sind. Wenn also eine Einschränkung einen built-in Datentyp xs:string (Content) auf einen bestimmten Wertebereich (Range) einschränkt, dann würde eine Typänderung zu einer Liste zwar den Content übernehmen, aber nicht zwangsläufig den Wertebereich. Somit müsste die Update-Operation, gewährleisten, dass der entsprechende Wertebereich erreicht wird, indem zusätzliche Typdefinitionen eingefügt werden bzw. bestehende wiederverwendet und anschließend referenziert werden. Ansonsten werden nicht mehr benötigte einfache Typdefinitonen aus dem Quellschema gelöscht und erforderliche Typdefinitionen aus dem Zielschema eingefügt (siehe Operationen *4.1.7.1-4.1.7.3*).

Komplexe Typdefinitionen

Die komplexen Typdefinitionen sind ebenfalls, wie die einfachen Typdefinitonen, global definiert und besitzen die in Abbildung 4.12 dargestellten Änderungsoperationen.
Hierbei passen die Operationen *5.1.1-5.1.3*, *5.1.6* und *5.1.8* potentiell vorkommende Attribute in der komplexen Typdefinition an, z.B. das name- oder mixed-Attribut. Weiterhin besteht die Möglichkeit die Basis der Einschränkungen bzw. Erweiterungen innerhalb der simpleContent- und complexContent-Definitonen zu ändern (siehe Operation *5.1.4*). Dabei können die Einschränkungen in simpleContent-Definitionen wiederrum Facetten und Assertions enthalten, die analog zu 4.8 und 4.9 gespeichert und aktualisiert werden. Diesselbe Vorgehensweise wird auch bei auftretenden Assert-Deklarationen angewandt (siehe Operation *5.1.8*). Die Typänderung der simpleContent- und complexContent-Definitionen ist äquivalent zu den einfachen Typdefinitionen, d.h. der Inhalt (vgl. [144]) und Wertebereich wird als Kriterum angegeben, um eine entsprechende Update-Operation durchzuführen.

Schema-komponente	Mapping-Beziehung	Match-kardinalität	Mapping-Regel				ELaX-Operation
5. xs:complex Type	5.1 V_s ↔ V_t	1:1	5.1.1 name_s != name_t				update complextype name '*name_s*' change name '*name_t*';
			5.1.2 mixed_s != mixed_t				update complextype name '*complextype_s_name*' change mixed '*mixed_t*';
			5.1.3 final_s != final_t				update complextype name '*complextype_s_name*' change final '*final_t*';
			5.1.4 (simple|complex)Content_s_base != (simple|complex)Content_t_base				update complextype name '*complextype_s_name*' change mode (extension_cc|extension_sc| restriction_cc|restriction_sc) with base '*simpleContent_t_base*';
			5.1.5 restriction_sc_s != restriction_sc_t	5.1.5.1 M[n] = {facet_s_1,..., facet_s_n}, N[m] = {facet_t_1,..., facet_t_n}	5.1.5.1 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	5.1.5.1.1 n = m	update complextype name '*complextype_s_name*' change mode restriction_sc *loop n:* modify '*facet_t_n*' at '*facet_s_n_locator*' with base '*restriction_sc_s base*';
						5.1.5.1.2 n > m	analog zu 5.1.5.1.1 (P[m]!=∅, loop m, O[n]\{facet_s_m})
							update complextype name '*complextype_s_name*' change mode restriction_sc *loop n:* remove '*facet_s_n*' at '*facet_s_n_locator*' with base '*restriction_sc_s_base*';
						5.1.5.1.3 n < m	analog zu 5.1.5.1.1 (O[n]!=∅, P[m]\{facet_t_n})
							update complextype name '*complextype_s_name*' change mode restriction_sc *loop n:* insert '*facet_t_n*' at '*facet_t_n_locator*' with base '*restriction_sc_s_base*';
			5.1.6 id_s != id_t				update complextype name '*complextype_s_name*' change id '*id_t*';
			5.1.7 defaultAttributesApply_s != defaultAttributesApply_t				update complextype name '*complextype_s_name*' change defaultattributesapply '*defaultAttributesApply_t*';
			5.1.8 assert_s != assert_t				update complextype name '*complextype_s_name*' change assert '*assert_t*' at locator '*assert_s_locator*';
		n:m	5.1.9 M[n]={complextype_s_1,..., complextype_s_n}, N[m]={complextype_t_1,..., complextype_t_m}	5.1.9 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]		5.1.9.1 n = m	loop n: complextype_s_n ↔ complextype_t_n (analog zu 5.1.1-5.1.6)
						5.1.9.2 n > m	analog zu 5.1.9.1 (P[m]!=∅, loop m, O[n]\{complextype_s_m})
							loop n: delete complextype name '*complextype_s_n_name*';
						5.1.9.3 n < m	analog zu 5.1.9.1 (O[n]!=∅, P[m]\{complextype_t_n})
							loop n: add complextype name '*complextype_t_n_name*' mixed = '*mixed_t_n*' final = *final_t_n* mode '*complextype_t_n_mode*' id = '*id_t_n*' defaultattributesapply = '*defaultAttributesApply_t_n*' assert='*assert_t_n*';

Abbildung 4.12: ELaX-Übersetzungsregeln für komplexe Typdefinitionen abgeleitet aus C

Dabei zeigen die folgenden Ausdrücke 4.10 und 4.11 die Erzeugung einer solchen Update-Opration.

$$name = nameType\ simpleContent = extension\ extension = xs : string \tag{4.10}$$

$$update\ complextype\ name\ 'nameType'\ change\ mode\ extension\_cc\ with\ base\ 'beschreibungType'; \tag{4.11}$$

Im Quellschema ist die komplexe Typdefinition „nameType“ eine Erweiterung des built-in Datentyps xs:string (Content) um ein Attribut (Range). Die Informationen, die für die nötige Update-Operation gespeichert werden, sind in 4.10 dargestellt. Hier wird der Typ über „simpleContent = extension“ und das benötigte base-Attribut über „extension = xs:string“ gespeichert. Für die Transformation in 4.11 wird die Anforderung des Contents aufgrund der Tatsache erfüllt, dass die Basis „beschreibungType“ ebenfalls den built-in Datentyp xs:string unterstützt. Außerdem bleibt der Wertebereich erhalten, da das Attribut im transformierten Zielschema nicht aus dem komplexen Typ entfernt wird.
Werden keine weiteren Update-Operationen erzeugt, so werden die übriggebliebenen komplexen Typdefinitionen im Quellschema entfernt, sofern diese nicht im Zielschema mehr auftauchen *5.1.9.2*. Außerdem werden die im Zielschema, aber nicht im Quellschema vorhandenen, komplexen Typdefinitionen letzterem hinzugefügt (siehe Operation *5.1.9.3*).

Modellgruppen

Die Inhaltsmodelle (Modellgruppen) Sequenz, Alternative und Menge treten in den jeweiligen komplexen Typdefinitionen auf und besitzen die in Abbildung 4.13 dargestellten Aktualisierungsoptionen.
Hierbei können bei allen Modellgruppen das minimale und maximale Vorkommen, sowie die id angepasst werden (siehe Operationen *6.1.1*, *6.1.2* und *6.1.4*). Zusätzlich bietet ELaX die Option, dass ein groupDefault-Wert für die Alternative angegeben werden kann (siehe Operation *6.1.3*), dessen Definition in Theorem 4.12 gezeigt wird.

$$groupdefault ::= "first"|"last"|INT; \tag{4.12}$$

Schema-komponente	Mapping-Beziehung	Match-kardinalität	Mapping-Regel			ELaX-Operation
6. group: xs:sequence, xs:choice und xs:all	6.1 V_s ↔ V_t	1:1	6.1.1 minOccurs_s != minOccurs_t			update group at '*group_s_locator*' change minoccurs '*minOccurs_t*';
			6.1.2 maxOccurs_s != maxOccurs_t			update group at '*group_s_locator*' change maxoccurs '*maxOccurs_t*';
			6.1.3 choice_s_groupDefault != choice_t_groupDefault			update group at '*group_s_locator*' change mode choice with '*choice_t_groupDefault*';
			6.1.4 id_s != id_t			update group at '*group_s_locator*' change id '*id_t*';
		n:m	6.1.5 M[n] = {group_s_1,..., group_s_n}, N[m] = {group_t_1,..., group_t_m}	6.1.5 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	6.1.5.1 n = m	loop n: group_s_n ↔ group_t_n (analog zu 6.1.1-6.1.4)
					6.1.5.2 n > m	analog zu 6.1.5.1 (P[m]!=∅, loop m, O[n]\{group_s_m})
						loop n: delete group at '*group_s_n_locator*';
					6.1.5.3 n < m	analog zu 6.1.5.1 (O[n]!=∅, P[m]\{group_t_n})
						loop n: add group mode (sequence|choice|all) minoccurs '*minoccurs_t_n*' maxoccurs '*maxoccurs_t_n*' id '*id_t_n*' in '*group_t_n_locator*';

Abbildung 4.13: ELaX-Übersetzungsregeln für Modellgruppen abgeleitet aus C

Dabei wird in *CodeXMapper* standarmäßig angenommen, dass der groupDefault-Wert einer Alternative immer die erste (first) Elementdeklaration bzw. Modellgruppe ist. Möchte der Benutzer diesen Defaultwert ändern, fügt er zunächst eine Mappingbeziehung zwischen der Quell- und Ziel-Choice-Modellgruppe ein, sofern diese nicht schon existiert. Anschließend wird auf eines der beiden choice-Elemente per Rechtsklick ein Popup-Menü geöffnet, welches das Auswählen eines anderen groupDefault-Wertes ermöglicht. Danach wird der entsprechende ELaX-Ausdruck erzeugt (siehe Operation *6.1.3*).
Weiterhin können die Modellgruppen ineinander überführt werden, indem darauf geachtet werden muss, ob der Inhalt (Content) bzw. der Wertebereich (Range) übereinstimmt. Der Content lässt sich analog zu einfachen und komplexen Typdefinitionen aus [144] ableiten. Der Wertebereich ist darauf fokussiert, dass beispielsweise nach einer Transformation von einer Sequenz zu einer Alternative, die Alternativauswahl sich auf ein Element bzw. Modellgruppe beschränkt, die mit den Elementen der vorherigen Sequenz übereinstimmt.
Weiterhin werden nicht mehr im Zielschema benötigte Inhaltsmodelle aus dem Quellschema entfernt bzw. nur im Zielschema vorkommende Modellgruppen hinzugefügt (siehe Operationen *6.1.5.2* und *6.1.5.3*).

Elementdeklarationen

Die Elementdeklarationen können nach dem *Garden of Eden Stil* global definiert und lokal referenziert werden. Die Element-Wildcards sollen ebenfalls zu Elementdeklarationen gezählt werden, da sie beliebige Elemente aus dem Schema selbst, oder aus einem externen Schema integrieren. Außerdem werden Elementgruppen als mehrere einzelne Elementdeklarationen in CodeX aufgelöst, weshalb diese nicht weiter betrachtet werden. Die möglichen Update-Operationen für die einzelnen Elementarten sind in Abbildung 4.14 dargestellt.
Dabei lassen sich für alle Elementdeklarationen das id-Attribut anpassen (siehe Operation *7.1.14*). Außerdem können bei den globalen Definitionen das name-, type-, default-, fixed-, final-, und nillable-Attribut geändert werden (siehe Operationen *7.1.1-7.1.6*). Äuqivalent zu Namensänderung von globalen Elementdeklarationen verhält sich die Anpassung des ref-Attributs einer Elementreferenz (siehe Operation *7.1.7*). Wenn sich die Elementreferenz im Zielschema an einer anderen Position befindet kann ein entsprechender „move to"-Ausdruck erzeugt werden, analog zu 4.7 (siehe Operation *7.1.15*). Die Elementreferenz und die Element-Wildcard haben gemein, dass sie jeweils das minimale und/oder maximale Vorkommen anpassen können (siehe Operationen *7.1.8* und *7.1.9*). Weiterhin können Wildcard-spezifische Attribute in den Operationen *7.1.10-7.1.13* angepasst werden.
Werden Elementdeklarationen im Zielschema nicht mehr benötigt, so werden sie aus dem Quellschema entfernt (siehe Operation *7.1.16.2*). Im Gegensatz dazu werden Elementdefinitionen eingefügt, sofern diese im Zielschema vorkommen, aber nicht im Quellschema vorhanden sind (siehe Operation *7.1.16.3*). Als Beispiel wird in 4.13 das Einfügen einer Element-Wildcard gezeigt.

$$\begin{aligned} &add\ any\ namespace\ 'http://www.ausstattung.org'\ maxoccurs\ 'unbounded' \\ &\quad in\ '/node()/node()[@name = "ausstattungType"]/node()/.'; \end{aligned} \tag{4.13}$$

Schema-komponente	Mapping-Beziehung	Match-kardinalität	Mapping-Regel			ELaX-Operation
7. xs:element: global, reference und any	7.1 V_s ↔ V_t	1:1	7.1.1 global_s_name != global_t_name			update element name '*global_s_name*' change name '*global_t_name*';
			7.1.2 global_s_type != global_t_type			update element name '*global_s_name*' change type '*global_t_type*';
			7.1.3 global_s_default != global_t_default			update element name '*global_s_name*' change default '*global_t_default*';
			7.1.4 global_s_fixed != global_t_fixed			update element name '*global_s_name*' change fixed '*global_t_fixed*';
			7.1.5 global_s_final != global_t_final			update element name '*global_s_name*' change final '*global_t_final*';
			7.1.6 global_s_nillable != global_t_nillable			update element name '*global_s_name*' change nillable '*global_t_nillable*';
			7.1.7 reference_s_ref != reference_t_ref			update elementref '*reference_s_ref*' (at '*reference_s_locator*' | '*reference_s_refposition*') change ref '*reference_t_ref*';
			7.1.8 (reference_s|any_s)_minOccurs != (reference_t|any_t)_minOccurs			update elementref '*reference_s_ref*' (at '*reference_s_locator*' | '*reference_s_refposition*') change minoccurs '*reference_t_minOccurs*';
						update any at '*any_s_locator*' change minoccurs '*any_t_minOccurs*';
			7.1.9 (reference_s|any_s)_maxOccurs_s != (reference_t|any_t)_maxOccurs			update elementref '*reference_s_ref*' (at '*reference_s_locator*' | '*reference_s_refposition*') change maxoccurs '*reference_t_maxOccurs*';
						update any at '*any_s_locator*' change maxoccurs '*any_t_maxOccurs*';
			7.1.10 any_s_notQName != any_t_notQName			update any at '*any_s_locator*' change not '*any_t_notQName*';
			7.1.11 any_s_namespace != any_t_namespace			update any at '*any_s_locator*' change namespace '*any_t_namespace*';
			7.1.12 any_s_notNamespace != any_t_notNamespace			update any at '*any_s_locator*' change namespace not '*any_t_notNamespace*';
			7.1.13 any_s_processContents != any_t_processContents			update any at '*any_s_locator*' change processcontent '*any_t_processContents*';
			7.1.14 (global_s|reference_s|any_s)_id != (global_t|reference_t|any_t)_id			update element name '*global_s_name*' change id '*global_t_id*';
						update elementref '*reference_s_ref*' (at '*reference_s_locator*' | '*reference_s_refposition*') change id '*reference_t_id*';
						update any at '*any_s_locator*' change id '*any_t_id*';
			7.1.15 reference_s_position != reference_t_position			update elementref '*reference_s_ref*' (at '*reference_s_locator*' | '*reference_s_refposition*') move to '*reference_t_position*';
		n:m	7.1.16 M[n] = {element_s_1,..., element_s_n}, N[m] = {element_t_1,..., element_t_m}	7.1.16 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	7.1.16.1 n = m	loop n: element_s_n ↔ element_t_n (analog zu 7.1.1-7.1.15)
					7.1.16.2 n > m	analog zu 7.1.16.1 (P[m]!=∅, loop m, O[n]\{element_s_m})
						loop n: delete element name '*global_s_n_name*';
						loop n: delete elementref '*reference_s_n_ref*' (at '*reference_s_n_locator*'|'*reference_s_n_refposition*');
						loop n: delete any at '*any_s_n_locator*';
					7.1.16.3 n < m	analog zu 7.1.16.1 (O[n]!=∅, P[m]\{element_t_n})
						loop n: add element name '*global_t_n_name*' type '*global_t_n_type*' (default '*global_t_n_default*' | fixed '*global_t_n_fixed*') final '*global_t_n_final*' nillable '*global_t_n_nillable*' id '*global_t_n_id*';
						loop n: add elementref '*reference_t_n_ref*' minoccurs '*reference_t_n_minOccurs*' maxoccurs '*reference_t_n_maxOccurs*' id '*reference_t_n_id*' '*reference_t_n_position;*
						loop n: add any not '*any_t_n_notQName*' namespace '*any_t_n_namespace*' not '*any_t_n_notNamespace*' processcontent '*any_t_n_processContents*' minoccurs '*any_t_n_minOccurs*' maxoccurs '*any_t_n_maxOccurs*' id '*any_t_n_id*' in '*any_t_n_locator*';

Abbildung 4.14: ELaX-Übersetzungsregeln für Elementdeklarationen abgeleitet aus C

Attributdeklarationen

Die Attributdeklarationen umfassen die globalen Attributgruppen und Attribute, sowie die lokal definierbaren Attribut(gruppen)referenzen und Attribut-Wildcards. Die möglichen Update-Operationen sind in der Abbildung 4.15 zusammengefasst.
Dabei lassen sich analog zu den Elementdeklarationen die id-Attribute bei allen Attributdeklarationen anpassen (siehe Operation *8.1.13*). Außerdem lässt sich bei einer globalen Attribut(gruppen)definition das name-Attribut ändern (siehe Operation *8.1.1*). Innerhalb der Attributgruppe können verschiedene Attributreferenzen und/oder eine Attribut-Wildcard vorkommen. Diese Menge kann sich im Zielschema ändern, sodass Anpassungen notwendig werden (siehe Operation *8.1.2*). Dabei lassen sich die unterschiedlichen Attributreferenzen oder Attribut-Wildcards nur darüber anpassen, wenn zunächst die neue Referenz oder Wildcard eingefügt und die alte gelöscht wird (siehe Theorem 4.14).

$$\begin{aligned} &update\ attributegroup\ name\ 'tour'\ change \\ &add\ attributeref\ 'start'\ use\ 'required'\ in\ '/node()/node()[@name = "tour"]/.' \\ &delete\ attributeref\ 'anfang'\ at\ '/node()/node()[@name = "tour"]/.'; \end{aligned} \tag{4.14}$$

Weiterhin können bei globalen Attributen das type- und das inheritable-Attribut angepasst werden (siehe Operationen *8.1.3* und *8.1.6*). Außerdem lassen sich bei Attributen und Attributreferenzen die default- und fixed-Werte ändern (siehe Operationen *8.1.4* und *8.1.5*). Daneben lassen sich bei Attribut(gruppen)referenzen der ref-Name anpassen bzw. die Position (vgl. 4.7), wobei nur bei den Attributrefernzen das use-Attribut geändert werden kann (siehe Operationen *8.1.7*, *8.1.14* und *8.1.8*). Die für Attribut-Wildcard-Deklarationen spezifischen Anpassungen sind in den Operationen *8.1.9-8.1.12* einzusehen. Sind keine Update-Operationen möglich, werden die fehlenden Attributdeklarationen vom Ziel- ins Quellschema eingefügt bzw. nicht mehr im Zielschema vorkommende Definitionen aus dem Quellschema entfernt (siehe Operationen *8.1.15.3* und *8.1.15.2*).

Schema-komponente	Mapping-Beziehung	Match-kardinalität	Mapping-Regel			ELaX-Operation
8. xs:attribute Group und xs:attribute: globalAG, referenceAG, globalA, referenceA und any	8.1 V_s ↔ V_t		8.1.1 (globalAG_s|globalA_s)_name != (globalAG_t|globalA_t)_name			update (attributegroup|attribute) name '*(globalAG_s|globalA_s)_name*' change name '*(globalAG_t|globalA_t)_name*';
			8.1.2 M[n] = {globalAG_Content_s_1,..., globalAG_Content_s_n}, N[m] = {globalAG_Content_t_1,..., globalAG_Content_t_n}	8.1.2 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]		O[n]!=∅: update attributegroup name '*globalAG_s_name*' change (*loop n:* delete <delattributeref>_s_n) (delete <delattributewildcard>_s);
						P[m]!=∅: update attributegroup name '*globalAG_s_name*' change (*loop n:* add <addattributeref>_t_n) (add <addattributewildcard>_t);
			8.1.3 globalA_s_type != globalA_t_type			update attribute name '*globalA_s_name*' change type '*globalA_t_type*';
			8.1.4 (globalA_s|referenceA_s)_default != (globalA_t|referenceA_t)_default			update attribute name '*globalA_s_name*' change default '*globalA_t_default*';
						update attributeref '*referenceA_s_ref*' at '*referenceA_s_locator*' change default '*referenceA_t_default*';
			8.1.5 (globalA_s|referenceA_s)_fixed != (globalA_t|referenceA_t)_fixed			update attribute name '*globalA_s_name*' change fixed '*globalA_t_fixed*';
						update attributeref '*referenceA_s_ref*' at '*referenceA_s_locator*' change fixed '*referenceA_t_fixed*';
			8.1.6 globalA_s_inheritable != globalA_t_inheritable			update attribute name '*globalA_s_name*' change inheritable '*globalA_t_inheritable*';
		1:1	8.1.7 (referenceAG_s|referenceA_s)_ref != (referenceAG_t|referenceA_t)_ref			update (attributegroupref|attributeref) '*(referenceAG_s| referenceA_s)_ref*' at '*(referenceAG_s|referenceA_s)_locator*' change ref '*(referenceAG_t|referenceA_t)_ref*';
			8.1.8 referenceA_s_use != referenceA_t_use			update attributeref '*referenceA_s_ref*' at '*referenceA_s_locator*' change use '*referenceA_t_use*';
			8.1.9 any_s_notQName != any_t_notQName			update anyattribute at '*any_s_locator*' change not '*any_t_notQName*';
			8.1.10 any_s_namespace != any_t_namespace			update anyattribute at '*any_s_locator*' change namespace '*any_t_namespace*';
			8.1.11 any_s_notNamespace != any_t_notNamespace			update anyattribute at '*any_s_locator*' change namespace not '*any_t_notNamespace*';
			8.1.12 any_s_processContents != any_t_processContents			update anyattribute at '*any_s_locator*' change processcontent '*any_t_processContents*';
			8.1.13 (globalAG_s|referenceAG_s|globalA_s|referenceA_s|any_s)_id != (globalAG_t|referenceAG_t|globalA_t|referenceA_t|any_t)_id			update (attributegroup|attribute) name '*(globalAG_s| globalA_s)_name*' change id '*(globalAG_t|globalA_t)_id*;
						update (attributegroupref|attributeref) '*(referenceAG_s| referenceA_s)_ref*' at '*(referenceAG_s|referenceA_s)_locator*' change id '*(referenceAG_t|referenceA_t)_id*';
						update anyattribute at '*any_s_locator*' change id '*any_t_id*';
			8.1.14 (referenceAG_s|referenceA_s)_locator != (referenceAG_t|referenceA_t)_locator			update (attributegroupref|attributeref) '*(referenceAG_s| referenceA_s)_ref*' at '*(referenceAG_s|referenceA_s)_locator*' move into '*(referenceAG_t|referenceA_t)_locator*';
		n:m	8.1.15 M[n]={(attributeGroup_s|attribute_s)_1,..., (attributeGroup_s|attribute_s)_n}, N[m]={(attributeGroup_t|attribute_t)_1,..., (attributeGroup_t|attribute_t)_m}	8.1.15 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	8.1.15.1 n = m	loop n: (attributeGroup_s|attribute_s)_n ↔ (attributeGroup_t|attribute_t)_n (analog zu 8.1.1-8.1.14)
					8.1.15.2 n > m	analog zu 8.1.15.1 (P[m]!=∅, loop m, O[n]\{(attributeGroup_s|attribute_s)_m})
						loop n: delete (attributegroup|attribute) name '*(globalAG_s|globalA_s)_n_name*';
						loop n: delete (attributegroupref|attribute) '*(referenceAG_s| referenceA_s)_ref*' at '*(referenceAG_s|referenceA_s)_locator*';
						loop n: delete anyattribute at '*any_s_n_locator*';
					8.1.15.3 n < m	analog zu 8.1.15.1 (O[n]!=∅, P[m]\{(attributegroupref_t|attribute_t)_n})
						loop n: add attributegroup name '*globalAG_t_n_name*' id '*globalAG_t_n_id*' with *loop x* <addattributeref>_x <addattributewildcard>; x>0
						loop n: add attributegroupref '*referenceAG_t_n_name*' id '*referenceAG_t_n_id*' in '*referenceAG_t_n_locator*';
						loop n: add attribute name '*globalA_t_n_name*' type '*globalA_t_n_type*' (default '*globalA_t_n-_default*' | fixed '*globalA_t_n_fixed*') inheritable '*globalA_t_n_inheritable*' id '*globalA_t_n_id*';
						loop n: add attributeref name '*referenceA_t_n_name*' (default '*referenceA_t_n_default*' | fixed '*ref erenceA_t_n_fixed*') use '*referenceA_t_n_use*' id '*referenceA_t_n_id*' in '*referenceA_t_n_locator*';
						loop n: add anyattribute not '*any_t_n_notQName*' namespace '*any_t_n_namespace*' not '*any_t_n _notNamespace*' processcontent '*any_t_n_processContents*' id '*any_t_n_id*' in '*any_t_n_locator*';

Abbildung 4.15: ELaX-Übersetzungsregeln für Attributdeklarationen abgeleitet aus C

Constraintdeklarationen

Unter den Constraintdeklarationen werden die Schemakomponenten key, unique und keyref verstanden. Darauf sind die in Abbildung 4.16 dargestellten Update-Operationen anwendbar.
Bei allen Contraintdeklarationen lassen zunächst die name- und id-Attribute anpassen, sowie deren Position ähnlich dem Theorem 4.7 (siehe Operationen *9.1.1*, *9.1.4* und *9.1.5*). Außerdem lässt sich nur beim Keyref-Constraint das refer-Attribut ändern, da dies zur Referenz auf ein bestehenden (optionalen) Primärschlüssel benötigt wird. Weiterhin lassen sich bei allen Contraintdeklarationen die Selector- bzw. Field-Pfade ändern, indem diese ähnlich zur Attributgruppe (siehe Theorem 4.14) angepasst werden (siehe Operation *9.1.3*). Diese Analogie wird im folgenden Ausdruck 4.15 deutlich.

$$update\ constraint\ name\ 'routeKey'\ at\ '/node()/node()[@name = "fernbusverkehr"]/.'$$
$$change\ type\ 'unique'\ remove\ selector\ './fernbuslinie/fahrplan/route'\ insert\ selector\ './routen/route';\tag{4.15}$$

Der dargestellte Ausdruck 4.15 beinhaltet auch gleich eine Typänderung von einem zuvor deklarierten Key-Contraint zu einem Unique-Constraint. Dabei lassen sich die Constraint-Deklarationen ineinander überführen, sofern sie sich strukturell (Content nach [144]) übereinstimmen, sowie den gleichen Wertebereich (Range) abdecken. In diesem Fall bedeutet der Wertebereich, dass dieselben Schemakomponenten im Quell- und Zielschema derselben Constraintbedingung unterliegen. Dabei muss häufig der Selector- bzw. Field-Pfad angepasst werden, abhängig von der neuen XML-Struktur.
Werden die Contraintdeklarationen im Quellschema im Zielschema nicht mehr benötigt werden sie im Ausgangsschema entfernt (siehe Operation *9.1.6.2*). Analog dazu werden die Constraints ins Quellschema eingefügt, die im Ziel- aber nicht im Ausgangsschema vorkommen (siehe Operation *9.1.6.3*).

Schema-komponente	Mapping-Beziehung	Match-kardinalität	Mapping-Regel			ELaX-Operation
9. xs:key, xs:unique und xs:keyref	9.1 V_s ↔ V_t	1:1	9.1.1 (key_s|unique_s|keyref_s)_name != (key_t|unique_t|keyref_t)_name			update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator*' change name '*(key_t|unique_t|keyref_t)_name*';
			9.1.2 keyref_s_ref != keyref_t_refer			update constraint name '*keyref_s_name*' at '*keyref_s_locator*' change type keyref refer '*keyref_t_refer*';
			9.1.3 M[n]={(selector_s|field_s)_1,..., (selector_s|field_s)_n }, N[m]={(selector_t|field_t)_1,..., (selector_t|field_t)_n }	9.1.3 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]		O[n]!=∅: update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator*' change *loop n:* remove <updconstraintpath>_s_n;
						P[m]!=∅: update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator*' change *loop n:* insert <updconstraintpath>_t_n;
			9.1.4 (key_s|unique_s|keyref_s)_id != (key_t|unique_t|keyref_t)_id			update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator*' change id '*(key_t|unique_t|keyref_t)_id*';
			9.1.5 (key_s|unique_s|keyref_s)_locator != (key_t|unique_t|keyref_t)_locator			update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator*' move into '*(key_t|unique_t|keyref_t)_locator*';
		n:m	9.1.6 M[n]={(key_s|unique_s|keyref_s)_1, ..., (key_s|unique_s|keyref_s)_n}, N[m]={(key_t|unique_t|keyref_t)_1, ..., (key_t|unique_t|keyref_t)_m}	9.1.6 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	9.1.6.1 n = m	loop n: (key_s|unique_s|keyref_s)_n ↔ (key_t|unique_t|keyref_t)_n (analog zu 9.1.1-9.1.5)
					9.1.6.2 n > m	analog zu 9.1.6.1 (P[m]!=∅, loop m, O[n]\{(key_s|unique_s|keyref_s)_m})
						loop n: delete constraint name '*(key_s|unique_s|keyref_s)_n_name*';
					9.1.6.3 n < m	analog zu 9.1.6.1 (O[n]!=∅, P[m]\{(key_t|unique_t|keyref_t)_n})
						loop n: add constraint name '*(key_t|unique_t|keyref_t)_n_name*' type (key|unique|(keyref refer '*keyref_t_n_refer*')) with <addconstraintpath> id '*(key_t|unique_t|keyref_t)_n_id*' in '*(key_t|unique_t|keyref_t)_n_locator*';

Abbildung 4.16: ELaX-Übersetzungsregeln für Constraintdeklarationen abgeleitet aus C

4.3.3 Integration CodeXMapper in CodeX

Nachdem die Funktionalitäten von COMA in den *CodeXMapper* überführt und mit dem ELaX-Übersetzungsprozess erweitert wurde, wird der *CodeXMapper* in CodeX integriert, dargestellt in Abbildung 4.17.
Aufgrund der Tatsache, dass CodeX das Google Web Toolkit als Entwicklungsoberfläche benutzt, wird eine Client-seitige Integration angestrebt. Dabei sind Probleme hauptsächlich bei der Integration von zusätzlichen Bibliotheken, sowie der Benutzeroberfläche vom *CodeXMapper* aufgetreten. Das Bibliotheksproblem hat sich dahingehend gelöst, dass ein entsprechender Einstiegspunkt in den eingebundenen Bibliotheken über eine XML-Datei der Form „*.gwt.xml“ erfolgen musste. Das zweite Problem bezieht sich darauf, dass im Client-Modus keine Bibliotheken unterstützt werden, die „javax.swing“ emulieren könnten. Aufgrund des zu hohen Aufwandes die komplette Benutzeroberfläche von *CodeXMapper* neu aufzusetzen, wurde sich für den asynchronen Serveraufruf entschieden, der die entsprechende Anwendung auf den Server startet und somit die Funktionalitäten dem CodeX-Tool zur Verfügung stellt.
Dazu wird in der Benutzeroberfläche von CodeX zunächst ein neuer Menüpunkt „Matcher“ erzeugt, welcher bei Aktivierung ein Objekt der MatcherCommand-Klasse erzeugt. Danach wird ein Dialog über die MatcherDialog-Klasse erzeugt, welche den asynchronen RPC Aufruf zum Server tätigt. Dafür wird über das MatcherServiceAsync-Interface, welche über die entsprechende MatcherService-Interface die MatcherServiceImpl-Klasse auf dem Server aufruft. Letzere führt dann den Befehl zum Starten der Controller-Klasse aus, wodurch die in den Packages zusammengefassten Funktionalitäten zur Verfügung gestellt werden.

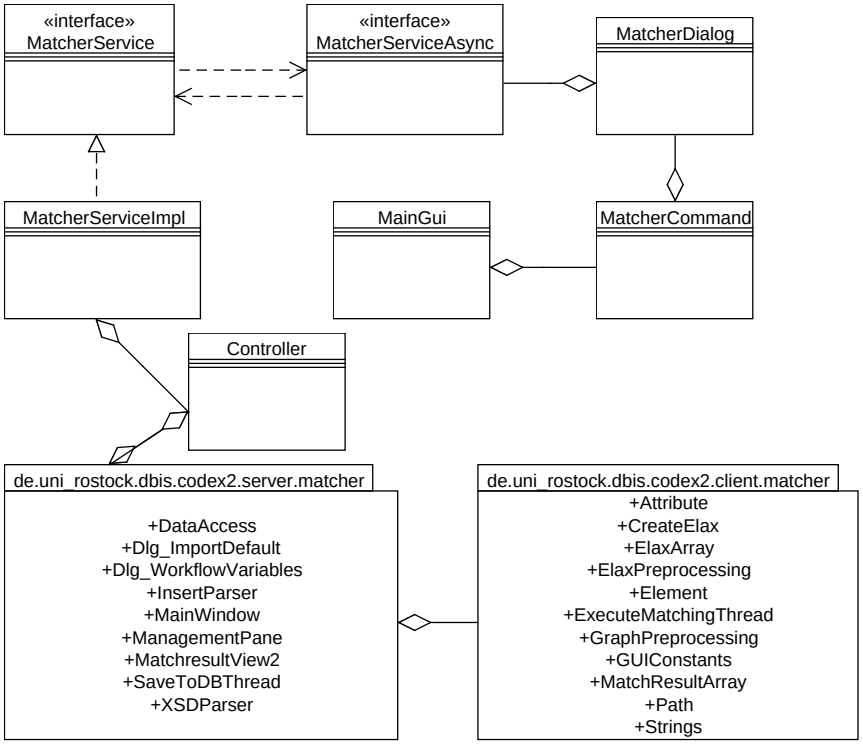

Abbildung 4.17: Klassendiagramm von CodeX nach der CodeXMapper-Integration

Kapitel 5

Selbstgewähltes Beispiel für die XML-Schema Evolution

Aufgrund der Zulassung von Fernbuslinien seit dem 1. Januar 2013 wächst das Liniennetz und somit das Angebot von Fernverbindungen. Die Verantwortlichen einer Suchmaschine für Fernbuslinien möchten daher einen besonderen Service anbieten. Dieser besteht darin, dass potentielle Kunden jeder Zeit Routeninformationen abfragen können. Für das Vorhaben werden ein vorhandenes XML-Dokument mit zugehörigem XML-Schema im *Garden of Eden* Stil (vgl. [91] und [71]) verwendet, die im Abschnitt 5.1 vorgestellt werden. Der detaillierte Code der beiden Dokumente befindet sich im Anhang E. Das veraltete XML-Dokument ist allerdings nur umständlich auswertbar, weswegen Anpassungen am XML-Schema und XML-Dokument nötig werden. Dazu werden relevante Anpassungsschritte im Abschnitt 5.2 entwickelt, die zusätzlich gegen eine Kategorisierung von Schemaevolutionsschritten abgeglichen werden. Diese befindet sich wegen des großen Umfangs im Anhang B. Abschließend werden im Abschnitt 5.3 die erwarteten Auswirkungen durch die Schemaevolutionsschritte auf das XML-Schema und/oder XML-Dokument zusammengefasst.

5.1 Aufbau XML-Schema und XML-Dokument

Der grundlegende Aufbau des XML-Schemas ist in Abbildung 5.1 dargestellt. Das abgebildete XML-Schema setzt sich aus den XML-Schemata Fernbusverkehr (E.1) und Zeit (E.5) zusammen. Hierbei wird das XML-Schema Zeit über einen include-Befehl eingebettet. Das definierte Wurzelelement (Root-Element) ist Fernbusverkehr, welches maximal zehn Fernbuslinie-Elemente enthalten kann. Die geringe Anzahl spiegelt die wenigen Fernbuslinien vor 2013 wieder. Zusätzlich enthält das Root-Element ein Attribut „last_schema_change“, das den Zeitpunkt der letzten Schemaänderung angibt.
Die Fernbuslinie-Elemente wiederrum enthält die Elemente Anbieter, Fahrplan, Ausstattung und Beschreibung in der angegebenen Reihenfolge. Zum Anbieter werden der Name, das Gründungjahr, die Unternehmungsform und die Homepage angegeben. Die Angaben gelten als abgeschlossen, weshalb das Attribut „final“ auf „#all“ gesetzt wird. Der Name erweitert den vordefinierten (built-in) Datentyp xs:string durch das Attribut Abkürzung, um zusätzlich zum Anbieternamen dessen Akronym zu speichern. Das Gründungsjahr ist vom built-in Datentyp xs:gYear und definiert das Jahr der Unternehmensgründung. Die dabei möglichen Organisationsformen werden in den Unternehmensformen als Einschränkung definiert, indem nur die gängigen Rechtsformen GmbH, AG, KG und GmbH & Co. KG gültig sind. Die Angabe der Homepage dient als weitere Informationsquelle zum Anbieter, deren Webadresse durch den voreingebauten Datentyp xs:anyURI dargestellt wird. Damit kein Anbieter doppelt in der Instanz vorkommen kann, wird das key-Element anbieterKey unter dem Root-Element eingefügt, welches die Eindeutigkeit des Anbieternamens im gesamten Dokument gewährleistet.

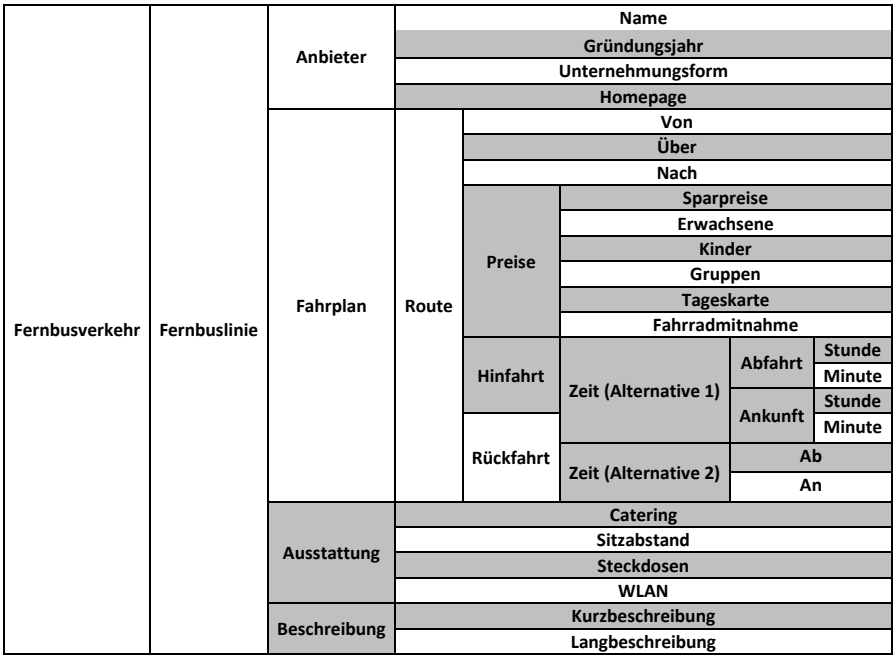

Abbildung 5.1: Aufbau des XML-Schemas abgeleitet aus E.1 und E.5

Der Fahrplan kann einen oder mehrere Route-Elemente enthalten, d.h. je nachdem wie viele Routen von einem Anbieter angeboten werden. Sind beispielsweise mehr als eine Route vorhanden, kann mit dem optionalen Attribut „routeID“ vom Typ string eine Unterscheidung durchgeführt werden. Außerdem werden meist die Transportdienstleistung von Dritten durchgeführt, sodass diese mit einem benötigten Attribut „buslinien“ dargestellt werden. Dabei stehen vordefinierte Busunternehmen, z.B. Joost's Ostsee-Express, zur Verfügung, die über eine Vereinigung mit dem Datentyp string verknüpft werden, falls mehrere Busunternehmen auf der Route eingesetzt werden. Zu jeder Route wird zunächst der Start (von), gegebenenfalls eine Zwischenstation (über) und das Ziel (nach) mit dem built-in Datentyp xs:string angegeben. Anschließend werden im Preise-Element die möglichen Kosten aufgelistet, die beispielsweise bei der Beförderung von einem Erwachsenen oder bei Fahrradmitnahme entstehen. Hierbei werden die Preise als Dezimalzahlen angegeben, beschränkt auf zwei Stellen nach dem Komma. Außerdem wird im Preise-Element selbst zum Einen das Währungsattribut benötigt, welches zwischen Euro (EUR) und Schweizer Franken (CHF) unterscheidet. Zum Anderen muss die Angabe der Mehrwertsteuer über das Attribut „MwSt.“ erfolgen, d.h. ob diese bereits in die Preisberechnung eingegangen ist (inklusive) oder nicht (nicht inklusive). Nach den Preis-Elementen können die Elemente Hinfahrt und Rückfahrt definiert werden, welche im XML-Schema Zeit definiert sind. Hierbei enthalten sowohl das Hinfahrt- als auch das Rückfahrt-Element das Element Zeit, wobei eine Auswahl über die Zeitangabe getroffen werden kann. Sollen Stunde und Minute in Extra-Elementen erscheinen, werden das Abfahrt- und Ankunft-Element gewählt. Andererseits werden über die Elemente ab und an die Uhrzeit als Datentyp time dargestellt. Außerdem können die einzelnen Zeit-Elemente durch das Attribut „rythmus“ die Anzahl der Verbindungen enthalten, d.h. ob die Busse an bestimmten Tagen oder täglich (Default) auf der Route verkehren.
Die Ausstattung-Elemente beinhalten die Kindelemente Catering, Sitzabstand, Steckdosen und WLAN. Beim Catering wird angegeben ob und wenn ja explizit welcher Service angeboten wird, z.B. Snacks oder Kaffee. Beim Sitzabstand werden die drei Kategorien groß, klein und nein (keine Angabe) unterschieden. Bei Steckdosen und WLAN wird deren Verfügbarkeit angegeben, d.h. wenn sie verfügbar sind mit ja sonst nein.

Das letzte Element Beschreibung enthält allgemeine Informationen zum Anbieter, d.h. komprimiert in einem Kurzbeschreibung-Element und ausführlich in einem Langbeschreibung-Element. Zum besseren Verständnis dient ein Auszug eines XML-Dokumentes in Abbildung 5.2, welches valide zum oben beschriebenen XML-Schema ist.

```
<?xml version="1.0" encoding="UTF-8"?>
<fernbusverkehr [...] last_schema_change="01.01.2012">
<fernbuslinie>
 <anbieter>
  <name abkürzung="MFB">MeinFernbus</name>
  <gründungsjahr>2011</gründungsjahr>
  <unternehmungsform>GmbH</unternehmungsform>
  <homepage>http://meinfernbus.de/</homepage>
 </anbieter>
 <fahrplan>
  <route routeID="rou1" buslinien="Joost's Ostsee-Express Wörlitz Tourist">
   <von>Warnemünde</von>
   <über>Rostock</über>
   <nach>Berlin</nach>
   <preise währung="EUR" MwSt.="inklusive">
    <sparpreise>11.00</sparpreise>
    <erwachsene>21.00</erwachsene>[...]
   </preise>
   <hinfahrt>
    <zeit rythmus="täglich">
     <abfahrt>
      <stunde>06</stunde>
      <minute>40</minute>
     </abfahrt>
     <ankunft>
      <stunde>09</stunde>
      <minute>45</minute>
     </ankunft>
    </zeit>[...]
   </hinfahrt>
   <rückfahrt>
    <zeit rythmus="täglich">
     <ab>07:00:00+01:00</ab>
     <an>10:00:00+01:00</an>
    </zeit>[...]
   </rückfahrt>
  </route>
 </fahrplan>
 <ausstattung>
  <catering>Snacks Kaffee </catering>
  <sitzabstand>groß</sitzabstand>
  <steckdosen>ja</steckdosen>
  <wlan>ja</wlan>
 </ausstattung>
 <beschreibung>
  <kurzbeschreibung>Der 2011 in Berlin gegründete Verbund MeinFernbus will nach eigenen Angaben beliebtester und
       bekanntester Anbieter von Fernbuslinien werden.</kurzbeschreibung>[...]
</fernbuslinie>[...]
</fernbusverkehr>
```

Abbildung 5.2: Auszug aus Fernbuslinie.xml abgleitet aus E.2

5.2 Beschreibung und Abgleich der Schemaevolutionsschritte

Zum besseren Verständnis der Ergebnisse im Abschnitt 5.3 werden in den folgenden Abschnitten Anforderungen entwickelt, aus denen entsprechende Schemaevolutionsschritte resultieren. Diese werden gegen die Kategorisierung von Schemaevolutionsschritten abgeglichen, um einerseits die verschiedenen Schemaevolutionsmöglichkeiten und andererseits die Ausdrucksfähigkeit von ELaX (vgl. [111] und [112]) abzudecken.

5.2.1 Strukturverbesserung für Suchanfragen

Die Verantwortlichen sehen den Bedarf zur Optimierung der XML-Dokumente für Suchanfragen, die durch potentielle Fahrgäste gestellt werden. Hierbei zielen die Anfragen häufig auf die angebotenen Routen, weshalb die gegenwärtige anbieterorientierte Speicherung umstrukturiert wird. Die nachfolgend entwickelten Editieranforderungen sind in den Abbildungen 5.3, 5.4 und 5.5 zusammengefasst.
Als Erstes sollen zunächst die vorhandenen Routen unter dem Root-Element Fernbusverkehr in Routen-Elementen nach Start- und Zielpunkt gebündelt werden. Hierzu wird die im komplexen Typ „fernbusverkehrType“ befindliche Fernbuslinie-Elementreferenz in „routen“ umbenannt (siehe Operation *3.6.2.1*). Die dadurch übernommene maxOccurs-Beschränkung auf zehn Elemente wird auf „unbounded“ erhöht, weil durchaus mehrere Routen-Elemente existieren können (siehe Operation *3.6.2.5*). Da das zuvor referenzierte Fernbuslinie-Element nicht mehr benötigt wird, kann es zusammen mit dem entsprechenden komplexen Typ „fernbuslinieType“ gelöscht werden (siehe Operationen *2.6.1* und *2.3.1.2*). Anschließend wird das Routen-Element mit dem type-Attributwert „routenType“ eingefügt (siehe Operation *1.6.1*). Aufgrund des Vorhandenseins der Route-Elemente im komplexen Typ „fahrplanType“ wird der Typname in „routenType“ umbenannt (siehe Operation *3.5.1*). Die erforderlichen Start- und Zielpunkte werden im „routenType“ über die Attributgruppenreferenz „tour“ eingebunden (siehe Operation *1.2.4*). Da die Referenz keine vorhandene Attributgruppe referenziert, wird die entsprechende Attributgruppendefiniton „tour“ eingefügt (siehe Operation *1.2.1*). In diese Definition werden dann die beiden Attributreferenzen „start“ und „ziel“ hinzugefügt, jeweils mit dem use-Attribut „required“ (siehe Operation *1.2.3.3*). Die damit erforderlichen Attributdefinitionen werden mit dem Attributinhalt vom built-in Datentyp xs:string ergänzt (siehe Operation *1.2.2*). Damit keine Dopplungen in der Attributbelegung von „start“ und „ziel“ auftreten, wird im Fernbusverkehr-Element das Constraint „routenKey“ eingefügt (siehe Operation *1.8.1.1*).
Weiterhin werden zur besseren Übersichtlichkeit die verschiedenen Fernbusanbieter kompakt unter dem Wurzelelement gespeichert. Dafür wird die Fernbusanbieter-Elementreferenz dem komplexen Typ „fernbusverkehrType“ hinzugefügt (siehe Operation *1.6.2.1.2*). Danach wird das benötigte Element „fernbusanbieter“ ergänzt, welches vom komplexen Typ „fernbusanbieterType“ ist (siehe Operation *1.6.1*). Das Fernbusanbieter-Element kann mehrere Anbieter enthalten, weshalb der neu eingefügte komplexe Typ „fernbusanbieterType“ durch die Elementreferenz „anbieter“ mit dem maximalen Vorkommen „unbounded“ vervollständigt wird (siehe Operation *1.6.2.1*).

ELaX-Operation: add

Die zuvor entwickelten Einfügeoperationen sind in Abbildung 5.3 in die Kategorien *addattributegroup (1.2)*, *addgroup (1.3)*, *addelement (1.6)* und *addconstraint (1.8)* eingeteilt worden.
Dabei ist die Operation *1.2.3.1* äquivalent zum Einfügen der zwingend erforderlichen (required) Attributreferenzen „start“ und „ziel“ (siehe Operation und *1.2.3.3*). Durch das Einfügen eines required-Attributs müssen diese im entsprechenden Element auftauchen. Wenn ein verbotenes (prohibited) Attribut eingefügt wird, ist das Hinzufügen des entsprechenden Attributs nicht erlaubt und somit nicht notwendig.
Die Einfügeoperation *1.3.1.2* ergänzt die Gruppendefinition Sequenz in den „fernanbieterType“, wobei standardmäßig mindestens und höchstens ein Element vom selben Typ in der festgelegten Reihenfolge auftreten kann. Wenn das minimale Vorkommen einer Sequenz Null beträgt, ist das Weglassen der gesamten Kindelemente erlaubt. Somit wäre die Operation *1.3.1.1* analog zu *1.3.1.2*, wenn bei den Elementreferenzen in *1.3.1.2* die minOccurs-Werte auf Null gesetzt werden. Die Alternative kann zwischen verschiedenen Elementen bzw. Elementreihenfolgen auswählen. Dabei kann eine Übereinstimmung der Operationen *1.3.2.1* und *1.3.2.2* mit der Operation *1.3.1.2* erreicht werden, wenn die die minOccurs-Werte der Sequenzelemente Null betragen und/oder die ausgewählte Alternative der Sequenz ähnelt. Die Menge unterscheidet sich von der der Sequenz darin, dass die Kindelemente in unterschiedlicher Reihenfolge auftreten dürfen. Außerdem sind die minOccurs- und maxOccurs-Werte auf Null und Eins beschränkt.

ELaX-Operation			Schemaevolutionsschritt			Instanzanpassung	Umsetzung im Beispiel
1. add	1.2 addattributegroup	1.2.1 addattributegroupdef	1.2.1 Hinzufügen einer Attributgruppen Definition			nein	ja, tour
		1.2.2 addattribute	1.2.2 Hinzufügen einer Attributdeklaration			nein	ja, start, ziel
		1.2.3 addattributeref	1.2.3 Hinzufügen einer Attributdeklaration Referenz	1.2.3.1 "prohibited"		nein	ja, analog zu 1.2.3.3
				1.2.3.3 "required"		ja	ja, start und ziel (tour)
		1.2.4 addattributegroupref	1.2.4 Hinzufügen einer Attributgruppen Definition Referenz			ja/nein	ja, tour (routenType)
	1.3 addgroup		1.3 Hinzufügen einer Gruppendefinition	1.3.1 Sequenz	1.3.1.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, analog zu 1.3.1.2
					1.3.1.2 minOccurs > 0 maxOccurs > 0	nein	ja, fernbusanbieterType
				1.3.2 Alternative	1.3.2.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, analog zu 1.3.1.2
					1.3.2.2 minOccurs > 0 maxOccurs > 0	nein	ja, analog zu 1.3.1.2
				1.3.3 Menge	1.3.3.1 minOccurs = 0 maxOccurs = 1	nein	ja, analog zu 1.3.1.2
					1.3.3.2 minOccurs = 1 maxOccurs = 1	nein	ja, analog zu 1.3.1.2
	1.6 addelement	1.6.1 addelementdef	1.6.1 Hinzufügen einer Elementdeklaration			nein	ja, routen, fernbusanbieter
		1.6.2 addelementref	1.6.2 Hinzufügen einer Elementdeklaration Referenz in	1.6.2.1 Sequenz	1.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja	ja, fernbusanbieter (fernbusverkehrType), anbieter(fernbusanbieterType)
				1.6.2.2 Alternative	1.6.2.2.2 minOccurs > 0 maxOccurs > 0	nein	ja, analog zu 1.6.2.1.2
				1.6.2.3 Menge	1.6.2.3.2 minOccurs > 0 maxOccurs > 0	ja	ja, analog zu 1.6.2.1.2
	1.8 addconstraint		1.8 Hinzufügen einer Constraintdeklaration	1.8.1 Hinzufügen eines Primärschlüssels	1.8.1.1 key	ja	ja, routenKey (fernbusverkehr)
					1.8.1.2 unique	ja	ja, analog zu 1.8.1.1

Abbildung 5.3: ELaX-Operation: add für Suchoptimierung abgeleitet aus B

Somit sind die beiden Operationen *1.3.3.1* und *1.3.3.2* zur Operation *1.3.1.2* äquivalent, wenn zum Einen die maxOccurs-Werte der Sequenz auf Eins beschränkt werden und zum Anderen die Menge die gleiche Elementreihenfolge wie die Sequenz einhält. Im Fall minOccurs="0" wäre ebenfalls das Vorhandensein von gleichen minOccurs-Werten in den Elementreferenzen nötig.
Bei der Einfügeoperation *1.6.2.1.2* werden Elementreferenzen mit mindestens einem Vorkommen in die Sequenz hinzugefügt. Dabei ist das Einfügen in Alternativen äquivalent, wenn die Sequenz und die Alternative einelementig sind (siehe Operation *1.6.2.2*). Bei der Menge ist das Hinzufügen der Elementreferenz zur Sequenz äquivalent, wenn diese in derselben Reihenfolge auftritt, wie in der Sequenz.
Das *Hinzufügen eines Primärschlüssels (1.8.1)* unterteilt sich in key und unique. Dabei ist unique ein optionaler Primärschlüssel, welcher zum Einen nur angegebene Schemakomponentwerte überprüft und zum Anderen die Überprüfung der Werte nicht im gesamten XML-Dokument ausweitet. Deshalb ist das unique-Contraint eine Vereinfachung des key-Constraints und die Operation *1.8.1.2* analog zu *1.8.1.1*.

ELaX-Operation: delete

Die zuvor entwickelten Löschoperationen sind in Abbildung 5.4 in die Kategorien *delgroup (2.3)* und *delelement (2.6)* eingeteilt worden.

ELaX-Operation			Schemaevolutionsschritt			Instanzanpassung	Umsetzung im Beispiel
2. delete	2.3 delgroup		2.3 Entfernen einer Gruppendefinition	2.3.1 Sequenz	2.3.1.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.3.1.2
					2.3.1.2 minOccurs > 0 maxOccurs > 0	ja/nein	ja, fernbuslinieType
				2.3.2 Alternative	2.3.2.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.3.1.2
					2.3.2.2 minOccurs > 0 maxOccurs > 0	ja/nein	ja, analog zu 2.3.1.2
				2.3.3 Menge	2.3.3.1 minOccurs = 0 maxOccurs = 1	ja/nein	ja, analog zu 2.3.1.2
					2.3.3.2 minOccurs = 1 maxOccurs = 1	ja/nein	ja, analog zu 2.3.1.2
	2.6 delelement	2.6.1 delelementdef	2.6.1 Entfernen einer Elementdeklaration			ja/nein	ja, fernbuslinie

Abbildung 5.4: ELaX-Operation: delete für Suchoptimierung abgeleitet aus B

Die Löschoperation *2.3.1.2* entfernt die Gruppendefinition Sequenz aus dem „fernbuslinieType". Dabei sind die Löschoperationen für die Sequenz mit minOccurs="0" (siehe Operation *2.3.1.1*), für die Alternative (siehe Operationen *2.3.2.1* und *2.3.2.2*) und für die Menge (siehe Operationen *2.3.3.1* und *2.3.3.2*) analog zur Operation *2.3.1.2*, wobei die Erklärungen aus dem vorherigen Abschnitt 5.2.1 entnommen werden können.

ELaX-Operation: update

Die zuvor entwickelten Änderungsoperationen sind in Abbildung 5.5 in die Kategorien *updattributegroup (3.2)*, *updct (3.5)* und *updelement (3.6)* eingeteilt worden.

ELaX-Operation			Schemaevolutionsschritt		Instanzanpassung	Umsetzung im Beispiel
3. update	3.2 updattributegroup	3.2.1 updattributegroupdef	3.2.1 Ändern einer Attributgruppen Definition	3.2.1.2 Hinzufügen einer Attributdeklaration Referenz	ja/nein	ja, analog zu 1.2.3.3
	3.4 updst		3.4 Ändern einer einfachen Typdefinition	3.4.1 Umbenennen der Typdefinition	nein	ja, analog zu 3.5.1
	3.5 updct		3.5 Ändern einer komplexen Typdefinition	3.5.1 Umbenennen der Typdefinition	nein	ja, fahrplanType in routenType
	3.6 updelement	3.6.2 updelementref	3.6.2 Ändern einer Elementdeklaration Referenz	3.6.2.1 Ändern des Referenznamen	ja/nein	ja, fernbuslinie in routen (fernbusverkehrType)
				3.6.2.4 Verringern des maximalen Vorkommens	ja	ja, analog zu 3.6.2.5
				3.6.2.5 Erhöhen des maximalen Vorkommens	nein	ja, routen (fernbusverkehrType)
		3.6.3 updelementwildcard	3.6.3 Ändern einer Wildcard	3.6.3.5 Verringern des maximalen Vorkommens	ja	ja, analog zu 3.6.2.5

Abbildung 5.5: ELaX-Operation: update für Suchoptimierung abgeleitet aus B

Hierbei ist das *Hinzufügen einer Attributdeklaration Referenz (3.2.1.2)* äquivalent zum Einfügen einer Attributdeklarationsreferenz (siehe Operation *1.2.3.3*), mit dem Unterschied, dass gleich mehrere Attribute damit referenziert werden können. Die Änderung des Name-Attributs bei komplexen Typen (siehe Operation *3.5.1*) ist vergleichbar mit der Umbenennung des gleichen Attributs bei einer einfachen Typdefinition (siehe Operation *3.4.1*). Wenn das maximale Vorkommen einer Elementdeklaration Referenz erhöht wird, können mehrere Elemente dieses Typs auftreten (siehe Operation *3.6.2.5*). Dazu ist zum Einen das *Verringern des maximalen Vorkommens (3.6.2.4)* analog, wenn die vorkommenden Elemente nicht entfernt werden müssen. Dadurch ist zum Anderen das *Verringern des maximalen Vorkommens (3.6.3.5)* äquivalent, weil das any-Element XML-Dokumente um Elemente erweitert, die im Schema vorkommen oder nicht Bestandteil des Schemas sind.

5.2.2 Strukturierung der Route

Die separate Speicherung von Routen und Fernbusanbietern aus dem vorherigen Abschnitt veranlasst zunächst eine Strukturierung der Routen, welche in den Abbildungen 5.6, 5.7 und 5.8 zusammenfassend dargestellt sind.
Die erste grundlegende Änderung umfasst die Unterscheidbarkeit der Route-Elemente in den jeweiligen Routen-Elemente. Dazu wird eine eindeutige ID gebraucht, welche durch das *Ändern der Typdefinition (3.2.2.2)* des RouteID-Attributs von „xs:string“ auf „xs:ID“ erzeugt wird.
Als nächstes soll in jeder Route eine Referenz auf einen Fernbusanbieter existieren. Diese stellt sicher, dass zu jeder Route eindeutig ein Fernbusanbieter zugeordnet werden kann. Deshalb wird in der komplexen Typdefinition „routeType“ die Elementreferenz „hinfahrt“ auf „fernbusanbieter-ref“ geändert, sowie das minOccurs-Attribut gelöscht (siehe Operationen *3.6.2.1* und *3.6.2.3*). Das referenzierte Element wird anschließend in das Schema eingefügt mit dem built-in Datentyp xs:IDREF (siehe Operation *1.8.4*).
Die Route-Elemente sollen nach den Vorstellungen der Verantwortlichen zusätzlich um einen Routenfahrplan erweitert werden, welcher die bisherigen Hinfahrt- und Rückfahrtszeiten beinhaltet. Für die Umsetzung werden zum Einen die Elementreferenz „rückfahrt“ im route-Type und zum Anderen das nicht mehr genutzte Fahrplan-Element in „routenfahrplan“ umbenannt (siehe Operationen *3.6.2.1* und *3.6.1.1*). Ähnlich wie bei der Fernbusanbieter-Referenz soll jedes Route-Elemente einen solchen Routenfahrplan besitzen, weshalb das minimale Vorkommen der Routenfahrplan-Referenz auf Eins erhöht wird (siehe Operation *3.6.2.3*). Damit schließlich die Hinfahrts- und Rückfahrtszeiten berücksichtigt werden können, wird als Erstes im Routenfahrplan-Element das Type-Attribut auf „routenfahrplanType“ umbenannt und die entsprechende komplexe Typdefinition eingefügt (siehe Operationen *3.6.1.2* und *1.3.1.2*).

In diesen Typ werden dann die Elementreferenzen „hinfahrt“ und „rückfahrt“ eingefügt (siehe Operation *1.6.2.1.1*). Für den Fall, dass keine Hinfahrt und Rückfahrt beim Routenfahrplan angegeben wird, brauchen sie auch nicht zwingend als Elemente im Schema zu erscheinen, weshalb beim Einfügen der Referenzen das minimale Vorkommen auf Null gesetzt wird. Die entsprechenden Elementdeklarationen befinden sich im ausgelagerten XML-Schema „zeit.xsd“, weshalb hier keine weiteren Elemente hinzugefügt werden müssen.
Durch eine Änderung der Elementdefinition „zeit“ im externen XML-Schema „zeit.xsd“ sollen Start- und Ziel-Attribute eingefügt werden, ähnlich zum Routen-Element. Diese Attribute sollen nach den Verantwortlichen identisch mit den Elementinhalten der Elemente „von“ und „nach“ sein. Deshalb wird zunächst das Constraint „routeKey“ vom Typ key in unique geändert, weil somit äquivalente Von- oder Nach-Orte zugelassen werden können, z.B. Warnemünde ist ein Stadtteil von Rostock und somit äquivalent zu Rostock (siehe Operation *3.8.2.1*). Anschließend werden zwei Key-Referenzen eingefügt um die Überprüfung des Zeit-Elements zu gewährleisten, d.h. zum Einen den „startzielKey“ für das Zeit-Element unter dem Element „hinfahrt“ und zum Anderen den „zielstartKey“ für das Zeit-Element unter dem Element „rückfahrt“ (siehe Operation *1.8.2*).
Aus der Überlegung der Verantwortlichen heraus, dass alle gespeicherten Preise in Zukunft immer mit der Mehrwertsteuer gespeichert werden, wird die entsprechende Attributreferenz gelöscht, wobei anschließend die überflüssige Attributdeklaration ebenfalls entfernt werden kann (siehe Operationen *2.2.3.3* und *2.2.2*). Außerdem wird die einfache Typdefinition „MwSt.Type“ nicht weiter benötigt und wird deshalb aus dem Schema gelöscht (siehe Operation *2.4.1*).
Die Anzeige der Preise ist zurzeit ungeordnet, was den Abgleich mit archivierten Preislisten erschweren könnte. Deshalb wird im „preiseType“ die ungeordnete Menge in eine Sequenz transformiert (siehe Operation *3.3.1.5*). Wenn unvollständige Preise vorhanden sind, sollen die fehlenden Preise aufgelistet werden, um ein späteres Hinzufügen zu erleichtern. In dem Fall wenn gar keine Preise vorhanden sind, dann sollen diese in gar keinen Fall angezeigt werden, um unnötige Informationen in der XML-Instanz zu vermeiden. Außerdem sollen mehrere Preisunterlemente zugelassen werden, z.B. können mehrere Sparpreise angeboten werden in Abhängigkeit von verschiedenen Zielgruppen. Somit wird im „preiseType“ das minimale Vorkommen der Sequenz reduziert und das maximale Vorkommen auf „unbounded“ erhöht (siehe Operationen *3.3.2* und *3.3.5*). Anschließend werden die minOccurs-Werte der Elementreferenzen „sparpreise“, „erwachsene“, „kinder“, „gruppen“, „tageskarte“ und „fahrradmitnahme“ erhöht, damit die getätigten Änderungen in der Sequenz auch Auswirkungen haben (siehe Operation *3.6.2.3*). Außerdem soll bei fehlenden Preisen nicht nur „0“ Euro drin stehen, sondern „keine Angabe“, weshalb zunächst bei allen Preis-Elementen der Typ von geldType auf geldneuType geändert wird (siehe Operation *3.6.1.2*). Diese Änderung erfordert das Einfügen einer entsprechenden einfachen Typdefinition, die eine Vereinigung aus dem alten „geldType“ und einem neuen „fehlType“ beinhaltet (siehe Operation *1.4.3*). Letztere Typdefinition wird neu hinzugefügt und beschränkt den built-in Datentyp xs:string auf „keine Angabe“ (siehe Operation *1.4.1*).
Die Verantwortlichen wollen ab sofort den Umgang mit den auf den Routen verkehrenden Buslinien kontrollieren, d.h. die frei wählbaren Busunternehmen sollen auf eine erweiterbare Liste reduziert werden. Dazu wird im „buslinienType“ der built-in Datentyp xs:string aus der Vereinigung gelöscht, damit diese einelementig ist und in eine Liste umgewandelt werden kann (siehe Operationen *3.4.4* und *3.4.2.6*). Aufgrund der letzten Änderung werden Listenelemente durch Leerzeichen getrennt, weshalb alle Namen der Busunternehmen mit Leerzeichen angepasst werden müssen, d.h. die enumeration-Werte werden modifiziert (siehe Operation *3.4.7*).

ELaX-Operation: add

Die soeben entwickelten Einfügeoperationen unterteilen sich in die Kategorien *addgroup (1.3)*, *addst (1.4)*, *addelement (1.6)* und *addconstraint (1.8)*, dargestellt in Abbildung 5.6.
Das *Hinzufügen einer einfachen Typdefinition in Hierarchie als Liste (1.4.2)* bedeutet, dass eine leerzeichenseparierte Menge einer einfachen Typdefinition erlaubt wird.

ELaX-Operation			Schemaevolutionsschritt			Instanzanpassung	Umsetzung im Beispiel
1. add	1.3 addgroup		1.3 Hinzufügen einer Gruppendefinition	1.3.1 Sequenz	1.3.1.2 minOccurs > 0 maxOccurs > 0	nein	ja, routenfahrplanType
	1.4 addst		1.4 Hinzufügen einer einfachen Typdefinition in Hierarchie als	1.4.1 Einschränkung		nein	ja, fehlType
				1.4.2 Liste		nein	ja, analog zu 1.4.1
				1.4.3 Vereinigung		nein	ja, geldneuType
	1.6 addelement	1.6.2 addelementref	1.6.2 Hinzufügen einer Elementdeklaration Referenz in	1.6.2.1 Sequenz	1.6.2.1.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, hinfahrt und rückfahrt (routenfahrplanType)
				1.6.2.2 Alternative	1.6.2.2.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, analog zu 1.6.2.1.1
				1.6.2.3 Menge	1.6.2.3.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, analog zu 1.6.2.1.1
	1.8 addconstraint		1.8 Hinzufügen einer Constraintdeklaration	1.8.2 Hinzufügen eines Fremdschlüssels (keyref)		ja	ja, startzielKey und zielstartKey
				1.8.4 Hinzufügen einer Identitätsreferenz		ja	ja, fernbusanbieter-ref

Abbildung 5.6: ELaX-Operation: add für Strukturierung der Route abgeleitet aus B

Dieses kann verglichen werden mit dem Einfügen einer Einschränkung eines beliebigen Typs. Wenn diese Einschränkung Listenelemente beispielsweise durch Enumerationswerte simuliert bzw. die Liste nur einelementig ist, dann ist die Operation *1.4.2* äquivalent zu *1.4.1*, wobei hier eine Einschränkung eines beliebigen einfachen Typs erfolgt.
Beim *Hinzufügen einer Elementdeklaration Referenz in (1.6.2)* werden die drei Fälle Sequenz, Alternative und Menge unterschieden. Die Sequenz erzwingt die angegebene Reihenfolge der Elemente im Schema. Im Fall von Operation *1.6.2.1.1* wird durch die Angabe minOccurs=„0“ erlaubt, dass die Sequenz leer sein darf. Diese Semantik lässt sich ebenfalls auf die Alternative *1.6.2.2.1* und Menge *1.6.2.3.1* übertragen. Die Alternative stimmt mit der Sequenz überein, wenn beide komplett leer sind oder wenn beide dieselben Elemente repräsentieren, d.h. die Alternative entweder einelementig oder standardmäßig die Alternative auswählt, die mit der Sequenz übereinstimmt. Ebenfalls stimmt die Menge mit der Sequenz überein, wenn zum Einen beide keine Elemente enthalten und zum Anderen die ausgewählte Elementreihenfolge der Menge mit der von der Sequenz übereinstimmt.

ELaX-Operation: delete

Die zuvor entwickelten Löschoperationen werden in die Kategorien *delattributegroup (2.2)*, *delst (2.4)*, *delct (2.5)* und *delconstraint (2.8)* eingeteilt, abgebildet in Abbildung 5.7.

ELaX-Operation			Schemaevolutionsschritt		Instanzanpassung	Umsetzung im Beispiel
2. delete	2.2 delattributegroup	2.2.1 delattributegroupdef	2.2.1 Entfernen einer Attributgruppen Definition		ja/nein	ja, analog zu 2.2.2
		2.2.2 delattribute	2.2.2 Entfernen einer Attributdeklaration		ja/nein	ja, MwSt.
		2.2.3 delattributeref	2.2.3 Entfernen einer Attributdeklaration Referenz	2.2.3.1 "prohibited"	nein	ja, analog zu 2.2.3.3
				2.2.3.2 "optional"	ja/nein	ja, analog zu 2.2.3.3
				2.2.3.3 "required"	ja	ja, MwSt. (preiseType)
		2.2.4 delattributegroupref	2.2.4 Entfernen einer Attributgruppen Definition Referenz		ja/nein	ja, analog zu 2.2.3.3
		2.2.5 delattributewildcard	2.2.5 Entfernen einer Attribut-Wildcard		ja/nein	ja, analog zu 2.2.3.3
	2.4 delst		2.4 Entfernen einer einfachen Typdefinition aus Hierarchie	2.4.1 Einschränkung	nein	ja, MwSt.Type
	2.5 delct		2.5 Entfernen einer komplexen Typdefinition aus Hierarchie	2.5.1 Einschränkung: simpleContent	nein	ja, analog zu 2.4.1
				2.5.2 Einschränkung:complexContent	nein	ja, analog zu 2.4.1
	2.8 delconstraint		2.8 Entfernen einer Constraintdeklaration	2.8.3 Entfernen der Identität	ja/nein	ja, analog zu 2.2.2
				2.8.4 Entfernen einer Identitätsreferenz	ja	ja, analog zu 2.2.2

Abbildung 5.7: ELaX-Operation: delete für Strukturierung der Route abgeleitet aus B

Das *Entfernen einer Attributdeklaration (2.2.2)* bedeutet, dass eine globale Attribut Definition aus dem XML-Schema gelöscht wird. Hierzu ist zum Einen die Operation *2.2.1* äquivalent, weil Attributgruppendefinitionen ebenfalls global vorliegen und ähnlich gelöscht werden. Zum Anderen sind die Operationen *2.8.3* und *2.8.4* zu *2.2.2* analog, aufgrund der Tatsache das Identitäten und Identitätsreferenzen Attributdeklarationen sind.

Wird eine zwingend erforderliche Attributdeklarationsreferenz entfernt, so muss diese in der XML-Instanz ebenfalls gelöscht werden (siehe Operation *2.2.3.3*). Dazu äquivalent wären das Löschen von optionalen oder verbotenen Attributreferenzen, weil diese nur im Falle der Angabe beim optionalen Attribut gelöscht werden müssten. Ebenfalls äquivalent wäre zum Einen das Entfernen von Attributgruppenreferenzen *2.2.4*, da hier mehrere Attribute im Element gelöscht werden könnten, je nach Verwendung (use). Zum Anderen ist das *Entfernen einer Attribut-Wildcard (2.2.5)* zu *2.2.3.3* analog, weil die Wildcard das XML-Schema um beliebige Attribute erweitert, die im Schema vorkommen oder nicht Bestandteil dessen sind.
Beim *Entfernen einer einfachen Typdefinition aus Hierarchie (2.4)* wird mit der Operation *2.4.1* eine Einschränkung gelöscht, d.h. global wird der Typ aus dem XML-Schema entfernt. Äquivalent dazu sind die Operationen *2.5.1* und *2.5.2* von komplexen Typdefinitionen, die Restriktionen von simpleContent oder complexContent enthalten. Da diese komplexen Typdefinitionen ebenfalls global definiert sind, können diese ohne weiteres aus dem XML-Schema entfernt werden.

ELaX-Operation: update

Auf Grundlage der erarbeiteten Anpassungsoperationen wurden diese in die Katerogien *updattributegroup (3.2)*, *updgroup (3.3)*, *updst (3.4)*, *updelement (3.6)*, *updmodule (3.7)* und *updconstraint (3.8)* einsortiert, dargestellt in 5.8.

ELaX-Operation			Schemaevolutionsschritt			Instanzanpassung	Umsetzung im Beispiel
3. update	3.2 updattributegroup	3.2.1 updattributegroupdef	3.2.1 Ändern einer Attributgruppen Definition	3.2.1.3 Entfernen einer Attributdeklaration Referenz		ja/nein	ja, analog zu 2.2.3.3
				3.2.1.5 Entfernen einer Attribut-Wildcard		ja/nein	ja, analog zu 2.2.3.3
		3.2.2 updattribute	3.2.2 Ändern einer Attributdeklaration	3.2.2.2 Ändern der Typdefinition		ja/nein	ja, routeID
	3.3 updgroup		3.3 Ändern einer Gruppendefinition	3.3.1 Ändern des Typs	3.3.1.1 Sequenz -> Menge	nein	ja, analog zu 3.3.1.5
					3.3.1.5 Menge -> Sequenz	ja/nein	ja, preiseType
				3.3.2 Verringern des minimalen Vorkommens		nein	ja, preiseType
				3.3.3 Erhöhen des minimalen Vorkommens		ja/nein	ja, analog zu 3.3.2
				3.3.4 Verringern des maximalen Vorkommens		ja/nein	ja, analog zu 3.3.5
				3.3.5 Erhöhen des maximalen Vorkommens		nein	ja, preiseType
	3.4 updst		3.4 Ändern einer einfachen Typdefinition	3.4.2 Ändern des Typs	3.4.2.1 atomar -> Liste	nein	ja, analog zu 1.4.3
					3.4.2.2 atomar -> Vereinigung	nein	ja, analog zu 1.4.3
					3.4.2.3 Liste -> Vereinigung	ja/nein	ja, analog zu 3.4.2.6
					3.4.2.4 Liste -> atomar	ja/nein	ja, analog zu 3.4.2.6
					3.4.2.5 Vereinigung -> atomar	ja/nein	ja, analog zu 3.4.2.6
					3.4.2.6 Vereinigung -> Liste	ja/nein	ja, buslinienType
				3.4.3 Hinzufügen einer einfachen Typdefinition zu einer Vereinigung		nein	ja, analog zu 3.4.4
				3.4.4 Entfernen einer einfachen Typdefinition aus einer Vereinigung		ja/nein	ja, buslinienType
				3.4.5 Hinzufügen einer Einschränkung		ja/nein	ja, analog zu 3.4.7
				3.4.6 Entfernen einer Einschränkung		nein	ja, analog zu 3.4.7
				3.4.7 Modifizieren einer Einschränkung		ja/nein	ja, buslinieType
	3.6 updelement	3.6.1 updelementdef	3.6.1 Ändern einer Elementdeklaration	3.6.1.1 Umbenennen einer Elementdeklaration		ja	ja, fahrplan in routenfahrplan
				3.6.1.2 Ändern der Typdefinition		ja/nein	ja, routenfahrplan, sparpreise, erwachsene, kinder, gruppen, tageskarte, fahrradmitnahme
		3.6.2 updelementref	3.6.2 Ändern einer Elementdeklaration Referenz	3.6.2.1 Ändern des Referenznamen		ja/nein	ja, hinfahrt in fernbusanbieter-ref und rückfahrt in routenfahrplan (routeType)
				3.6.2.2 Verringern des minimalen Vorkommens		nein	ja, analog zu 3.6.2.3
				3.6.2.3 Erhöhen des minimalen Vorkommens		ja	ja, fernbusanbieter-ref und routenfahrplan (routeType), sparpreise erwachsene, kinder, gruppen, tageskarte, fahrradmitnahme (preiseType)
		3.6.3 updelementwildcard	3.6.3 Ändern einer Wildcard	3.6.3.3 Verringern des minimalen Vorkommens		nein	ja, analog zu 3.6.2.3
				3.6.3.4 Erhöhen des minimalen Vorkommens		ja	ja, analog zu 3.6.2.3
	3.7 updmodule		3.7 Ändern einer Moduldeklaration	3.7.1 Umbenennen der Moduldeklaration		nein	ja, analog zu 3.6.1.1
	3.8 updconstraint		3.8 Ändern einer Constraintdeklaration	3.8.2 Ändern des Schlüsseltyps	3.8.2.1 key -> unique	nein	ja, routeKey
					3.8.2.2 key -> keyref	ja/nein	ja, analog zu 3.8.2.1
					3.8.2.3 keyref -> key	ja/nein	ja, analog zu 3.8.2.1
					3.8.2.4 keyref -> unique	ja/nein	ja, analog zu 3.8.2.1
					3.8.2.5 unique -> key	ja/nein	ja, analog zu 3.8.2.1
					3.8.2.6 unique -> keyref	ja/nein	ja, analog zu 3.8.2.1
				3.8.3 Ändern des Identitätstyps	3.8.3.1 id -> idref	ja	ja, analog zu 3.8.2.1
					3.8.3.2 idref -> id	ja	ja, analog zu 3.8.2.1

Abbildung 5.8: ELaX-Operation: update für Strukturierung der Route abgeleitet aus B

Das *Ändern einer Attributgruppen Definition(3.2.1)*, insbesondere das Entfernen von Attributdeklarationsreferenzen *3.2.1.3* und Attribut-Wildcards *3.2.1.5* sind äquivalent zur Operation *2.2.3.3*. Die Gründe dafür sind im vorherigen Abschnitt für die Operationen *2.2.4* und *2.2.5* erläutert worden.
Wenn der Typ beim *Ändern einer Gruppendefinition (3.3)* von einer Menge zu Sequenz geändert wird (siehe Operation *3.3.1.5*), dann muss die Elementreihenfolge der Menge mit der in der Sequenz übereinstimmen und gegebenenfalls in der XML-Instanz angepasst werden. Äquivalent dazu ist die Änderung von Sequenz in eine Menge (siehe Operation *3.3.1.1*).

Die Elementreihenfolge ist nicht mehr fest vor geschrieben und deshalb muss auch keine Instanzanpassung durchgeführt werden. Weiterhin ist das *Erhöhen des minimalen Vorkommens (3.3.3)* analog zur Reduzierung desselbigen (siehe Operation *3.3.2*). Diese Analogie entsteht, wenn das Erhöhen keine Instanzanpassung hervorruft und somit keine Schemakomponenten aufgrund von Vorkommensangaben hinzugefügt werden müssen. Ähnlich verhält es sich mit dem *Verringern des maximalen Vorkommens (3.3.4)*, welches äquivalent zum Erhöhen ist (siehe Operation *3.3.5*). Hierbei werden nur dann Schemakomponenten entfernt, wenn deren Vorkommen größer ist als der neu geänderte maxOccurs-Wert. Wird das Löschen nicht erforderlich, dann sind die beiden zuletzt genannten Operationen von der Auswirkung identisch.
Beim *Ändern des Typs (3.4.2)* einer einfachen Typdefinition können zunächst atomare oder built-in Typen in Listen bzw. Vereinigungen transformiert werden (siehe Operationen *3.4.2.1* und *3.4.2.2*). Diese sind äquivalent zum Einfügen des „geldneuType“ in Operation *1.4.3*. Hierbei wurde eine Einschränkung eines atomaren Typs xs:decimal zu einer Vereinigung erweitert. Da die beiden zuletzt genannten Operationen, jeweils den atomaren Typ erweitern, lösen sie damit keine Instanzanpassungen aus. Weiterhin existieren Analogien der Operationen *3.4.2.3*, *3.4.2.4* und *3.4.2.5* zur Änderung des Typs union zu list, die beim „buslinienType“ erfolgte (siehe Operation *3.4.2.6*). Dabei bedeutet die Änderung von Vereinigung zu Liste, dass gegebenenfalls bei mehr als einer einfachen Typdefinition in memberTypes der union diese gelöscht werden müssen (siehe Operation *3.4.4*). Dadurch findet eine Typeinschränkung statt, welche vergleichbar ist mit der Operation *3.4.2.5*. Die Operationen *3.4.2.3* und *3.4.2.4* führen ebenfalls zu einer Typeinschränkung, wenn entweder die Vereinigung nur eine Typdefinition beinhaltet oder der atomare Typ einen begrenzten built-in Typ verwendet. Weiterhin ist das *Hinzufügen einer einfachen Typdefinition zu einer Vereinigung (3.4.3)* analog zu *3.4.4*, weil die Typerweiterung keinen Einfluss auf die bereits bestehenden Schemakomponenten hat. Außerdem können bei Typänderungen und/oder potentiellen Änderungswünschen die getätigten Einschränkungen (Facetten) gelöscht, modifiziert oder neu hinzugefügt werden. Dabei können die Operationen *3.4.5* und *3.4.6* äquivalent zum Modifizieren einer Einschränkung *3.4.7* ausgeführt werden, solange die Anzahl der Facetten der betrachteten Quell- und Zieltypen identisch sind. Somit wäre eine Modifizierung ausdrückbar als eine Löschoperation, die die alte Facette löscht, und eine Einfügeoperation, die die neue Facette dem Schema hinzufügt.
Das *Ändern einer Elementdeklaration Referenz (3.6.2)* beinhaltet die Erhöhung des minimalen Vorkommens, was unter Umständen ein Hinzufügen bereits fehlender Elemente zur Folge hätte (siehe Operation *3.6.2.3*). Falls keine neuen Elemente trotz Erhöhung des minOccurs-Wertes eingefügt werden müssen, dann ist das *Verringern des minimalen Vorkommen (3.6.2.2)* äquivalent zu *3.6.2.3*. Letzteres hätte ebenfalls zur Folge, dass keine weiteren Elemente eingefügt werden müssten, weil durch die Verringerung die Möglichkeit zum Löschen dieser besteht. Die gleiche Argumentation kann auf die Wildcard any übertragen werden, sodass die Operationen *3.6.3.3* und *3.6.3.4* analog zu *3.6.2.3* sind.
Das Umbenennen einer Moduldeklaration in Operation *3.7.1* ist analog mit dem *Umbenennen einer Elementdeklaration (3.6.1.1)*, da bei beiden dadurch neue Semantik entsteht. Im Fall des Elements wird sofort eine Instanzanpassung notwendig, wohingegen beim Modul sich das schemaLocation-Attribut wahlweise das namespace-Attribut ändert, wodurch keine direkte Schemaanpassung nötig wird.
Beim *Ändern einer Constraintdeklaration (3.8)* können Schlüsseltypen *3.8.2* und *3.8.3* geändert werden. Wenn der Schlüsseltyp vom primären Schlüssel zum optionalen Primärschlüssel modifiziert wird, dann wird nicht mehr jedes Von- und Nach-Element aller Route-Elemente überprüft (siehe Operation *3.8.2.1*). Äquivalent wäre dazu die Umkehrung (siehe Operation *3.8.2.5*), wobei dann unter Umständen im XML-Schema Anpassungen vorgenommen werden müssen, weil die Schlüsseleigenschaft verletzt sein kann. Wenn ein (optionaler) Primärschlüssel zu einem Fremdschlüssel transformiert wird (siehe Operationen *3.8.2.2* und *3.8.2.6*), dann werden die bisher unter key bzw. unique stehenden Schemakomponenten nicht weiter betrachtet. Diese Art der Transformation kann deshalb mit der von key zu unique und somit mit *3.8.2.1* verglichen werden. Außerdem können Fremdschlüssel in (optionale) Primärschlüssel geändert werden, wobei hier darauf geachtet werden muss, dass bei einem key alle neuen unter Schlüsseleigenschaft stehenden Schemakomponenten möglicherweise angepasst werden müssen (siehe Operationen *3.8.2.2* und *3.8.2.3*).

Ebenfalls äquivalent ist die Betrachtung der Typänderung bei Identitäten, d.h. wenn eine zuvor zwingend erforderliche ID eine Referenz wird (siehe Operation *3.8.3.1*), so kann sie in der Instanz mehrfach auftreten (analog Operation *3.8.2.1*), wobei key als id und unique als idref angesehen werden kann. Außerdem ist die Umkehrung ebenfalls äquivalent zu *3.8.2.1*, falls nur einzigartige idrefs in der Instanz vorkamen und somit keine Änderungen hinsichtlich der id notwendig sind (siehe Operation *3.8.3.2*).

5.2.3 Grundlegende Überarbeitung der Anbieterdaten

Die Verantwortlichen sehen nach der Auslagerung der Anbieterdaten in das Fernbusanbieter-Element den Bedarf der Integration der bis jetzt nicht im neuen Schema auftauchenden Ausstattung- und Beschreibung-Elementes in das Fernbusanbieter-Element bzw. Anbieter-Element.
Als Erstes wird sich der Integration des Ausstattung-Elementes angenommen, welches in Zukunft direkt unter dem Element „anbieter“ vorkommen soll. Dazu soll der bereits existierende Anbieter-Typ erweitert werden. Dieser muss zunächst seine Abgeschlossenheit ändern, damit er auch erweiterbar gestaltet werden kann, d.h. von # all auf restriction (siehe Operation *3.5.3*). Nach dieser Änderung ist der Anbieter-Typ zwar erweiterbar, aber kann nicht weiter beschränkt werden. Danach wird die komplexe Typdefinition „anbieterneuType“ eingefügt, um das Anbieter-Element zum Einen um das Ausstattung-Element und zum Anderen um eine „anbieterID“ zu erweitern (siehe Operationen *1.5.4*, *1.6.2* und *1.8.3*). Die ID ist ähnlich wie bei den Route-Elemente zur eindeutigen Unterscheidung der Anbieter-Elemente unter dem Fernbusanbiet-Element. Damit diese Änderungen Auswirkungen auf das Anbieter-Element haben, muss das type-Attribut auf „anbieterneuType“ geändert werden (siehe Operation *3.6.1.2*).
Die Ausstattung soll nicht wie bisher im gleichen Schema wie die Anbieterdaten gespeichert werden, sodass ein separates Hinzufügen von Ausstattungseigenschaften bzw. -informationen erfolgen kann, die dann nachgepflegt werden können. Den Verantwortlichen schwebt in naher Zukunft das Hinzufügen von Busausstattungen vor, wie z.B. „5 Sterne Bus“ und die damit verbundenen Ausstattungsmerkmale. Dazu wird ein neues XML-Schema mit dem targetNamespace-Attribut „http://www.ausstattung.org“ erstellt, welches von nun an initial die Elemente „catering“, „sitzabstand“, „steckdosen“ und „wlan“ enthält. Damit diese Informationen im aktuellen Schema referenziert werden können, wird eine entsprechende Wildcard (any) mit maxOccurs=„unbounded“ eingefügt, die eine Erweiterung um Elemente aus einem externen Schema erlaubt (siehe Operation *1.6.3.2*). Dabei erlaubt die Erhöhung des maximalen Vorkommens die Einbindung mehrerer ausgelagerter Elemente in das Schema bzw. Instanz. Die Verantwortlichen wollen im Falle nicht vorhandener Extras bzw. wenn alle Ausstattungsmerkmale verneint sind, die Möglichkeit haben, dass explizit in der Instanz ein Vermerk steht, dass das Ausstattung-Element leer ist. Dazu wird in der Elementdeklaration „ausstattung“ das nillable-Attribut mit dem Wahrheitswert „true“ eingefügt (siehe Operation *3.6.1.6.2*). Durch die Auslagerung der Ausstattungsunterelemente fallen zunächst die nicht mehr benötigten Elementreferenzen und dazugehörigen Elemente der Ausstattungselemente weg und können aus dem Schema gelöscht werden (siehe Operationen *2.6.2.1.2* und *2.6.1*). Weiterhin können die somit nicht mehr gebrauchten Typdefinitionen „serviceType“, „sitzabstandType“, „wahlType“ und „cateringType“ ebenfalls entfernt werden (siehe Operationem *2.4.1* und *2.4.2*).
Die zweite größere Änderung betrifft die Integration des Beschreibung-Elementes. Dieses soll nach dem Willen der Verantwortlichen nicht mehr direkt im Schema bzw. in der Instanz auftauchen, sondern im Namen des jeweiligen Anbieters, weshalb die Elementdeklaration „beschreibung“ gelöscht werden kann (siehe Operation *2.6.1*). In Zukunft soll immer erst der Name des Fernbusanbieters auftauchen, gefolgt von einer Kurzbeschreibung und optional einer Langbeschreibung. Da die vorhandene komplexe Typdefinition „beschreibungsType“ diese Funktion erfüllt wird diese als Erweiterungsbasis im komplexen Typ „nameType“ verwendet, wodurch sich die bisherige Einschränkung einer einfachen Typdefinition in die Erweiterung eines komplexen Typs ändert (siehe Operation *3.5.4.9*). Damit jedoch die Angabe des Namens und der Beschreibung(en) auf einmal erfolgen kann, muss das mixed-Attribut im Beschreibungs-Typ von „false“ auf „true“ gesetzt werden (siehe Operation *3.5.2.1*).
Bei der letzten Integration haben die Verantwortlichen festgestellt, dass sich der Name von Anbieter strukturell verschoben hat.

Deshalb muss der Selector-Pfad im Constraint „anbieterKey“ angepasst werden (siehe Operation *3.8.4.1*). Damit keine Verwechselung mit der neu eingefügten „anbieterID“ auftreten, wird das Constraint in „anbieternamKey“ umbenannt (siehe Operation *3.8.1*). Aus der Beobachtung heraus, dass sich die Abkürzungen aus den Anfangsbuchstaben der jeweiligen Einzelwörter bilden, z.B. BLB aus Berlin Linien Bus, haben sich die Verantwortlichen für die genauere Bezeichnung Akronym anstelle von Abkürzung entschieden (siehe Operation *3.2.2.1*). Dadurch, dass das Attribut jetzt „akronym“ heißt, muss anschließend eine Änderung der entsprechenden Attributreferenz erfolgen, damit die Änderung auch Auswirkung hat (siehe Operation *3.2.3.1*). Außerdem haben die Verantwortlichen nach Betrachtung der Gründungszahlen der Fernbusanbieter geschlossen, dass die ersten Fernbuslinien Anfang des 20.Jahrhunderts eingerichtet wurden. Deshalb soll „1900“ als Startjahr gesetzt werden, sodass der Wertebereich die Jahreszahlen ab dem Startjahr umfasst. Dazu wird ein neuer einfacher Typ „gründungsjahrType“ als Einschränkung eingefügt, welcher auf Basis des built-in Datentyps xs:gYear den Wertebereich über die Facette „minInclusive“ auf die Jahre ab einschließlich 1900 beschränkt (siehe Operationen *1.4.1* und *3.4.7*). Anschließend muss der Elementtyp beim Gründungsjahr-Element von „xs:gYear“ auf „gründungsjahrType“ transformiert werden, damit die Änderungen wirksam werden (siehe Operation *3.6.1.2*).

ELaX-Operation: add

Die zuvor entwickelten Einfügeoperationen gliedern sich in die Kategorien *addattributegroup (1.2)*, *addct (1.5)*, *addelement (1.6)* und *addconstraint (1.8)* ein, dargestellt in Abbildung 5.9.

ELaX-Operation			Schemaevolutionsschritt			Instanzanpassung	Umsetzung im Beispiel
1. add	1.2 addattributegroup	1.2.5 addattributewildcard	1.2.5 Hinzufügen einer Attribut-Wildcard			nein	ja, analog zu 1.6.3.2
	1.4 addst	1.4 Hinzufügen einer einfachen Typdefinition in Hierarchie als	1.4.1 Einschränkung			nein	ja, gründungsjahrType
	1.5 addct		1.5 Hinzufügen einer komplexen Typdefinition in Hierarchie als	1.5.1 Einschränkung: simpleContent		nein	ja, analog zu 1.5.4
				1.5.2 Einschränkung:complexContent		nein	ja, analog zu 1.5.4
				1.5.3 Erweiterung: simpleContent		nein	ja, analog zu 1.5.4
				1.5.4 Erweiterung: complexContent		nein	ja, anbieterneuType
	1.6 addelement	1.6.2 addelementref	1.6.2 Hinzufügen einer Elementdeklaration Referenz in	1.6.2.1 Sequenz	1.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja	ja, ausstattung (anbieterneuType)
		1.6.3 addelementwildcard	1.6.3 Hinzufügen einer Wildcard	1.6.3.1 minOccurs = 0 maxOccurs ≥ 0		nein	ja, analog zu 1.6.3.2
				1.6.3.2 minOccurs > 0 maxOccurs > 0		ja	ja, any (ausstattungType)
	1.8 addconstraint		1.8 Hinzufügen einer Constraintdeklaration	1.8.3 Hinzufügen einer Identität		nein	ja, anbieterID

Abbildung 5.9: ELaX-Operation: add für Überarbeitung der Anbieterdaten abgeleitet aus B

Das *Hinzufügen einer Wildcard (1.6.3)* kann zunächst mit einem minimalen Vorkommen größer als Null eingefügt werden, sodass es in jedem Fall in der Instanz auftauchen muss (siehe Operation *1.6.3.2*). Wenn das minimale Vorkommen gleich Null ist (siehe Operation *1.6.3.1*), muss das Any-Element nicht in der Instanz vorkommen und ist somit ein Spezialfall von Operation *1.6.3.2*. Weiterhin äquivalent zum Einfügen eines Any-Elements ist das Einfügen eines Any-Attributs (siehe Operation *1.2.5*). Dabei gleichen sich beide Wildcards in ihrer Funktion, d.h. das anyAttribute bzw. any können Attribute bzw. Elemente, dem Schema hinzufügen, die zum Einem schon im Schema vorkommen oder nicht Bestandteil dessen sind.
Wird eine komplexe Typdefinition hinzugefügt, so können vier Spezialfälle eintreten. Im Beispiel wurde ein komplexer Typ eingefügt, welcher einen anderen komplexen Typ erweitert hat (siehe Operation *1.5.4*). Dabei können beispielsweise neue Element- oder Attributreferenzen als Erweiterung hinzugefügt werden. Letzteres wird häufig beim Hinzufügen komplexer Typen verwendet, die eine einfache Typdefinition erweitern sollen, weshalb die Operation *1.5.3* analog zu *1.5.4* angesehen werden kann. Außerdem sind die beiden Operationen *1.5.1* und *1.5.2* äquivalent zu *1.5.4*, weil sie ebenfalls auf Basis von einfachen bzw. komplexen Typdefinitionen arbeiten, allerdings mit dem Unterschied, dass sie diese weiter einschränken. Eine weitere Gemeinsamkeit zwischen den Operationen *1.5.1-1.5.4* ist, dass sie keine direkten Instanzanpassungen nach sich ziehen, weil sie noch nicht referenziert worden sind.

ELaX-Operation: delete

Die bisher entwickelten Löschoperationen können in die Kategorien *delst (2.4)*, *delct (2.5)*, *delelement (2.6)* und *delconstraint (2.8)* eingeteilt werden, dargestellt in 5.10.

ELaX-Operation			Schemaevolutionsschritt			Instanzanpassung	Umsetzung im Beispiel
2. delete	2.4 delst		2.4 Entfernen einer einfachen Typdefinition aus Hierarchie	2.4.1 Einschränkung		nein	ja, serviceType, sitzabstandType, wahlType
				2.4.2 Liste		ja/nein	ja, cateringType
				2.4.3 Vereinigung		ja/nein	ja, analog zu 2.4.2
	2.5 delct		2.5 Entfernen einer komplexen Typdefinition aus Hierarchie	2.5.3 Erweiterung: simpleContent		ja/nein	ja, analog zu 2.4.2
				2.5.4 Erweiterung: complexContent		ja/nein	ja, analog zu 2.4.2
	2.6 delelement	2.6.1 delelementdef	2.6.1 Entfernen einer Elementdeklaration			ja/nein	ja, catering, sitzabstand, steckdosen, wlan, beschreibung
		2.6.2 delelementref	2.6.2 Entfernen einer Elementdeklaration Referenz aus	2.6.2.1 Sequenz	2.6.2.1.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.6.2.1.2
					2.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja	ja, catering, sitzabstand, steckdosen, wlan (ausstattungType)
				2.6.2.2 Alternative	2.6.2.2.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.6.2.1.2
					2.6.2.2.2 minOccurs > 0 maxOccurs > 0	ja/nein	ja, analog zu 2.6.2.1.2
				2.6.2.3 Menge	2.6.2.3.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.6.2.1.2
					2.6.2.3.2 minOccurs > 0 maxOccurs > 0	ja	ja, analog zu 2.6.2.1.2
		2.6.3 delelementwildcard	2.6.3 Entfernen einer Wildcard	2.6.3.1 minOccurs = 0 maxOccurs ≥ 0		ja/nein	ja, analog zu 2.6.2.1.2
				2.6.3.2 minOccurs > 0 maxOccurs > 0		ja	ja, analog zu 2.6.2.1.2
	2.8 delconstraint		2.8 Entfernen einer Constraintdeklaration	2.8.1 Entfernen eines Primärschlüssels	2.8.1.1 key	nein	ja, analog zu 3.8.4.1
					2.8.1.2 unique	nein	ja, analog zu 3.8.4.1
				2.8.2 Entfernen eines Fremdschlüssels (keyref)		nein	ja, analog zu 3.8.4.1

Abbildung 5.10: ELaX-Operation: delete für Überarbeitung der Anbieterdaten abgeleitet aus B

Das *Entfernen einer einfachen Typdefinition aus Hierarchie (2.4)* wurde im Beispiel bei einer Liste in Operation *2.4.2* durchgeführt. Die Liste im XML-Schema ermöglicht eine Erweiterung von atomaren Typen derart, dass verschiedene Ausprägungen einer Typdefinition im Schema gebündelt erscheinen können. Wird die Liste nun entfernt, könnte der Typ nicht mehr referenziert und die Erweiterungen damit nicht mehr genutzt werden. Dazu äquivalent ist das Löschen einer Vereinigung (siehe Operation *2.4.3*), da diese mehrere einzelne einfache Typdefinitionen miteinander vereint und somit einen größeren Wertebereich ermöglicht, der nach dem Löschen nicht mehr zur Verfügung steht. Ähnlich verhalten sich die Operationen *2.5.3* und *2.5.4*, die entweder eine einfache oder komplexe Typdefinition erweitern und somit ebenfalls zusätzliche Informationen verloren gehen, wenn diese gelöscht werden.
Beim *Entfernen einer Elementdeklaration Referenz aus (2.6.2)* einer Sequenz mit mindestens einem Vorkommen (siehe Operation *2.6.2.1.2*) wird eine direkte Instanzanpassung ausgelöst, indem alle vorhandenen Elemente gelöscht werden müssen, da sie nicht mehr in der festgelegten Reihenfolge auftreten dürfen. Daneben können Elementreferenzen existieren, die nicht zwingend im Schema auftauchen müssen. Wenn diese optionalen Elemente in der Instanz vorhanden sind, müssen diese gelöscht werden, weshalb die Operation *2.6.2.1.1* analog zu *2.6.2.1.2* ist. Ebenfalls äquivalent zu *2.6.2.1.2* sind die Operationen *2.6.2.2.1*, *2.6.2.2.2*, *2.6.2.3.1* und *2.6.2.3.2*, wenn die Elementreferenzen in der Instanz auftauchen und deshalb gelöscht werden müssen. Die beiden Operationen *2.6.3.1* und *2.6.3.2* behandeln das *Entfernen einer Wildcard (2.6.3)*, wodurch eingefügte Elemente aus fremden Schemata und/oder aus dem Schema selbst entfernt werden müssen. Dies ist analog zu Operation *2.6.2.1.2*.
Durch die Update-Operation *Ändern des selector-Pfades (3.8.4.1)* wird eine nicht mehr benötigte Constraintanweisung in eine neue transformiert, die geänderten Ansprüchen genügt. Der erste Teil kann mit dem Löschen von (optionalen) Primärschlüsseln oder Fremdschlüsseln gleichgesetzt werden, weshalb die Operationen *2.8.1.1*, *2.8.1.2* und *2.8.2* analog zu *3.8.4.1* sind.

ELaX-Operation: update

Die in diesem Abschnitt entwickelten Anpassungsoperationen teilen sich in die Kategorien *updattributegroup (3.2)*, *updst (3.4)*, *updct (3.5)*, *updelement (3.6)*, *updconstraint (3.8)* und *updschema (3.9)* auf, abgebildet in Abbildung 5.11.
Das *Umbenennen einer Attributgruppen Definition (3.2.1.1)* und das *Ändern des Referenznamen (3.2.4.1)* einer Attributgruppenreferenz sind äquivalent zu den Operationen *3.2.2.1* und *3.2.3.1*.

ELaX-Operation			Schemaevolutionsschritt				Instanzanpassung	Umsetzung im Beispiel
3. update	3.2 upd attribute group	3.2.1updattribute groupdef	3.2.1 Ändern einer Attributgruppen Definition	3.2.1.1 Umbenennen einer Attributgruppen Definition			nein	ja, analog zu 3.2.2.1
				3.2.1.4 Hinzufügen einer Attribut-Wildcard			nein	ja, analog zu 1.6.3.2
		3.2.2updattribute	3.2.2 Ändern einer Attributdeklaration	3.2.2.1 Umbenennen einer Attributdeklaration			ja/nein	ja, abkürzung in akronym
		3.2.3updattributeref	3.2.3 Ändern einer Attributdeklaration Referenz	3.2.3.1 Ändern des Referenznamen			ja/nein	ja, abkürzung in akronym (nameType)
		3.2.4updattributegroupref	3.2.4 Ändern einer Attributgruppen Definition Referenz	3.2.4.1 Ändern des Referenznamen			ja/nein	ja, analog zu 3.2.3.1
		3.2.5updattributewildcard	3.2.5 Ändern einer Attribut-Wildcard	3.2.5.1 Ändern der Namensräume			ja/nein	ja, analog zu 1.6.3.2
				3.2.5.2 Ändern der Validierung			ja/nein	ja, analog zu 1.6.3.2
	3.4 updst		3.4 Ändern einer einfachen Typdefinition	3.4.7 Modifizieren einer Einschränkung			ja/nein	ja, gründungsjahrType
	3.5 updct		3.5 Ändern einer komplexen Typdefinition	3.5.2Ändern des Inhaltstyps auf	3.5.2.1 mixed = "true"		nein	ja, beschreibungType
					3.5.2.2 mixed = "false"		ja	ja, analog zu 3.5.2.1
				3.5.3 Ändern der Abgeschlossenheit			ja/nein	ja, anbieterType
				3.5.4 Ändern des Inhalts (und Basis)	3.5.4.1 Einschränkung: simpleContent -> Einschränkung: complexContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.2 Einschränkung: simpleContent -> Erweiterung: simpleContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.3 Einschränkung: simpleContent -> Erweiterung: complexContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.4 Einschränkung: complexContent ->Einschränkung: simpleContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.5 Einschränkung: complexContent -> Erweiterung: simpleContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.6 Einschränkung: complexContent -> Erweiterung: complexContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.7 Erweiterung: simpleContent -> Einschränkung: simpleContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.8 Erweiterung: simpleContent -> Einschränkung: complexContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.9 Erweiterung: simpleContent -> Erweiterung: complexContent		ja/nein	ja, nameType
					3.5.4.10 Erweiterung: complexContent -> Einschränkung: simpleContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.11 Erweiterung: complexContent -> Einschränkung: complexContent		ja/nein	ja, analog zu 3.5.4.9
					3.5.4.12 Erweiterung: complexContent -> Erweiterung: simpleContent		ja/nein	ja, analog zu 3.5.4.9
	3.6 upd element	3.6.1 upd elementdef	3.6.1 Ändern einer Elementdeklaration	3.6.1.2 Ändern der Typdefinition			ja/nein	ja, anbieter, gründungsjahr
				3.6.1.5 Ändern der Abgeschlossenheit			ja/nein	ja, analog zu 3.5.3
				3.6.1.6 Ändern der Nullwertfähigkeit	3.6.1.6.1 nillable= "false" einfügen	3.6.1.6.1 minOccurs ≥ 0 maxOccurs ≥ 0	nein	ja, analog zu 3.6.1.6.2
					3.6.1.6.2 nillable= "true" einfügen	3.6.1.6.2 minOccurs ≥ 0 maxOccurs ≥ 0	ja/nein	ja, ausstattung
					3.6.1.6.3 nillable= "true"->nillable="false"	3.6.1.6.3 minOccurs ≥ 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 3.6.1.6.2
					3.6.1.6.4 nillable= "false"->nillable="true"	3.6.1.6.4 minOccurs ≥ 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 3.6.1.6.2
					3.6.1.6.5 nillable= "false" entfernen	3.6.1.6.5 minOccurs ≥ 0 maxOccurs ≥ 0	nein	ja, analog zu 3.6.1.6.2
					3.6.1.6.6 nillable= "true" entfernen	3.6.1.6.6 minOccurs ≥ 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 3.6.1.6.2
		3.6.3 updelement wildcard	3.6.3 Ändern einer Wildcard	3.6.3.1 Ändern der Namensräume			ja/nein	ja, analog zu 1.6.3.2
				3.6.3.2 Ändern der Validierung			ja/nein	ja, analog zu 1.6.3.2
				3.6.3.6 Erhöhen des maximalen Vorkommens			nein	ja, analog zu 1.6.3.2
	3.8 updconstraint		3.8 Ändern einer Constraintdeklaration	3.8.1 Umbenennen der Schlüssel-/Identitätsdeklaration			nein	ja, anbieterKey in anbieternameKey
				3.8.4 Ändern des Constraint-pfades	3.8.4.1 Ändern des selector-Pfades		ja/nein	ja, anbieterKey
					3.8.4.2 Ändern des field-Pfades		ja/nein	ja, analog zu 3.8.4.1
	3.9 updschema		3.9 Ändern einer Schemadeklaration	3.9.1 Ändern des Ziel-Namensraumes			nein	ja, analog zu 1.6.3.2
				3.9.7 Ändern der Abgeschlossenheit			ja/nein	ja, analog zu 3.5.3

Abbildung 5.11: ELaX-Operation: update für Überarbeitung der Anbieterdaten abgeleitet aus B

Das liegt darin begründet, dass sie ebenfalls den Namen bzw. das ref-Attribut ändern und somit im letzteren Fall eine Instanzanpassung hervorrufen können.
Im bisher vorgestelltem Szenario wurde eine Element-Wildcard in Operation *1.6.3.2* eingefügt, durch die beliebige Elemente aus dem Schema selbst oder aus einem externen Schemata eingebunden werden können. Ebenfalls aus externen Schemata können beliebige Attribute eingebunden werden, wodurch die Operation *3.2.1.4* analog zu *1.6.3.2* ist. Weiterhin wurde durch das Einfügen eines bestimmten Namespace bei der Einfügeoperation *1.6.3.2* sozusagen der Namensraum von #all auf „http://www.ausstattung.org“ geändert, sodass die Operationen *3.2.5.1*, *3.6.3.1* und *3.9.1* äquivalent zum Hinzufügen der Wildcard sind. Außerdem ist die Validierung des any-Elementes standardmäßig auf „strict“ eingestellt, sodass diese beim Auftauchen im im Schema auch gegen das entsprechende XML-Schema validiert werden. Die beiden anderen Validierungsarten „lax“ und „skip“ können auf die Validierung teilweise bzw. komplett verzichten, wenn das entsprechende XML-Schema fehlt. Dadurch sind diese Arten eine Lockerung der Validierungsart „strict“, weshalb die Operationen *3.2.5.2* und *3.6.3.2* analog zu *1.6.3.2* sind. Weiterhin ist die Wildcard mit einem maximalen Vorkommen von „unbounded“ eingefügt worden und hat somit den Standardwert von 1 erhöht, wodurch die Operation *3.6.3.6* analog zu dieser Änderung ist.
Das *Ändern des Inhaltstyps auf (3.5.2)* mixed = „true“ in Operation *3.5.2.1* bedeutet, dass zusätzlich zu definierten Elementen Fließtext als Elementinhalt vorkommen kann. Dabei ist es möglich, dass nur die definierten Elemente vorkommen dürfen, weshalb die Operation *3.5.2.2* analog zu *3.5.2.1* ist.
Die Erweiterung des „nameType“ von *Erweiterung: simpleContent->Erweiterung:complexContent (3.5.4.9)* kann ohne Einschränkungen erfolgen. Ebenfalls ohne Einschränkungen können die Inhaltstypen der Operationen *3.5.4.6-3.5.4.8* und *3.5.4.11* geändert werden, weshalb sie analog zu *3.5.4.9* sind. Außerdem können einige Typdefinitionen nur dann ineinander überführt werden, wenn sie zum Einen keine Facetten (siehe Operationen *3.5.4.1-3.5.4.3*) oder zum Anderen keine Inhaltsmodelle enthalten (siehe Operationen *3.5.4.4*, *3.5.4.5*, *3.5.4.10*, *3.5.4.12*). In der Operation *3.5.4.4* beispielsweise dürfen im complexContent keine Sequenzen, Mengen oder Alternativen eingeschränkt worden sein.

Wenn diese Operationen die genannten Vorbedingungen erfüllen, sind sie äquivalent zu *3.5.4.9*.
Damit der „anbieterType“ im Szenario erweitert werden konnte, wurde dessen Abgeschlossenheit geändert, d.h. von einem vollständigen Einschränkungs- und Erweiterungsverbot (#all) zu „restriction“ wurde das Erweiterungsverbot „extension“ aufgehoben. Diese Semantik lässt sich ebenfalls auf die Operationen *3.6.1.5* und *3.9.7* übertragen, sodass diese analog zu *3.5.3* angesehen werden können.
Das *Ändern der Nullwertfähigkeit (3.6.1.6)* bei Elementdeklarationen bedeutet im Falle des Wahrheitswertes „true“, dass wenn der Elementinhalt leer ist, dann eine explizite Angabe in der Instanz erfolgen muss. Im Beispiel wurde die Operation *3.6.1.6.2* umgesetzt, d.h. das nillable-Attribut mit Wert „true“ wurde eingefügt. Aufgrund der Tatsache, dass der Default-Wert vom nillable-Attribut „false“ ist, käme das Einfügen einer Änderung einer expliziten Angabe des nillable-Attributes mit Wert „false“ gleich, weshalb die Operation *3.6.1.6.4* analog zu *3.6.1.6.2* ist. Die Operationen *3.6.1.6.1* und *3.6.1.6.5* führen entweder ein Hinzufügen oder Löschen des nillable-Attributs mit Wert „false“ durch was keine Auswirkungen auf die Instanz hat und somit zu Operation *3.6.1.6.2* in der Hinsicht analog sind, dass das nillable-Attribut hinzugefügt bzw. gelöscht wird. Eine weitere Umkehrung stellt die Änderung von „true“ nach „false“ in den Operationen *3.6.1.6.3* und *3.6.1.6.6* dar, weil hier die Elemente mit leerem Inhalt nun nicht mehr explizit gekennzeichnet werden, sodass diese im Falle des Vorkommens gelöscht werden müssen. Die Analogie der beiden zuletzt genannten Operationen zu *3.6.1.6.2* besteht darin, dass sie beide Instanzanpassungen auslösen aufgrund der Änderung des nillable-Attributs.
Das *Ändern des selector-Pfades (3.8.4.1)* im „anbieterKey“ hat zur Folge, dass der zuvor nicht mehr existente XPath-Pfad in einen transformiert wird, der wieder im XML-Schema zu finden ist. Ähnlich lässt sich der Field-Pfad ändern, indem angegeben wird, welches Element bzw. Attribut ausgewählt wird, ausgehend vom Selector-Pfad. Damit ist die Operation *3.8.4.2* analog zu *3.8.4.1* ausführbar.

5.2.4 Formale Anpassungen

Die Verantwortlichen haben nach den grundlegenden strukturellen Änderungen in den vorherigen Abschnitten einige formale Anforderungswünsche aufgestellt. Als Erstes sollen die bisher fehlenden Start- und Ziel-Attribute für die Zeit-Elemente eingefügt werden, die sich unter den Hinfahrt- und Rückfahrt-Elementen befinden. Der Grund dafür ist, dass auch die Haltezeiten bei Zwischenstationen in Zukunft mitgespeichert werden sollen. Bei der initialen Transformation wird vorgeben, dass bei Hinfahrt die Von- und Nach-Werte und bei der Rückfahrt die Nach- und Von-Werte als Start- und Ziel-Attribut verwendet werden. Außerdem wird gleichzeitig noch die Problematik der Zeitangabe geklärt, d.h. die parallele Verwendung zweier Zeitangaben führte letztendlich zu mehr Verwirrung, weshalb sich die Verantwortlichen auf die Angabe als xs:time entschieden und somit die Elemente „ab“ und „an“. Bei der Transformation werden dann die nicht mehr erlaubten Zeitangaben in den built-in Typ xs:time umgewandelt, z.B. „stunde“ mit Wert „10“ und „minute“ mit Wert „25“ wird zu „10:25:00+01:00“. Durch den Zusatz +01:00 wird angenommen, dass sich die Fernbuslinien nur in einer Zeitzone befinden.
Für die Umsetzung wird zunächst das bisher über include eingebundene XML-Schema „zeit.xsd“ über override hinzugefügt, um die angesprochenen Änderungen durchzuführen (siehe Operation *3.7.2.9*). Anschließend wird die komplexe Typdefinition „zeitType“ als Sequenz hinzugefügt, in der die Elementreferenzen „ab“ und „an“ eingefügt werden (siehe Operationen *1.3.1.2* und *1.6.2.1.2*). Als nächstes wird noch das Rythmus-Attribut übernommen, sowie für die Start- und Ziel-Attribute die Attributgruppenreferenz „tour“ eingefügt (siehe Operationen *1.2.3.2* und *1.2.4*).
Nachdem diese Änderungen vollzogen wurden, steht die Schemaanpassung kurz vor dem Ende. Die Verantwortlichen sehen eine Änderung der Version des Schemas von 1.0 auf 1.1 vor, um zu verdeutlichen, dass neue Schemakonzepte wie „xs:override“ verwendet wurden (siehe Operation *3.9.4*). Außerdem werden der Kommentar und das Attribut „last_schema _change“ auf das Jahr 2013 geändert, um die Aktualität bzw. den Zeitpunkt der Schemaevolution abzubilden (siehe Operationen *3.1.1* und *3.2.2.3*).

ELaX-Operation: add

Die zuletzt entwickelten Einfügeoperationen lassen sich in die Kategorien *addannotation (1.1)*, *addattributegroup (1.2)*, *addgroup (1.3)*, *addelement (1.6)* und *addmodule (1.7)* einteilen, dargestellt in Abbildung 5.12.

ELaX-Operation			Schemaevolutionsschritt			Instanzanpassung	Umsetzung im Beispiel
1. add	1.1 addannotation		1.1 Hinzufügen einer Anmerkung	1.1.1 Hinzufügen einer documentation		nein	ja, analog zu 3.1.1
				1.1.2 Hinzufügen einer appinfo		nein	ja, analog zu 3.1.1
	1.2 addattributegroup	1.2.3 addattributeref	1.2.3 Hinzufügen einer Attributdeklaration Referenz	1.2.3.2 "optional"		nein	ja, rythmus (zeitType)
		1.2.4 addattributegroupref	1.2.4 Hinzufügen einer Attributgruppen Definition Referenz			ja/nein	ja, tour (routenType)
	1.3 addgroup		1.3 Hinzufügen einer Gruppendefinition	1.3.1 Sequenz	1.3.1.2 minOccurs > 0 maxOccurs > 0	nein	ja, zeitType
	1.6 addelement	1.6.2 addelementref	1.6.2 Hinzufügen einer Elementdeklaration Referenz in	1.6.2.1 Sequenz	1.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja	ja, ab und an (zeitType)
	1.7 addmodule		1.7 Hinzufügen einer Moduldeklaration	1.7.1 import		nein	ja, analog zu 3.7.2.9
				1.7.2 redefine		nein	ja, analog zu 3.7.2.9
				1.7.3 include		nein	ja, analog zu 3.7.2.9
				1.7.4 override		nein	ja, analog zu 3.7.2.9

Abbildung 5.12: ELaX-Operation: add für formale Anpassungen abgeleitet aus B

Das *Ändern einer Anmerkung (3.1)* wird im Documentation-Element vorgenommen (siehe Operation *3.1.1*), d.h. der Kommentar wird an einer bestimmten Stelle verändert. Das gleiche Ergebnis könnte erreicht werden, indem zunächst der alte Kommentar gelöscht und ein neuer mit der Änderung eingefügt wird. Angenommen die Löschung ist bereits erfolgt, dann ist das *Hinzufügen einer Anmerkung (1.1)* (und somit auch die Operationen *1.1.1* und *1.1.2*) äquivalent zu *3.1.1*.
Die Typänderung von Modulen, z.B. von include zu override für das XML-Schema „zeit.xsd“ (siehe Operation *3.7.2.9*), kann auch dahingehend umgestaltet werden, dass zunächst das alte Modul gelöscht und das neue eingefügt wird. Im letzteren Fall wären die Operationen *1.7.1*-*1.7.4* äquivalent zu *3.7.2.9*.

ELaX-Operation: delete

Basierend auf den zuvor entwickelten Änderungsoperationen lassen sich Löschoperationen ableiten, die in die Kategorien *delannotation (2.1)* und *delmodule (2.7)* eingeordnet werden können, dargestellt in Abbildung 5.13.

ELaX-Operation		Schemaevolutionsschritt		Instanzanpassung	Umsetzung im Beispiel
2. delete	2.1 delannotation	2.1 Entfernen einer Anmerkung	2.1.1 Entfernen einer documentation	nein	ja, analog zu 3.1.1
			2.1.2 Entfernen einer appinfo	nein	ja, analog zu 3.1.1
	2.7 delmodule	2.7 Entfernen einer Moduldeklaration	2.7.1 import	ja/nein	ja, analog zu 3.7.2.9
			2.7.2 redefine	ja/nein	ja, analog zu 3.7.2.9
			2.7.3 include	ja/nein	ja, analog zu 3.7.2.9
			2.7.4 override	ja/nein	ja, analog zu 3.7.2.9

Abbildung 5.13: ELaX-Operation: delete für formale Anpassungen abgeleitet aus B

Wenn eine Annotation geändert wird kann dies zum Einen direkt geschehen (siehe Operation *3.1.1*) oder zum Anderen über den Umweg des Löschens der alten und des Einfügens der neuen Annotation. Dabei führen die Operationen *2.1.1* und *2.1.2* die Löschung der Annotationen durch und bereiten somit das Einfügen vor. Dies ist analog zu Operation *3.1.1*.
Die prinzipiell gleiche Vorgehensweise kann angewandt werden, wenn ein Modultyp geändert werden soll. Dazu wird erst das alte Modul gelöscht und anschließend ein neues eingefügt, welches die verwendbaren Informationen des Ausgangsmodul berücksichtigt. Damit sind die Löschoperationen für die Module *2.7.1*-*2.7.4* äquivalent zur direkten Änderung in Operation *3.7.2.9*.

ELaX-Operation: update

Die zuletzt entwickelten Änderungsoperationen können in die Kategorien *updannotation (3.1)*, *updattributegroup (3.2)*, *updgroup (3.3)*, *updelement (3.6)*, *updmodule (3.7)* und *updschema (3.9)* einsortiert werden, abgebildet in Abbildung 5.14.

ELaX-Operation			Schemaevolutionsschritt			Instanzanpassung	Umsetzung im Beispiel
3. update	3.1 updannotation		3.1 Ändern einer Anmerkung	3.1.1 Ändern einer documentation		nein	ja, von 2012 auf 2013
				3.1.2 Ändern einer appinfo		nein	ja, analog zu 3.1.1
	3.2 updattributegroup	3.2.2 updattribute	3.2.2 Ändern einer Attributdeklaration	3.2.2.3 Ändern des fixed-Wertes		ja	ja, last_schema_change
				3.2.2.4 Ändern des default-Wertes		nein	ja, analog zu 3.2.2.3
		3.2.3 updattributeref	3.2.3 Ändern einer Attributdeklaration Referenz	3.2.3.2 Ändern des fixed-Wertes		ja	ja, analog zu 3.2.2.3
				3.2.3.3 Ändern des default-Wertes		nein	ja, analog zu 3.2.2.3
	3.3 updgroup		3.3 Ändern einer Gruppendefinition	3.3.1 Ändern des Typs	3.3.1.2 Sequenz -> Alternative	ja/nein	ja, analog zu 3.3.1.4
					3.3.1.3 Alternative->Menge	ja/nein	ja, analog zu 3.3.1.4
					3.3.1.4 Alternative->Sequenz	ja/nein	ja, zeitType und zeit.xsd
					3.3.1.6 Menge->Alternative	ja/nein	ja, analog zu 3.3.1.4
	3.6 updelement	3.6.1 updelementdef	3.6.1 Ändern einer Elementdeklaration	3.6.1.3 Ändern des fixed-Wertes		ja	ja, analog zu 3.2.2.3
				3.6.1.4 Ändern des default-Wertes		nein	ja, analog zu 3.2.2.3
	3.7 updmodule		3.7 Ändern einer Moduldeklaration	3.7.2 Ändern des Typs	3.7.2.1 import -> redefine	nein	ja, analog zu 3.7.2.9
					3.7.2.2 import -> include	nein	ja, analog zu 3.7.2.9
					3.7.2.3 import -> override	nein	ja, analog zu 3.7.2.9
					3.7.2.4 redefine -> import	nein	ja, analog zu 3.7.2.9
					3.7.2.5 redefine -> include	nein	ja, analog zu 3.7.2.9
					3.7.2.6 redefine -> override	nein	ja, analog zu 3.7.2.9
					3.7.2.7 include -> import	nein	ja, analog zu 3.7.2.9
					3.7.2.8 include -> redefine	nein	ja, analog zu 3.7.2.9
					3.7.2.9 include -> override	nein	ja, zeit.xsd
					3.7.2.10 override -> import	nein	ja, analog zu 3.7.2.9
					3.7.2.11 override -> include	nein	ja, analog zu 3.7.2.9
					3.7.2.12 override -> redefine	nein	ja, analog zu 3.7.2.9
	3.9 updschema		3.9 Ändern einer Schemadeklaration	3.9.2 Ändern des Ziel-Namensraumenspräfix		nein	ja, analog zu 3.9.4
				3.9.3 Ändern der Sprache (language)		nein	ja, analog zu 3.9.4
				3.9.4 Ändern der Version		nein	ja, von 1.0 auf 1.1
				3.9.5 Ändern der Elementform	3.9.5.1 unqualified -> qualified	nein	ja, analog zu 3.9.4
					3.9.5.2 qualified -> unqualified	nein	ja, analog zu 3.9.4
				3.9.6 Ändern der Attributform	3.9.6.1 unqualified -> qualified	nein	ja, analog zu 3.9.4
					3.9.6.2 qualified -> unqualified	nein	ja, analog zu 3.9.4

Abbildung 5.14: ELaX-Operation: update für formale Anpassungen abgeleitet aus B

Das *Ändern einer appinfo (3.1.2)* ist analog zum *Ändern einer documentation (3.1.1)* mit dem Unterschied, dass die appinfo Informationen für die Applikation beinhaltet, während die documentation zum besseren Verständnis eines Schemas beim Lesen durch einen Menschen dient.
Die Änderung des Fixed-Wertes, z.B. bei der Attributdeklaration last_schema_change, kann nur im Schema selbst vorgenommen werden, sodass in der Instanz dieser Wert immer verwendet werden muss. Im Gegensatz dazu beschreibt das Default-Attribut einen Wert, der dann angegeben wird, wenn das Attribut nicht weiter spezifiziert wurde. Somit wird der Default-Wert nur optional angegeben, weshalb er eine Abschwächung des Fixed-Wertes und deshalb die Operation *3.2.2.4* analog zu *3.2.2.3* ist. Ebenfalls sind fixed- und default-Attribute bei Attributreferenzen und Elementdeklarationen erlaubt, welche dieselben Eigenschaften wie bei einer Attributdeklarationen haben. Deshalb sind die Operationen *3.2.3.2*, *3.2.3.3*, *3.6.1.3* und *3.6.1.4* äquivalent zu *3.2.2.3*.
Die Überschreibung der komplexen Typdefinition „zeitType“ aus dem eingebundenen XML-Schema „zeit.xsd“ hat zur Folge, dass die zuvor verwendete Alternative zwischen verschiedenen Zeitangaben zu einer Sequenz einer bestimmten Zeitangabe geändert worden ist (siehe Operation *3.3.1.4*). Wenn mehrere Alternativoptionen in der Instanz vorkommen und die Umwandlung in eine Sequenz nur noch eine Alternativauswahl unterstützt, dann müssen diejenigen Schemakomponenten entfernt werden, die zur nicht mehr unterstützen choice-Option gehören. Ähnlich ist die Umformung einer Alternative zu einer Menge, weil hier die nicht mehr unterstützten Alternativen aus der Instanz gelöscht werden müssen, weshalb die Operation *3.3.1.3* äquivalent zu *3.3.1.4* ist. Bei der Umkehrung der Transformation, d.h. von Sequenz bzw. Menge zur Alternative, findet dann eine Instanzanpassung statt, wenn mehr als ein Element vorkommt.

Aus der Menge dieser Elemente darf nur eins ausgewählt werden, weshalb andere in der Instanz vorkommende Elemente womöglich gelöscht werden müssten, je nach festgelegtem Vorkommen. Damit können die Operationen *3.3.1.2* und *3.3.1.6* analog zu *3.3.1.4* ausgeführt werden.
Das *Ändern des Typs (3.7.2)* einer Moduldeklaration erfolgt zwischen den Modulen import, redefine, include und override, z.B. von include zu override in Operation *3.7.2.9*. Diese Umformung erlaubt, dass einige Teile des komplett eingebundenen XML-Schemas „zeit.xsd“ überschrieben werden können, weshalb die Operation *3.7.2.3* mit *3.7.2.9* verglichen werden kann. Dabei bietet override im Gegensatz zu redefine mehr Anpassungsmöglichkeiten, sodass redefine eine Teilmenge von override abdeckt, sodass die Operationen *3.7.2.1*, *3.7.2.6*, *3.7.2.8* und *3.7.2.12* analog zu *3.7.2.9* sind. Angenommen die Änderungen von override des Schemas „zeit.xsd“ befinden sich in einem anderen Namensraum, dann könnte die Einbindung dieses XML-Schemas über import dieselben Auswirkungen hervorrufen, sodass die Operation *3.7.2.7* äquivalent zu *3.7.2.9* ist. Dieselbe Argumentation gilt für die Umkehrungen *3.7.2.2*, *3.7.2.4*, *3.7.2.5*, *3.7.2.10* und *3.7.2.11*, sodass diese ebenfalls analog zu *3.7.2.9* sind.
Das *Ändern der Version (3.9.4)* im Wurzelelement eines jeden XML-Schemas hat die Bedeutung, dass beispielsweise XML-Validatoren aktuelle Konzepte von XML-Schemata zur Validierung mit einbeziehen. Dabei wird lediglich ein Attributwert geändert, wodurch dieser Vorgang vergleichbar ist mit den Operationen *3.9.2*, *3.9.3*, *3.9.5.1*, *3.9.5.2*, *3.9.6.1* und *3.9.6.2*, mit dem Unterschied, dass die Operation *3.9.2* nicht den Attributwert selbst, sondern den Attributnamen anpasst.

5.3 Erwartetes Ergebnis der XML-Schema Evolution

Auf Basis der aufgestellten und abgeglichenen Schemaevolutionsschritte in Abschnitt 5.2 werden die Auswirkungen auf das XML-Schema und XML-Dokument aus Abschnitt 5.1 zusammengefasst. In Abbildung 5.15 ist der Aufbau des evolutionierten XML-Schemas dargestellt.

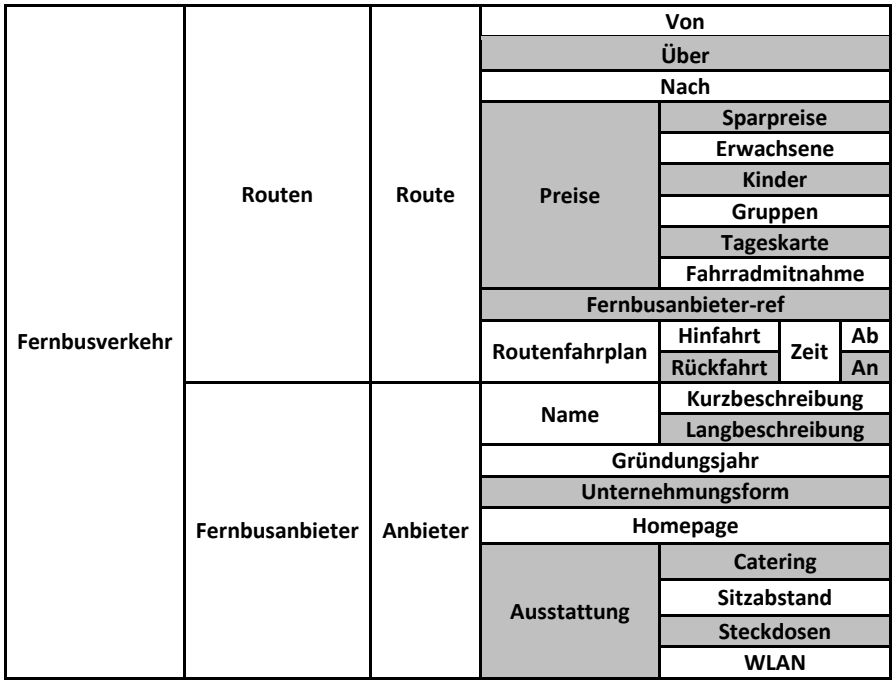

Abbildung 5.15: Aufbau des evolutionierten XML-Schemas abgeleitet aus E.3, E.5 und E.6

Im Vergleich zum Ausgangsschema in Abbildung 5.1 ist das Fernbuslinien-Element im evolutionierten Schema durch die Elemente Routen und Fernbusanbieter ersetzt worden. Die Routen-Elemente bündeln alle Route-Elemente, die dieselbe Strecke fahren, d.h. Start und Ziel der Fernbuslinie stimmt überein. Dabei wurde unter dem Route-Element eine Fernbusanbieter-ref eingefügt, die einen bestimmten Anbieter unter dem Fernbusanbieter referenziert. Die Informationen zu Hinfahrt- und Rückfahrt-Elemente sind in der Hierarchie unter dem neu eingefügten Routenfahrplan-Element gewandert. Dabei wurden die Zeitangaben auf die im Quellschema spezifizierte Alternative 2 reduziert.
Weiterhin sind die Anbieterdaten unter das Fernbusanbieter-Element eingefügt worden, wobei mehrere Anbieter auftauchen können. Die Beschreibung zu einem Fernbusanbieter ist in den Namen desselbigen eingegliedert worden, sodass mindestens eine Kurzbeschreibung für jeden Anbieter angegeben wird. Außerdem werden die Ausstattung-Elemente im Anbieter integriert, um somit die Ausstattung der verwendeten Fernbusse des Anbieters abzubilden. Die vorgestellten Neuerungen sind zur besseren Veranschaulichung im Auszug der Zielinstanz in Abbildung 5.16 angegeben, wobei der gezeigte Ausschnitt inhaltlich mit der Quellinstanz aus Abbildung 5.2 übereinstimmt.

```
<?xml version="1.0" encoding="UTF-8"?>
<fernbusverkehr [...] last_schema_change="01.01.2013">
<routen start="Rostock" ziel="Berlin">
 <route routeID="rouRB1" buslinien="Joost-Ostsee-Express Wörlitz-Tourist">
  <von>Warnemünde</von>
  <über>Rostock</über>
  <nach>Berlin</nach>
  <preise währung="EUR">
   <sparpreise>11.00</sparpreise>
   <erwachsene>21.00</erwachsene>[...]
  </preise>
  <fernbusanbieter-ref>a01</fernbusanbieter-ref>
  <routenfahrplan>
   <hinfahrt>
    <zeit rythmus="täglich" start="Warnemünde" ziel="Berlin">
     <ab>06:40:00+01:00</ab>
     <an>09:45:00+01:00</an>
    </zeit>[...]
   </hinfahrt>
   <rückfahrt>
    <zeit rythmus="täglich" start="Berlin" ziel="Warnemünde">
     <ab>07:00:00+01:00</ab>
     <an>10:00:00+01:00</an>
    </zeit>[...]
   </rückfahrt>
  </routenfahrplan>
 </route>[...]
</routen>[...]
<fernbusanbieter>
 <anbieter anbieterID="a01">
  <name akronym="MFB">MeinFernbus<kurzbeschreibung>Der 2011 in Berlin gegründete Verbund MeinFernbus will nach
       eigenen Angaben beliebtester und bekanntester Anbieter von Fernbuslinien werden.</kurzbeschreibung></name>
  <gründungsjahr>2011</gründungsjahr>
  <unternehmungsform>GmbH</unternehmungsform>
  <homepage>http://meinfernbus.de/</homepage>
  <ausstattung>
   <extra:catering>Snacks Kaffee </extra:catering>
   <extra:sitzabstand>groß</extra:sitzabstand>
   <extra:steckdosen>ja</extra:steckdosen>
   <extra:wlan>ja</extra:wlan>
  </ausstattung>
 </anbieter>[...]
</fernbusanbieter>
</fernbusverkehr>
```

Abbildung 5.16: Auszug aus Fernbuslinie_Evolution.xml abgeleitet aus E.4

Kapitel 6

Evaluierung des vorgestellten Ansatzes

Die Evaluierung des vorgestellten Ansatzes *CodeXMapper* in 4 erfolgt in zwei Phasen. In der ersten Phase werden die erzeugbaren ELaX-Ausdrücke des *CodeXMapper* anhand des Fernbusverkehr-Szenarios aus 5 in 6.1 Abschnitt bewertet. Die zweite Phase überprüft in Abschnitt 6.2 die erzeugten ELaX-Ausdrücke des *CodeXMapper* mit denen in CodeX generierten, auf Basis der Anforderungen aus dem Fernbusverkehr-Szenario. Hierbei werden ebenfalls Kombinationsmöglichkeiten der beiden Tools vorgestellt. Eine Übersicht der Evaluationsergebnisse befindet sich im Anhang D.

6.1 Bewertung anhand eines selbstgewählten Beispiels

Aufbauend auf dem in Kapitel 5 beschriebenen Fernbusverkehr-Szenario, werden die entsprechenden Quell- und Zielschemata (siehe E.1 und E.3) in das stand-alone Tool *CodeXMapper* geladen. Anschließend werden nach Durchführung des Matchingprozesses die ELaX-Ausdrücke für die Einfüge-, Lösch- und Aktualisierungsoperationen zur Durchführung der Schemaevolution erzeugt. Dabei produziert der *CodeXMapper* die built-in-Typen „xs:IDREF“ „xs:ID“ und „xs:anyType“, die für CodeX relevant, aber für die Bewertung anhand des Szenarios außen vor gelassen werden können, sodass die in den Abbildungen 6.1, 6.2 und 6.3 dargestellten Änderungsoperationen näher betrachtet werden.
Zunächst wird in Operation *1.2.1* das Hinzufügen der Attributgruppe „tour“ richtig umgesetzt, wobei gleichzeitig die Attributreferenzen „start“ und „ziel“ aus Operation *1.2.3.3* ebenfalls eingefügt werden. Dabei stellt der *CodeXMapper* sicher, dass die entsprechend referenzierten Attribute vorher eingefügt worden sind (siehe Operation *1.2.2*). Die beiden Attributgruppenreferenzen „tour“ in Operation *1.2.4* werden somit erst eingefügt, nachdem die Attributgruppe hinzugefügt wurde. Da das Mapping zwischen dem Abkürzung- und Akronym-Attribut nicht automatisch generiert wurde, wird zunächst eine Einfügeoperation für das Attribut „akronym“ in Operation *1.2.2* ausgeführt und anschließend das Abkürzung-Attribut aus dem Quellschema gelöscht (siehe Operation *2.2.2*). Dadurch wird zwar semantisch die angestrebte Update-Operation *3.2.2.1* umgesetzt, jedoch die syntaktische Umsetzung einer Aktualisierng des Attributnamens schlug fehl. Analog dazu wurde die entsprechende Referenzumbenennung der beiden Attribute „abkürzung“ und „akronym“ nicht erkannt (siehe Operation *3.2.3.1*), weshalb die Einfügeoperation *1.2.3.3* und die Löschoperation *2.2.3.3* durchgeführt werden. Daneben wurden die restlichen Attributreferenzen „rythmus“ und „anbieterID“ ordnungsgemäß eingefügt (siehe Operationen *1.2.3.2* und *1.2.3.3*), wobei das referenzierte AnbieterID-Attribut vorher ebenfalls eingefügt wurde (siehe Operation *1.8.3*). Dabei verweist die Attributreferenz „rythmus“ auf das eingebundene XML-Schema „zeit.xsd“, weshalb keine globale Attributdeklaration eingefügt werden muss.
Das Hinzufügen von Modellgruppen erfordert das Vorhandensein der entsprechenden komplexen Typdefinitionen, wobei diese zuerst eingefügt werden müssen.

ELaX-Operation			Schemaevolutionsschritt			Umsetzung im Beispiel	Umsetzung im CodeXMapper
1. add	1.2 add attribute group	1.2.1 addattribute groupdef	1.2.1 Hinzufügen einer Attributgruppen Definition			ja, tour	**√**add attributegroup name 'tour' with attributeref 'start' use 'required' in '/node()/node()[@name="tour"]/.' attributeref 'ziel' use 'required' in '/node()/node()[@name="tour"]/.' ;
		1.2.2 addattribute	1.2.2 Hinzufügen einer Attributdeklaration			ja, start, ziel	**√**add attribute name 'start' type 'xs:string' ;
							√add attribute name 'ziel' type 'xs:string' ;
							add attribute name 'akronym' type 'xs:string' ;
		1.2.3 add attributeref	1.2.3 Hinzufügen einer Attribut-deklaration Referenz	1.2.3.2 "optional"		ja, rythmus (zeitType)	**√**add attributeref 'rythmus' use 'optional' in '/node()/node()/node()[@name="zeitType"]/.' ;
				1.2.3.3 "required"		ja, start und ziel (tour), anbieterID (anbieterneuType)	***√** start und ziel bereits in 1.2.1 eingefügt*
							√ add attributeref 'anbieterID' use 'required' in '/node()/node()[@name="anbieterneuType"]/node()/.' ;
							add attributeref 'akronym' use 'required' in '/node()/node()[@name="nameType"]/node()/.' ;
		1.2.4 addattribute groupref	1.2.4 Hinzufügen einer Attributgruppen Definition Referenz			ja, tour (zeitType, routenType)	**√**add attributegroupref 'tour'in'/node()/node()[@name="routenType"]/.' ;
							√add attributegroupref 'tour' in '/node()/node()/node()[@name="zeitType"]/.' ;
	1.3 addgroup		1.3 Hinzufügen einer Gruppen-definition	1.3.1 Sequenz	1.3.1.2 minOccurs > 0 maxOccurs > 0	ja, zeitType, routenfahrplanType, fernbusanbieterType	**(√)***zeitType wird beim Einfügen in 1.7.4 erledigt*
							√add group mode sequence in '/node()/node()/node()[@name="zeitType"]/.' ;
							√add complextype name 'routenfahrplanType' ;
							√add group mode sequence in '/node()/node()[@name="routenfahrplanType"]/.' ;
							√add complextype name 'fernbusanbieterType' ;
							√add group mode sequence in '/node()/node()[@name="fernbusanbieterType"]/.' ;
							add group mode sequence in '/node()/node()[@name="anbieterneuType"]/node()/.' ;
							add complextype name 'routenType' ;
							add group mode sequence in '/node()/node()[@name="routenType"]/.' ;
							add complextype name 'ausstattungType' ;
							add group mode sequence in '/node()/node()[@name="ausstattungType"]/.' ;
	1.4 addst		1.4 Hinzufügen einer einfachen Typdefinition in Hierarchie als	1.4.1 Einschränkung		ja, gründungsjahrType, fehlType	**√**add simpletype name 'gründungsjahrType' mode restriction of 'xs:gYear' with mininclusive '1900' ;
							Xadd fehlType
				1.4.3 Vereinigung		ja, geldneuType	**√**add simpletype name 'geldneuType' mode union 'geldType' 'fehlType' ;
	1.5 addct		1.5 Hinzufügen einer komplexen Typdefinition in Hierarchie als	1.5.4 Erweiterung: complexContent		ja, anbieterneuType	**√**add complextype name 'anbieterneuType' mode extension_cc with base 'anbieterType' ;
							mit add group benötigte Sequenz in 1.3.1.2 eingefügt
							add complextype name 'nameType' mode extension_cc with base 'beschreibungType' ;
	1.6 add element	1.6.1 add elementdef	1.6.1 Hinzufügen einer Elementdeklaration			ja, routen, fernbusanbieter	**√**add element name 'routen' type 'routenType' ;
							√add element name 'fernbusanbieter' type 'fernbusanbieterType' **;**
							X*add element name 'name' type 'nameType' ;*
							add element name 'routenfahrplan' type 'routenfahrplanType' ;
							add element name 'ausstattung' type 'ausstattungType' nillable 'true' ;
		1.6.2 add elementref	1.6.2 Hinzufügen einer Element-deklaration Referenz in	1.6.2.1 Sequenz	1.6.2.1.1 minOccurs = 0 maxOccurs ≥ 0	ja, hinfahrt und rückfahrt (routenfahrplanType)	**(√)***mit update elementref 'rückfahrt' in 3.6.2.1 erledigt*
							(√)*mit update elementref 'hinfahrt' in 3.6.2.1 eledigt*
					1.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja, fernbusanbieter (fernbusverkehrType), anbieter(fernbusanbieterType), ausstattung (anbieterneuType), ab und an (zeitType)	**(√)***mit update fernbuslinien in 3.6.2.1 erledigt*
							add elementref 'routen' maxoccurs 'unbounded' in '/node()/node()[@name="fernbusverkehrType"]/node()/.' ;
							(√)*anbieter wird in neue Position verschoben in 3.6.2.5 erledigt*
							√add elementref 'ausstattung' in '/node()/node()[@name="anbieterneuType"]/node()/node()/.' ;
							√add elementref 'ab' in '/node()/node()/node()[@name="zeitType"]/node()/.' ;
							√add elementref 'an' in '/node()/node()/node()[@name="zeitType"]/node()/.' ;
							add elementref 'fernbusanbieter-ref' in '/node()/node()[@name="routeType"]/node()/.' ;
							add elementref 'routenfahrplan' in '/node()/node()[@name="routeType"]/node()/.' ;
		1.6.3 addelement wildcard	1.6.3 Hinzufügen einer Wildcard	1.6.3.2 minOccurs > 0 maxOccurs > 0		ja, any (ausstattungType)	**√**add any namespace 'http://www.ausstattung.org' maxoccurs 'unbounded' in '/node()/node()[@name="ausstattungType"]/node()/.' ;
	1.7 addmodule		1.7Hinzufügen einer Moduldeklaration	1.7.4 override		ja, analog zu 3.7.2.9	*add module from 'zeit.xsd' mode override complextype name 'zeitType' ;*
	1.8 addconstraint		1.8 Hinzufügen einer Constraint-deklaration	1.8.1Hinzufügen eines Primärschlüssels	1.8.1.1 key	ja, routenKey (fernbusverkehr)	**√**add constraint name 'routenKey' type 'key' with selector './routen' field '@start' field '@ziel' in '/node()/node()[@name="fernbusverkehr"]/.' ;
				1.8.2 Hinzufügen eines Fremdschlüssels (keyref)		ja, startzielKey und zielstartKey	**√**add constraint name 'startzielKey' type 'keyref' refer 'routeKey' with selector './/hin fahrt/zeit' field '@start' field '@ziel' in '/node()/node()[@name="fernbusverkehr"]/.' ;
							√add constraint name 'zielstartKey' type 'keyref' refer 'routeKey' with selector './/rück fahrt/zeit' field '@ziel' field '@start' in '/node()/node()[@name="fernbusverkehr"]/.' ;
				1.8.3 Hinzufügen einer Identität		ja, anbieterID	**√**add attribute name 'anbieterID' type 'xs:ID' ;
				1.8.4 Hinzufügen einer Identitätsreferenz		ja, fernbusanbieter-ref	**√**add element name 'fernbusanbieter-ref' type 'xs:IDREF' ;
√ = umgesetzt, (√) = teilweise oder andersweitig umgesetzt, X = nicht umgesetzt							

Abbildung 6.1: Abgleich der ELaX-Add-Operationen abgeleitet aus D

Dabei erzeugt der *CodeXMapper* für die benötigten komplexen Typdefinitionen „zeitType“, „routenfahrplanType“ und „fernbusanbieterType“ die korrekten Ausdrücke nebst den Modellgruppen (siehe Operationen *1.3.1.2*). Dabei wird der „zeitType“ in Operation *1.7.4* durch das Hinzufügen einer Moduldeklaration erzeugt. Weiterhin wird durch das korrekte Einfügen der komplexen Typdefinition „anbieterneuType“ in Operation *1.5.4* die benötigte Sequenz-Gruppendefinition dem Quellschema hinzugefügt (siehe Operation *1.3.1.2*). Aufgrund der nicht erkannten Verbindung zwischen den komplexen Typdefinitionen „fahrplanType“ und „routenType“ kann die Umbenennung in Operation *3.5.1* nicht erfolgen, weshalb die komplexe Typdefinition „routenType“ mit Sequenz-Gruppendefiniton in *1.3.1.2* eingefügt und der „fahrplanType“-Typ mit Modellgruppe in *2.3.1.2* gelöscht werden. Dadurch fehlt jedoch die im Quellschema zuvor unter der Sequenz vorhandenen Route-Elementrefenz im Zielschema, sodass vor dem Löschen eine Update-Operation erzeugt wird, basierend auf dem gefundenen Mapping zwischen den Route-Elementreferenzen (siehe Operation *3.6.2.1*). Diese Aktualisierung verschiebt die Referenz von „fahrplanType“ in den neu eingefügten „routenType“. Des Weiteren wird die im Quellschema befindliche komplexe Typdefinition „ausstattungType“ nicht mit der im Zielschema gemappt, weshalb zum Einen eine Löschoperation in *2.3.1.2* und anschließend die Einfügeoperation in *1.3.1.2* ausgelöst werden.
Das Hinzufügen der einfachen Typdefinitionen „gründungsjahrType“ und „geldneuType“ werden vom *CodeXMapper* erkannt und korrekt in ELaX ausgedrückt (siehe Operationen *1.4.1* und *1.4.3*). Aufgrund einer zusammengefassten Darstellung der Vereinigungsmenge (memberTypes) im *CodeXMapper* ist nur die erste Typdefinition der Vereinigungsmenge dargestellt, weshalb das Einfügen der einfachen Typdefinition „fehlType“ nicht ausgelöst werden kann (siehe Operation *1.4.1*). Weiterhin wird das Mapping zwischen dem komplexen Typdefinitionen „nameType“ im Quell- und Zielschema nicht erkannt, wodurch eine Inhaltsänderung in Operation *3.5.4.9* mit dem Löschen des alten „nameType“ in *2.3.1.2* und dem Einfügen des neuen „nameType“ in *1.5.4* ersetzt wird.
Beim Hinzufügen der Elementdeklarationen „routen“ und „fernbusanbieter“ in *1.6.1*, der Element-Wildcard in *1.6.3.2*, sowie der Constraintdeklarationen in *1.8.1.1*, *1.8.2* und *1.8.4* werden die Schemakomponenten mit den richtigen, von *CodeXMapper* erzeugten ELaX-Operationen hinzugefügt. Daneben wird das gleichnamige Name-Element in Quell- und Zielschema nicht erkannt, weshalb eine unnötige Löschoperation in *2.6.1* und Einfügeoperation in *1.6.1* stattfinden. Die Umbenennung der Elementdeklaration „fahrplan“ in „routenfahrplan“ (siehe Operation *3.6.1.1*) wird andersweitig umgesetzt, d.h. das alte Fahrplan-Element wird aus dem Quellschema entfernt (siehe Operation *2.6.1*) und das neue Routenfahrplan-Element wird in Operation *1.6.1* hinzugefügt. Dadurch wird auch gleichzeitig die Typänderung von „fahrplanType“ zu „routenfahrplanType“ vorgenommen (siehe Operation *3.6.1.2*). Um die Nullwertigkeit des Ausstattung-Elementes zu ändern, werden analog zur Operation *3.6.1.1* das Ausstattung-Element gelöscht und anschließend mit den Änderungen neu hinzugefügt (siehe Operationen *2.6.1* und *1.6.1*).
Das Hinzufügen der Elementdeklarationsreferenzen „ausstattung“, „ab“ und „an“ werden vom *CodeXMapper* gemäß dem Szenario erkannt und die entsprechenden Einfügeoperationen generiert (siehe Operation *1.6.2.1.2*). Die Elementreferenzen „hinfahrt“ und „rückkfahrt“ (siehe Operation *1.6.2.1.1*) werden vom *CodeXMapper* durch move-Operationen in *3.6.2.1* vorgenommen, indem die Elementreferenzen aus der komplexen Typdefinition „routeType“ in die Sequenz des Typs „routenfahrplanType“ verschoben werden. Dadurch können keine Update-Operationen mehr auf den Hinfahrt- und Rückfahrt-Elementreferenzen im „routeType“ durchgeführt werden (siehe Operation *3.6.2.1*), sodass die Elementreferenzen „routen“ und „fernbusanbieter-ref“ neu eingefügt werden müssen (siehe Operation *1.6.2.1.2*). Dadurch wird gleichzeitig das minimale Vorkommen in *3.6.2.3* erhöht. Analog dazu wird die Anbieter-Elementreferenz aus dem „fernbuslinieType“ in die neue Typdefinition „fernbusanbieterType“ verschoben (siehe Operation *3.6.2.5*), sodass die Einfügeoperation in *1.6.2.1.2* der Elementreferenz „anbieter“ hinfällig wird.
In der komplexen Typdefinition „fernbusverkehrType“ soll die Fernbuslinie-Elementreferenz im Szenario in „routen“ umbenannt werden (siehe Operation *3.6.2.1*). Dabei erkennt das Programm zwar nicht das Mapping zwischen „fernbuslinie“ und „routen“, aber das zwischen „fernbuslinie“ und „fernbusanbieter“, sodass eine entsprechende Umbenennung in *3.6.2.1* ausgelöst wird. Das hat zur Folge, dass die Routen-Referenz neu hinzugefügt werden muss (siehe Operation *1.6.2.1.2*). Außerdem wird durch die Umbenennung das Einfügen der Fernbusanbieter-Elementreferenz hinfällig (siehe Operation *1.6.2.1.2*).

Das *Ändern des Typs (3.7.2)* einer Moduldeklaration von inlcude zu override in *3.7.2.9* kann in ELaX mit Einschränkungen durchgeführt werden, sodass bei Vorkommen von Attribut(gruppen)- und Elementdeklarationen, sowie einfachen und komplexen Typdefintionen das alte Modul entfernt und das neue eingefügt werden muss (siehe Operationen *2.7.3* und *1.7.4*). Diese Einschränkung kommt daher, dass die soeben genannten Schemakomponenten nur global eingefügt werden können und nicht an einer bestimmten Position im Schema. Dies ist begründet mit der Verwendung des Modellierungsstils *Garden of Eden*.

ELaX-Operation			Schemaevolutionsschritt			Umsetzung im Beispiel	Umsetzung im CodeXMapper
2. delete	2.2 del attribute group	2.2.2 delattribute	2.2.2 Entfernen einer Attributdeklaration			ja, MwSt.	√delete attribute name 'MwSt.' ;
							delete attribute name 'abkürzung' ;
		2.2.3 delattributeref	2.2.3 Entfernen einer Attributdeklaration Referenz	2.2.3.3 "required"		ja, MwSt. (preiseType)	√delete attributeref 'MwSt.' at '/node()/node()[@name="preiseType"]/.' ;
							delete attributeref 'abkürzung' at '/node()/node()[@name="nameType"]/node()/.' ;
	2.3 delgroup		2.3 Entfernen einer Gruppendefinition	2.3.1 Sequenz	2.3.1.2 minOccurs > 0 maxOccurs > 0	ja, fernbuslinieType	√delete complextype name 'fernbuslinieType' ; *delete group at '/node()/node()[@name="fernbuslinieType"]/.' ;*
							delete complextype name 'nameType' ;
							delete complextype name 'ausstattungType' ; *delete group at '/node()/node()[@name="ausstattungType"]/.' ;*
							delete complextype name 'fahrplanType' ; *delete group at '/node()/node()[@name="fahrplanType"]/.' ;*
	2.4 delst		2.4 Entfernen einer einfachen Typdefinition aus Hierarchie	2.4.1 Einschränkung		ja, MwSt.Type, serviceType, sitzabstandType, wahlType	√delete simpletype name 'MwSt.Type' ;
							√delete simpletype name 'serviceType' ;
							√delete simpletype name 'sitzabstandType' ;
							√delete simpletype name 'wahlType' ;
				2.4.2 Liste		ja, cateringType	√delete simpletype name 'cateringType' ;
	2.6 del element	2.6.1 delelementdef	2.6.1 Entfernen einer Elementdeklaration			ja, fernbuslinie, catering, sitzabstand, steckdosen, wlan, beschreibung	√delete element name 'fernbuslinie' ;
							√delete element name 'catering' ;
							√delete element name 'sitzabstand' ;
							√delete element name 'steckdosen' ;
							√delete element name 'wlan' ;
							√delete element name 'beschreibung' ;
							X*delete element name 'name' ;*
							delete element name 'fahrplan' ;
							delete element name 'ausstattung' ;
		2.6.2 delelementref	2.6.2 Entfernen einer Elementdeklaration Referenz aus	2.6.2.1 Sequenz	2.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja, catering, sitzabstand, steckdosen, wlan (ausstattungType)	√delete elementref 'catering' at '/node()/node()[@name="ausstattungType"]/node()/.' ;
							√delete elementref 'sitzabstand' at '/node()/node()[@name="ausstattungType"]/node()/.' ;
							√delete elementref 'steckdosen' at '/node()/node()[@name="ausstattungType"]/node()/.' ;
							√delete elementref 'wlan' at '/node()/node()[@name="ausstattungType"]/node()/.' ;
							delete elementref 'ausstattung' at '/node()/node()[@name="fernbuslinieType"]/node()/.' ;
							delete elementref 'fahrplan' at '/node()/node()[@name="fernbuslinieType"]/node()/.' ;
							delete elementref 'beschreibung' at '/node()/node()[@name="fernbuslinieType"]/node()/.' ;
	2.7 delmodule		2.7 Entfernen einer Moduldeklaration	2.7.3 include		ja, analog zu 3.7.2.9	*delete module at '/node()/node()[1]' ;*
√ = umgesetzt, (√) = teilweise oder andersweitig umgesetzt, X = nicht umgesetzt							

Abbildung 6.2: Abgleich der ELaX-Delete-Operationen abgeleitet aus D

Weiterhin werden die Löschoperationen für die Attributreferenz „MwSt.“, sowie des dazugehörigen Attributs und des damit referenzierten einfachen Typdefinition korrekt gelöscht (siehe Operationen in *2.2.2*, *2.2.3.3* und *2.4.1*). Des Weiteren wird die komplexe Typdefinition „fernbuslinieType“ in *2.3.1.2* korrekt gelöscht, wobei der *CodeXMapper* zusätzlich durch erzeugte Mappings die Löschoperationen für die Sequenz-Gruppendefinition und der Elementreferenzen „fahrplan“, „ausstattung“ und „beschreibung“ erzeugt, die im Typ vorkommen (siehe Operationen *2.3.1.2* und *2.6.2.1.2*). Außerdem werden die nicht mehr benötigten Elementdeklarationen „fernbuslinie“, „catering“, „sitzabstand“, „steckdosen“, „wlan“ und „beschreibung“, sowie deren referenzierenden Elementreferenzen ordnungsgemäß gelöscht (siehe Operationen *2.6.1* und *2.6.2.1.2*). Dabei werden alle referenzierten Typdefinitionen der soeben aufgezählten Elemente gelöscht, außer der „beschreibungType“ (siehe Operationen *2.4.1* und *2.4.2*).
Die folgenden Updateoperationen *3.1.1*, *3.2.2.2* und *3.2.2.3* werden für die Annotation und Attributdeklarationen „routeID“ und „last_schema_change“, sowie für die „routeID“-Attributreferenz (siehe Operation *3.2.3.4.1*) vom *CodeXMapper* richtig erzeugt. Beim Ändern der Gruppendefinition im „preiseType“ bündelt der *CodeXMapper* zum Einen die Typänderung (siehe Operation *3.3.1.5*) und zum Anderen die Änderung der Vorkommen (siehe Operation *3.3.2* und *3.3.5*) in einem ELaX-Ausdruck. Ebenfalls werden die Aktualiserungsoperation *3.4.4* mit in die ELaX-Operation in *3.4.2.6* integriert, wobei analog Operation *3.8.1* in *3.8.4.1* eingebettet wurde. Ansonsten werden die restlichen Update-Operationen in *3.4.7*, *3.5.2.1*, *3.5.3*, *3.6.1.2*, *3.6.2.3*, *3.8.2.1* und *3.9.4* korrekt ausgeführt.
Werden alle vorgestellten und abgeglichenen ELaX-Ausdrücke auf das Quellschema angewandt, entsteht das in Abbildung 5.15 bereits dargestellte XML-Schema.

ELaX-Operation			Schemaevolutionsschritt			Umsetzung im Beispiel	Umsetzung im CodeXMapper
3. update	3.1 updannotation		3.1 Ändern einer Anmerkung	3.1.1 Ändern einer documentation		ja, von 2012 auf 2013	√update annotation at '/node()/.' change documentation 'Übersicht über die Fernbuslinien in Deutschland Stand: 2013' ;
	3.2 upd attribute group	3.2.2 upd attribute	3.2.2 Ändern einer Attribut-deklaration	3.2.2.1 Umbenennen einer Attributdeklaration		ja,abkürzung in akronym	**(√)***mit delete abkürzung in 2.2.2 und add akronym in 1.2.2 erledigt*
				3.2.2.2 Ändern der Typdefinition		ja, routeID	√update attribute name 'routeID' change type 'xs:ID' ;
				3.2.2.3 Ändern des fixed-Wertes		ja, last_schema_change	√update attribute name 'last_schema_change' change fixed '01.01.2013' ;
		3.2.3 upd attribute ref	3.2.3 Ändern einer Attributdeklaration Referenz	3.2.3.1 Ändern des Referenznamen		ja, abkürzung in akronym (nameType)	**(√)***mit delete abkürzung in 1.2.3.3 und add akronym in 2.2.3.3 erledigt*
				3.2.3.4Ändern der Attribut verwendung	3.2.3.4.1 "optional" -> "required"	ja, routeID (routeType)	√update attributeref 'routeID' at '/node()/node()[@name="routeType"]/.' change use 'required' ;
	3.3 updgroup		3.3 Ändern einer Gruppendefinition	3.3.1 Ändern des Typs	3.3.1.5 Menge -> Sequenz	ja, preiseType	√update group at '/node()/node()[@name="preiseType"]/.' change mode sequence minoccurs '0' maxoccurs 'unbounded' ;
				3.3.2 Verringern des minimalen Vorkommens		ja, preiseType	**(√)***mit 3.3.1.5 erledigt*
				3.3.5 Erhöhen des maximalen Vorkommens		ja, preiseType	**(√)***mit 3.3.1.5 erledigt*
	3.4 updst		3.4 Ändern einer einfachen Typdefinition	3.4.2 Ändern des Typs	3.4.2.6 Vereinigung -> Liste	ja, buslinienType	√update simpletype name 'buslinienType' change mode list 'buslinieType' ;
				3.4.4 Entfernen einer einfachen Typdefinition aus einer Vereinigung		ja, buslinienType	**(√)***mit 3.4.2.6 erledigt*
				3.4.7 Modifizieren einer Einschränkung		ja, buslinieType	√update simpletype name 'buslinieType' change mode restriction of 'xs:string' modify enumeration 'Joost-Ostsee-Express' at '/node()/node()[@name="buslinieType"]/node()/node()[1]' modify enumeration 'Wörlitz-Tourist' at '/node()/node()[@name="buslinieType"]/node()/node()[2]' modify enumeration 'Bayern-Express&P.Kühn' at '/node()/node()[@name="buslinieType"]/node()/node()[3]' modify enumeration 'Eurolines-Scandinavia' at '/node()/node()[@name="buslinieType"]/node()/node()[4]' ;
	3.5 updct		3.5 Ändern einer komplexen Typdefinition	3.5.1 Umbenennen der Typdefinition		ja, fahrplanType in routenType	**(√)***mit delete fahrplanType in 2.3.1.2 und add routenType in 1.3.1.2 erledigt*
				3.5.2Ändern des Inhalts- typs auf	3.5.2.1 mixed = "true"	ja, beschreibungType	√update complextype name 'beschreibungType' change mixed 'true' ;
				3.5.3 Ändern der Abgeschlossenheit		ja, anbieterType	√update complextype name 'anbieterType' change final 'restriction' ;
				3.5.4 Ändern des Inhalts (und Basis)	3.5.4.9 Extension: simple Content -> Extension: complex Content	ja, nameType	**(√)** *mit delete nameType in 2.3.1.2 und add nameType in 1.5.4 erledigt*
	3.6 upd element	3.6.1 upd element def	3.6.1 Ändern einer Elementdeklaration	3.6.1.1Umbenennen einer Elementdeklaration		ja, fahrplan in routenfahrplan	**(√)** *mit delete fahrplan in 2.6.1 und add routenfahrplan in 1.6.1 erledigt*
				3.6.1.2 Ändern der Typdefinition		ja, routenfahrplan, anbieter, sparpreise, erwachsene, kinder, gruppen, tageskarte, fahrradmitnahme, gründungsjahr	**(√)** *mit delete fahrplan in 2.6.1 und add routenfahrplan in 1.6.1 erledigt* √update element name 'anbieter' change type 'anbieterneuType' ; √update element name 'sparpreise' change type 'geldneuType' ; √update element name 'erwachsene' change type 'geldneuType' ; √update element name 'kinder' change type 'geldneuType' ; √update element name 'gruppen' change type 'geldneuType' ; √update element name 'tageskarte' change type 'geldneuType' ; √update element name 'fahrradmitnahme' change type 'geldneuType' ; √*update element name 'gründungsjahr' change type 'gründungsjahrType' ;*
				3.6.1.6 Ändern der Nullwert-fähigkeit	3.6.1.6.2 nillable= "true" einfügen: minOccurs ≥ 0 maxOccurs ≥ 0	ja, ausstattung	**(√)***mit delete ausstattung in 2.6.1 und add ausstattung in 1.6.1 erledigt*
		3.6.2 upd elementref	3.6.2 Ändern einer Elementdeklaration Referenz	3.6.2.1 Ändern des Referenznamen		ja, fernbuslinie in routen (fernbusverkehrType), hinfahrt in fernbusanbieter-ref und rückfahrt in routenfahrplan (routeType)	**(√)** *mit add routen in 1.6.2.1.2 erledigt* *update elementref 'fernbuslinie' at '/node()/node()[@name="fernbusverkehrType"] /node()/.' change ref 'fernbusanbieter' maxoccurs '';* **(√)** *mit add fernbusanbieter-ref in 1.6.2.1.2* *update elementref 'hinfahrt' at '/node()/node()[@name="routeType"]/node()/.' change move to '/node()/node()[@name="routenfahrplanType"]/node()/.' ;* **(√)***mit add routenfahrplan in 1.6.2.1.2 erledigt* *update elementref 'rückfahrt' at '/node()/node()[@name="routeType"]/node()/.' change move to '/node()/node()[@name="routenfahrplanType"]/node()/.' ;* X update elementref 'route' at '/node()/node()[@name="fahrplanType"]/node()/.' change move to '/node()/node()[@name="routenType"]/node()/.' ;
				3.6.2.3 Erhöhen des minimalen Vorkommens		ja, fernbusanbieter-ref und routenfahrplan (routeType), sparpreise erwachsene, kinder, gruppen, tageskarte, fahrradmitnahme (preiseType)	**(√)***mit add fernanbusanbierter-ref und routenfahrplan in* 1.6.2.1.2 *erledigt* √update elementref 'sparpreise' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ; √update elementref 'erwachsene' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ; √update elementref 'kinder' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ; √update elementref 'gruppen' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ; √update elementref 'tageskarte' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ; √update elementref 'fahrradmitnahme' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ;
				3.6.2.5 Erhöhen des maximalen Vorkommens		ja, routen (fernbusverkehrType)	*update elementref 'anbieter' at '/node()/node()[@name="fernbuslinieType"]/node()/.' change maxoccurs 'unbounded' move to '/node()/node()[@name="fernbusanbieterType"]/node()/.' ;*
	3.7 updmodule		3.7 Ändern einer Moduldeklaration	3.7.2 Ändern des Typs	3.7.2.9 include -> override	ja, zeit.xsd	**(√)***mit delete include in 2.7.3 und add override in 1.7.4*
	3.8 updconstraint		3.8 Ändern einer Constraint-deklaration	3.8.1 Umbenennen der Schlüssel-/Identitätsdeklaration		ja, anbieterKey in anbieternameKey	**(√)***mit 3.8.4.1 erledigt*
				3.8.2 Ändern des Typs	3.8.2.1 key -> unique	ja, routeKey	√update constraint name 'routeKey' at '/node()/node()[@name="fernbusverkehr"]/.'change type 'unique' remove selector './fernbuslinie/fahrplan/route' insert selector './routen/route' ;
				3.8.4 Ändern des Constraint-pfades	3.8.4.1 Ändern des selector-Pfades	ja, anbieterKey	√update constraint name 'anbieterKey' at '/node()/node() [@name="fernbusverkehr"]/.' change name 'anbieternameKey' insert selector './fernbusanbieter/anbieter' insert field 'name/@akronym' at '/node()/node() [@name="fernbusverkehr"] /node()[@name="anbieternameKey"]/node()[2]' ;
	3.9 updschema		3.9 Ändern einer Schemadeklaration	3.9.4 Ändern der Version		ja, von 1.0 auf 1.1	√update schema change version '1.1' ;
√ = umgesetzt, (√) = teilweise oder andersweitig umgesetzt, X = nicht umgesetzt							

Abbildung 6.3: Abgleich der ELaX-Update-Operationen abgeleitet aus D

6.2 Vergleich und Bewertung mit der herkömmlichen Vorgehensweise in CodeX

Im vorherigen Abschnitt wurden die von *CodeXMapper* generierten ELaX-Ausdrücke gegen das Fernbusverkehr-Szenario abgeglichen, wobei in diesem Abschnitt ein Vergleich mit den in CodeX generierten Operationen erfolgt. Anschließend werden Kombinationsmöglichkeiten zwischen den beiden Tools analysiert.
Beim Abgleich der von den beiden Tools erzeugten ELaX-Ausdrücken (siehe Anhang D) kann generell festgestellt werden, dass beispielsweise beim Ändern von Attribut und/oder Elementnamen der *CodeXMapper* dahin tendiert, dass bei nicht ermitteltem Mapping die Attribute bzw. Elemente aus dem Quellschema gelöscht und anschließend neu eingefügt werden. Außerdem bietet das Programm *CodeXMapper* Alternativausdrücke zu denen in CodeX an. So ist im Szenario eine Umbenennung der Hinfahrt-Elementreferenz in „fernbusanbieter-ref“ und Einfügen eines Hinfahrt-Elements mit den gleichen Attributwerten in den RoutenfahrplanType-Typ vorgesehen. Dazu bietet CodeX exakt diese Abarbeitung an, siehe Theorem 6.1, während der *CodeXMapper* dazu eine Verschiebung der schon vorhandenen Elementreferenz anbietet und dafür eine neue „fernbusanbieter-ref“ einfügt, siehe Theorem 6.2.

$$\begin{aligned} &update\ elementref\ 'hinfahrt'\ at\ '/node()/node()[@name = "routeType"]/node()/.' \\ &change\ ref\ 'fernbusanbieter - ref'\ minoccurs\ '' ; \end{aligned} \tag{6.1}$$

$$\begin{aligned} &update\ elementref\ 'hinfahrt'\ at\ '/node()/node()[@name = "routeType"]/node()/.' \\ &change\ move\ to\ '/node()/node()[@name = "routenfahrplanType"]/node()/.' ; \end{aligned} \tag{6.2}$$

Weiterhin sind feine Unterschiede in der XPath-Adressierung vorhanden, siehe Theoreme 6.3 und 6.4.

$$add\ elementref\ 'ab'\ as\ first\ into\ '/node()/node()/node()[@name = "zeitType"]/node()/.' ; \tag{6.3}$$

$$add\ elementref\ 'ab'\ in\ '/node()/node()/node()[@name = "zeitType"]/node()/.' ; \tag{6.4}$$

Dabei ist zu sehen, dass ELaX verschiedene XPath-Ausdrücke beherrscht, sodass ein genaues Adressieren der Schemakomponenten, vor allem für Referenzdeklarationen, möglich ist. So ist beispielsweise in 6.3 die Adressierung über die Position möglich, d.h. im Beispiel mit „first into“, aber auch „last“, „all“ oder Angabe einer expliziten Positionsnummer sind erlaubt (vgl. [109]). Abgesehen von den beschriebenen Fällen stimmen alle ELaX-Ausdrücke zwischen den beiden erzeugten Tools überein, womit die Grundlage für Kombinationsmöglichkeiten der beiden Tools geschaffen wird.
Als eine große Herausforderung stellt sich die Erzeugung der semantisch richtigen ELaX-Ausdrücke heraus. In beiden Tools wird eine Benutzeroberfläche zur Verfügung gestellt, die den Nutzer mit in den Prozess der Schemaevolution involviert. Dabei findet CodeX eine Abstrahierung der XML-Schemata in Form des konzeptionellen Modells *Entity Model for XML-Schema* (EMX) dar. Im Gegensatz dazu arbeitet der *CodeXMapper* auf der Ebene der XML-Schemata, d.h. es findet keine Abstrahierung statt, sodass meist Expertenwissen zur Verfügung stehen muss, um zum Einen Anpassungen an den Mappings vorzunehmen und zum Anderen die erzeugten bzw. angezeigten ELaX-Ausdrücke zu verstehen. Deshalb wäre es sinnvoll, die beiden Tools dahingehend zu kombinieren, dass auf der einen Seite die hochgeladenen Quell- und Zielschemata in CodeX in EMX übersetzt werden. Auf der anderen Seite kann der *CodeXMapper* dazu verwendet werden die Mappings auf Schemaebene zu finden und sie entsprechend im konzeptionellen Modell anzeigen zu lassen. Somit könnte auch ein unerfahrender Nutzer Anpassungen am XML-Schema vornehmen bzw. bekommt auf diesem Wege mehrere Alternativen zum möglichen Evolutionsablauf aufgezeigt.

Weiterhin kann ein Nutzer ein Quell- und Ziel-EMX zur Verfügung stellen bzw. selber erzeugen, um dann daraus die XML-Schemas abzuleiten und gegebenenfalls Mappings bzw. ELaX-Ausdrücke über das *CodeXMapper*-Tool zu generieren. Dabei könnte eine mögliche Erweiterung sein, dass die Logik zur Erzeugung der Mappings und ELaX-Operationen auf die konzeptionelle Ebene übertragen wird, um so die Qualität der Schemaevolution bzw. des Ergebnisses in Form des letztendlich aus den ELaX-Operationen erzeugten XSLT-Skripten weiter zu erhöhen.

Kapitel 7

Zusammenfassung

Diese Masterarbeit untersucht, inwieweit die XML-Schemaevolution verbessert werden kann, indem ein eigener Ansatz auf Basis von Matching-und Mappingverfahren entwickelt wird. Zunächst erfolgte eine Analyse und Klassifizierung der verschiedenen Matching- und Mappingverfahren. Davon ausgehend wurde die Umsetzung der klassifizierten Verfahren in verschiedenen Forschungsprototypen und kommerziellen Systemen betrachtet und gegenübergestellt. Davon ausgehend wurde sich für die Verwendung des Matcher-Tool *Combining Matchers* (COMA) entschieden, welches die Grundlage zur Entwicklung des eigenen Ansatzes darstellt. Davon ausgehend wurden Konzepte entwickelt, wie das Mappingergebnis von COMA in adäquate *Evolution Language for XML-Schema* (ELaX)-Ausdrücke übersetzt werden kann. Dabei wurden Methoden entworfen, die zum Einen die Funktionalität der ELaX-Erzeugung und zum Anderen die Visualisierung in CodeX zu integrieren. Bezüglich der Schemaevolution wurde ein selbstgewähltes Beispiel vorgestellt, welches gegen eine Kategorisierung von Schemaevolutionsschritten abgeglichen wurde. Dieses Szenario und die von CodeX generierten ELaX-Ausdrücke wurden zur Evaluierung des entwickelten Ansatzes eingesetzt, um die korrekte Abarbeitung zu gewährleisten. Dabei sind ebenfalls Kombinationsmöglichkeiten erläutert worden.
Ein in der Arbeit nicht enthaltender Aspekt ist die Verwendung von zusätzlichen semantischen Informationen, wie der Einsatz von Synonymwörterbücher oder Ontologien (vgl. [8]). Diese Informationen könnten zur Steigerung beim Finden von semantischen Beziehungen eingesetzt werden. Außerdem wurde sich in der Umsetzung des eigenen Ansatzes auf eine optimierte Auswahl von Matchern des COMA-Tool beschränkt, sodass eine Prüfung zur Integration weiterer Tools und/oder Mappingergebnissen die Schemaevolution noch weiter verbessern kann.

Anhang A

Schema Matching Klassifikation

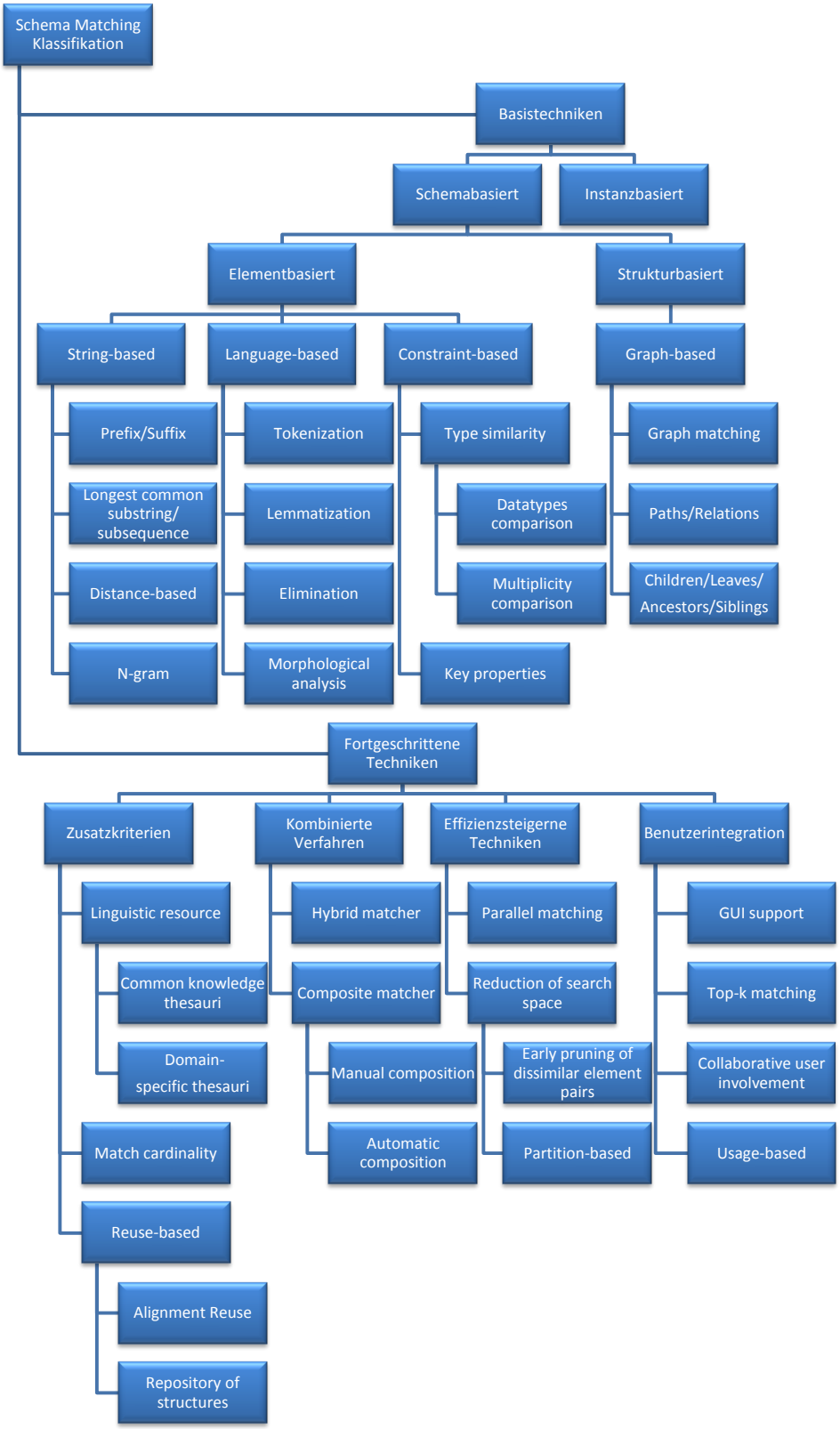

Abbildung A.1: Schema Matching Klassifikation komplett abgeleitet aus [123] und [17]

Anhang B

Schemaevolutionsschritte

ELaX-Operation			Schemaevolutionsschritt			Instanzanpassung	Umsetzung im Beispiel
1. add	1.1 addannotation		1.1 Hinzufügen einer Anmerkung	1.1.1 Hinzufügen einer documentation		nein	ja, analog zu 3.1.1
				1.1.2 Hinzufügen einer appinfo		nein	ja, analog zu 3.1.1
	1.2 addattributegroup	1.2.1 addattributegroupdef	1.2.1 Hinzufügen einer Attributgruppen Definition			nein	ja, tour
		1.2.2 addattribute	1.2.2 Hinzufügen einer Attributdeklaration			nein	ja, start, ziel
		1.2.3 addattributeref	1.2.3 Hinzufügen einer Attributdeklaration Referenz	1.2.3.1 "prohibited"		nein	ja, analog zu 1.2.3.3
				1.2.3.2 "optional"		nein	ja, rythmus (zeitType)
				1.2.3.3 "required"		ja	ja, start und ziel (tour), anbieterID (anbieterneuType)
		1.2.4 addattributegroupref	1.2.4 Hinzufügen einer Attributgruppen Definition Referenz			ja/nein	ja, tour (routenType)
		1.2.5 addattributewildcard	1.2.5 Hinzufügen einer Attribut-Wildcard			nein	ja, analog zu 1.6.3.2
	1.3 addgroup		1.3 Hinzufügen einer Gruppendefinition	1.3.1 Sequenz	1.3.1.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, analog zu 1.3.1.2
					1.3.1.2 minOccurs > 0 maxOccurs > 0	nein	ja, zeitType, routenfahrplanType, fernbusanbieterType
				1.3.2 Alternative	1.3.2.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, analog zu 1.3.1.2
					1.3.2.2 minOccurs > 0 maxOccurs > 0	nein	ja, analog zu 1.3.1.2
				1.3.3 Menge	1.3.3.1 minOccurs = 0 maxOccurs = 1	nein	ja, analog zu 1.3.1.2
					1.3.3.2 minOccurs = 1 maxOccurs = 1	nein	ja, analog zu 1.3.1.2
	1.4 addst		1.4 Hinzufügen einer einfachen Typdefinition in Hierarchie als	1.4.1 Einschränkung		nein	ja, gründungsjahrType, fehlType
				1.4.2 Liste		nein	ja, analog zu 1.4.1
				1.4.3 Vereinigung		nein	ja, geldneuType
	1.5 addct		1.5 Hinzufügen einer komplexen Typdefinition in Hierarchie als	1.5.1 Einschränkung: simpleContent		nein	ja, analog zu 1.5.4
				1.5.2 Einschränkung:complexContent		nein	ja, analog zu 1.5.4
				1.5.3 Erweiterung: simpleContent		nein	ja, analog zu 1.5.4
				1.5.4 Erweiterung: complexContent		nein	ja, anbieterneuType
	1.6 addelement	1.6.1 addelementdef	1.6.1 Hinzufügen einer Elementdeklaration			nein	ja, routen, fernbusanbieter
		1.6.2 addelementref	1.6.2 Hinzufügen einer Elementdeklaration Referenz in	1.6.2.1 Sequenz	1.6.2.1.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, hinfahrt und rückfahrt (routenfahrplanType)
					1.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja	ja, fernbusanbieter (fernbusverkehrType), anbieter(fernbusanbieterType), ausstattung (anbieterneuType), ab und an (zeitType)
				1.6.2.2 Alternative	1.6.2.2.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, analog zu 1.6.2.1.1
					1.6.2.2.2 minOccurs > 0 maxOccurs > 0	nein	ja, analog zu 1.6.2.1.2
				1.6.2.3 Menge	1.6.2.3.1 minOccurs = 0 maxOccurs ≥ 0	nein	ja, analog zu 1.6.2.1.1
					1.6.2.3.2 minOccurs > 0 maxOccurs > 0	ja	ja, analog zu 1.6.2.1.2
		1.6.3 addelementwildcard	1.6.3 Hinzufügen einer Wildcard	1.6.3.1 minOccurs = 0 maxOccurs ≥ 0		nein	ja, analog zu 1.6.3.2
				1.6.3.2 minOccurs > 0 maxOccurs > 0		ja	ja, any (ausstattungType)
	1.7 addmodule		1.7 Hinzufügen einer Moduldeklaration	1.7.1 import		nein	ja, analog zu 3.7.2.9
				1.7.2 redefine		nein	ja, analog zu 3.7.2.9
				1.7.3 include		nein	ja, analog zu 3.7.2.9
				1.7.4 override		nein	ja, analog zu 3.7.2.9
	1.8 addconstraint		1.8 Hinzufügen einer Constraintdeklaration	1.8.1 Hinzufügen eines Primärschlüssels	1.8.1.1 key	ja	ja, routenKey (fernbusverkehr)
					1.8.1.2 unique	ja	ja, analog zu 1.8.1.1
				1.8.2 Hinzufügen eines Fremdschlüssels (keyref)		ja	ja, startzielKey und zielstartKey
				1.8.3 Hinzufügen einer Identität		nein	ja, anbieterID
				1.8.4 Hinzufügen einer Identitätsreferenz		ja	ja, fernbusanbieter-ref
2. delete	2.1 delannotation		2.1 Entfernen einer Anmerkung	2.1.1 Entfernen einer documentation		nein	ja, analog zu 3.1.1
				2.1.2 Entfernen einer appinfo		nein	ja, analog zu 3.1.1
	2.2 delattributegroup	2.2.1 delattributegroupdef	2.2.1 Entfernen einer Attributgruppen Definition			ja/nein	ja, analog zu 2.2.2
		2.2.2 delattribute	2.2.2 Entfernen einer Attributdeklaration			ja/nein	ja, MwSt.
		2.2.3 delattributeref	2.2.3 Entfernen einer Attributdeklaration Referenz	2.2.3.1 "prohibited"		nein	ja, analog zu 2.2.3.3
				2.2.3.2 "optional"		ja/nein	ja, analog zu 2.2.3.3
				2.2.3.3 "required"		ja	ja, MwSt. (preiseType)
		2.2.4 delattributegroupref	2.2.4 Entfernen einer Attributgruppen Definition Referenz			ja/nein	ja, analog zu 2.2.3.3
		2.2.5 delattributewildcard	2.2.5 Entfernen einer Attribut-Wildcard			ja/nein	ja, analog zu 2.2.3.3
	2.3 delgroup		2.3 Entfernen einer Gruppendefinition	2.3.1 Sequenz	2.3.1.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.3.1.2
					2.3.1.2 minOccurs > 0 maxOccurs > 0	ja/nein	ja, fernbuslinieType
				2.3.2 Alternative	2.3.2.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.3.1.2
					2.3.2.2 minOccurs > 0 maxOccurs > 0	ja/nein	ja, analog zu 2.3.1.2
				2.3.3 Menge	2.3.3.1 minOccurs = 0 maxOccurs = 1	ja/nein	ja, analog zu 2.3.1.2
					2.3.3.2 minOccurs = 1 maxOccurs = 1	ja/nein	ja, analog zu 2.3.1.2
	2.4 delst		2.4 Entfernen einer einfachen Typdefinition aus Hierarchie	2.4.1 Einschränkung		nein	ja, MwSt.Type, serviceType, sitzabstandType, wahlType
				2.4.2 Liste		ja/nein	ja, cateringType
				2.4.3 Vereinigung		ja/nein	ja, analog zu 2.4.2
	2.5 delct		2.5 Entfernen einer komplexen Typdefinition aus Hierarchie	2.5.1 Einschränkung: simpleContent		nein	ja, analog zu 2.4.1
				2.5.2 Einschränkung:complexContent		nein	ja, analog zu 2.4.1
				2.5.3 Erweiterung: simpleContent		ja/nein	ja, analog zu 2.4.2
				2.5.4 Erweiterung: complexContent		ja/nein	ja, analog zu 2.4.2
	2.6 delelement	2.6.1 delelementdef	2.6.1 Entfernen einer Elementdeklaration			ja/nein	ja, fernbuslinie, catering, sitzabstand, steckdosen, wlan, beschreibung
		2.6.2 delelementref	2.6.2 Entfernen einer Elementdeklaration Referenz aus	2.6.2.1 Sequenz	2.6.2.1.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.6.2.1.2
					2.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja	ja, catering, sitzabstand, steckdosen, wlan (ausstattungType)
				2.6.2.2 Alternative	2.6.2.2.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.6.2.1.2
					2.6.2.2.2 minOccurs > 0 maxOccurs > 0	ja/nein	ja, analog zu 2.6.2.1.2
				2.6.2.3 Menge	2.6.2.3.1 minOccurs = 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 2.6.2.1.2
					2.6.2.3.2 minOccurs > 0 maxOccurs > 0	ja	ja, analog zu 2.6.2.1.2
		2.6.3 delelementwildcard	2.6.3 Entfernen einer Wildcard	2.6.3.1 minOccurs = 0 maxOccurs ≥ 0		ja/nein	ja, analog zu 2.6.2.1.2
				2.6.3.2 minOccurs > 0 maxOccurs > 0		ja	ja, analog zu 2.6.2.1.2
	2.7 delmodule		2.7 Entfernen einer Moduldeklaration	2.7.1 import		ja/nein	ja, analog zu 3.7.2.9
				2.7.2 redefine		ja/nein	ja, analog zu 3.7.2.9
				2.7.3 include		ja/nein	ja, analog zu 3.7.2.9
				2.7.4 override		ja/nein	ja, analog zu 3.7.2.9
	2.8 delconstraint		2.8 Entfernen einer Constraintdeklaration	2.8.1 Entfernen eines Primärschlüssels	2.8.1.1 key	nein	ja, analog zu 3.8.4.1
					2.8.1.2 unique	nein	ja, analog zu 3.8.4.1
				2.8.2 Entfernen eines Fremdschlüssels (keyref)		nein	ja, analog zu 3.8.4.1
				2.8.3 Entfernen der Identität		ja/nein	ja, analog zu 2.2.2
				2.8.4 Entfernen einer Identitätsreferenz		ja	ja, analog zu 2.2.2
3. update	3.1 updannotation		3.1 Ändern einer Anmerkung	3.1.1 Ändern einer documentation		nein	ja, von 2012 auf 2013
				3.1.2 Ändern einer appinfo		nein	ja, analog zu 3.1.1
	3.2 updattributegroup	3.2.1 updattributegroupdef	3.2.1 Ändern einer Attributgruppen Definition	3.2.1.1 Umbenennen einer Attributgruppen Definition		nein	ja, analog zu 3.2.2.1
				3.2.1.2 Hinzufügen einer Attributdeklaration Referenz		ja/nein	ja, analog zu 1.2.3.3
				3.2.1.3 Entfernen einer Attributdeklaration Referenz		ja/nein	ja, analog zu 2.2.3.3
				3.2.1.4 Hinzufügen einer Attribut-Wildcard		nein	ja, analog zu 1.6.3.2
				3.2.1.5 Entfernen einer Attribut-Wildcard		ja/nein	ja, analog zu 2.2.3.3
		3.2.2 updattribute	3.2.2 Ändern einer Attributdeklaration	3.2.2.1 Umbenennen einer Attributdeklaration		ja/nein	ja, abkürzung in akronym
				3.2.2.2 Ändern der Typdefinition		ja/nein	ja, routeID
				3.2.2.3 Ändern des fixed-Wertes		ja	ja, last_schema_change
				3.2.2.4 Ändern des default-Wertes		nein	ja, analog zu 3.2.2.3
		3.2.3 updattributeref	3.2.3 Ändern einer Attributdeklaration Referenz	3.2.3.1 Ändern des Referenznamen		ja/nein	ja, abkürzung in akronym (nameType)
				3.2.3.2 Ändern des fixed-Wertes		ja	ja, analog zu 3.2.2.3
				3.2.3.3 Ändern des default-Wertes		nein	ja, analog zu 3.2.2.3
				3.2.3.4 Ändern der Attributverwendung	3.2.3.4.1 "optional" -> "required"	ja/nein	ja, routeID (routeType)
					3.2.3.4.2 "optional" ->"prohibited"	ja/nein	ja, analog zu 3.2.3.4.1
					3.2.3.4.3 "required" -> "optional"	nein	ja, analog zu 3.2.3.4.1
					3.2.3.4.4 "required" ->"prohibited"	ja	ja, analog zu 3.2.3.4.1
					3.2.3.4.5 "prohibited"->"optional"	nein	ja, analog zu 3.2.3.4.1
					3.2.3.4.6 "prohibited"->"required"	ja	ja, analog zu 3.2.3.4.1
		3.2.4 updattributegroupref	3.2.4 Ändern einer Attributgruppen Definition Referenz	3.2.4.1 Ändern des Referenznamen		ja/nein	ja, analog zu 3.2.3.1
		3.2.5 updattributewildcard	3.2.5 Ändern einer Attribut-Wildcard	3.2.5.1 Ändern der Namensräume		ja/nein	ja, analog zu 1.6.3.2
				3.2.5.2 Ändern der Validierung		ja/nein	ja, analog zu 1.6.3.2
	3.3 updgroup		3.3 Ändern einer Gruppendefinition	3.3.1 Ändern des Typs	3.3.1.1 Sequenz -> Menge	nein	ja, analog zu 3.3.1.5
					3.3.1.2 Sequenz -> Alternative	ja/nein	ja, analog zu 3.3.1.4
					3.3.1.3 Alternative->Menge	ja/nein	ja, analog zu 3.3.1.4

Abbildung B.1: Kategorisierung der Schemaevolutionsschritte Teil 1/2 abgeleitet aus [38] und [112]

					3.3.1.4 Alternative->Sequenz	ja/nein	ja, zeitType und zeit.xsd
					3.3.1.5 Menge -> Sequenz	ja/nein	ja, preiseType
					3.3.1.6 Menge->Alternative	ja/nein	ja, analog zu 3.3.1.4
				3.3.2 Verringern des minimalen Vorkommens		nein	ja, preiseType
				3.3.3 Erhöhen des minimalen Vorkommens		ja/nein	ja, analog zu 3.3.2
				3.3.4 Verringern des maximalen Vorkommens		ja/nein	ja, analog zu 3.3.5
				3.3.5 Erhöhen des maximalen Vorkommens		nein	ja, preiseType
	3.4 updst		3.4 Ändern einer einfachen Typdefinition	3.4.1 Umbenennen der Typdefinition		nein	ja, analog zu 3.5.1
				3.4.2 Ändern des Typs	3.4.2.1 atomar -> Liste	nein	ja, analog zu 1.4.3
					3.4.2.2 atomar -> Vereinigung	nein	ja, analog zu 1.4.3
					3.4.2.3 Liste -> Vereinigung	ja/nein	ja, analog zu 3.4.2.6
					3.4.2.4 Liste -> atomar	ja/nein	ja, analog zu 3.4.2.6
					3.4.2.5 Vereinigung -> atomar	ja/nein	ja, analog zu 3.4.2.6
					3.4.2.6 Vereinigung -> Liste	ja/nein	ja, buslinienType
				3.4.3 Hinzufügen einer einfachen Typdefinition zu einer Vereinigung		nein	ja, analog zu 3.4.4
				3.4.4 Entfernen einer einfachen Typdefinition aus einer Vereinigung		ja/nein	ja, buslinienType
				3.4.5 Hinzufügen einer Einschränkung		ja/nein	ja, analog zu 3.4.7
				3.4.6 Entfernen einer Einschränkung		nein	ja, analog zu 3.4.7
				3.4.7 Modifizieren einer Einschränkung		ja/nein	ja, buslinieType, gründungsjahrType
				3.4.8 Ändern der Abgeschlossenheit		ja/nein	ja, analog zu 3.5.3
	3.5 updct		3.5 Ändern einer komplexen Typdefinition	3.5.1 Umbenennen der Typdefinition		nein	ja, fahrplanType in routenType
				3.5.2 Ändern des Inhaltstyps auf	3.5.2.1 mixed = "true"	nein	ja, beschreibungType
					3.5.2.2 mixed = "false"	ja	ja, analog zu 3.5.2.1
				3.5.3 Ändern der Abgeschlossenheit		ja/nein	ja, anbieterType
				3.5.4 Ändern des Inhalts (und Basis)	3.5.4.1 Einschränkung: simpleContent -> Einschränkung: complexContent	ja/nein, nur ohne Facets	ja, analog zu 3.5.4.9
					3.5.4.2 Einschränkung: simpleContent -> Erweiterung: simpleContent	ja/nein, nur ohne Facets	ja, analog zu 3.5.4.9
					3.5.4.3 Einschränkung: simpleContent -> Erweiterung: complexContent	ja/nein, nur ohne Facets	ja, analog zu 3.5.4.9
					3.5.4.4 Einschränkung: complexContent ->Einschränkung: simpleContent	ja/nein, nur ohne Inhaltsmodelle	ja, analog zu 3.5.4.9
					3.5.4.5 Einschränkung: complexContent -> Erweiterung: simpleContent	ja/nein, nur ohne Inhaltsmodelle	ja, analog zu 3.5.4.9
					3.5.4.6 Einschränkung: complexContent -> Erweiterung: complexContent	ja/nein	ja, analog zu 3.5.4.9
					3.5.4.7 Erweiterung: simpleContent -> Einschränkung: simpleContent	ja/nein	ja, analog zu 3.5.4.9
					3.5.4.8 Erweiterung: simpleContent -> Einschränkung: complexContent	ja/nein	ja, analog zu 3.5.4.9
					3.5.4.9 Erweiterung: simpleContent -> Erweiterung: complexContent	ja/nein	ja, nameType
					3.5.4.10 Erweiterung: complexContent -> Einschränkung: simpleContent	ja/nein, nur ohne Inhaltsmodelle	ja, analog zu 3.5.4.9
					3.5.4.11 Erweiterung: complexContent -> Einschränkung: complexContent	ja/nein	ja, analog zu 3.5.4.9
					3.5.4.12 Erweiterung: complexContent -> Erweiterung: simpleContent	ja/nein, nur ohne Inhaltsmodelle	ja, analog zu 3.5.4.9
	3.6 updelement	3.6.1 updelementdef	3.6.1 Ändern einer Elementdeklaration	3.6.1.1 Umbenennen einer Elementdeklaration		ja	ja, fahrplan in routenfahrplan
				3.6.1.2 Ändern der Typdefinition		ja/nein	ja, routenfahrplan, anbieter, sparpreise, erwachsene, kinder, gruppen, tageskarte, fahrradmitnahme, gründungsjahr
				3.6.1.3 Ändern des fixed-Wertes		ja	ja, analog zu 3.2.2.3
				3.6.1.4 Ändern des default-Wertes		nein	ja, analog zu 3.2.2.3
				3.6.1.5 Ändern der Abgeschlossenheit		ja/nein	ja, analog zu 3.5.3
				3.6.1.6 Ändern der Nullwertfähigkeit	3.6.1.6.1 nillable= "false" einfügen / 3.6.1.6.1 minOccurs ≥ 0 maxOccurs ≥ 0	nein	ja, analog zu 3.6.1.6.2
					3.6.1.6.2 nillable= "true" einfügen / 3.6.1.6.2 minOccurs ≥ 0 maxOccurs ≥ 0	ja/nein	ja, ausstattung
					3.6.1.6.3 nillable="true"->nillable="false" / 3.6.1.6.3 minOccurs ≥ 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 3.6.1.6.2
					3.6.1.6.4 nillable="false"->nillable="true" / 3.6.1.6.4 minOccurs ≥ 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 3.6.1.6.2
					3.6.1.6.5 nillable= "false" entfernen / 3.6.1.6.5 minOccurs ≥ 0 maxOccurs ≥ 0	nein	ja, analog zu 3.6.1.6.2
					3.6.1.6.6 nillable= "true" entfernen / 3.6.1.6.6 minOccurs ≥ 0 maxOccurs ≥ 0	ja/nein	ja, analog zu 3.6.1.6.2
		3.6.2 updelementref	3.6.2 Ändern einer Elementdeklaration Referenz	3.6.2.1 Ändern des Referenznamen		ja/nein	ja, fernbuslinie in routen (fernbusverkehrType), hinfahrt in fernbusanbieter-ref und rückfahrt in routenfahrplan (routeType)
				3.6.2.2 Verringern des minimalen Vorkommens		nein	ja, analog zu 3.6.2.3
				3.6.2.3 Erhöhen des minimalen Vorkommens		ja	ja, fernbusanbieter-ref und routenfahrplan (routeType), sparpreise erwachsene, kinder, gruppen, tageskarte, fahrradmitnahme (preiseType)
				3.6.2.4 Verringern des maximalen Vorkommens		ja	ja, analog zu 3.6.2.5
				3.6.2.5 Erhöhen des maximalen Vorkommens		nein	ja, routen (fernbusverkehrType)
		3.6.3 updelementwildcard	3.6.3 Ändern einer Wildcard	3.6.3.1 Ändern der Namensräume		ja/nein	ja, analog zu 1.6.3.2
				3.6.3.2 Ändern der Validierung		ja/nein	ja, analog zu 1.6.3.2
				3.6.3.3 Verringern des minimalen Vorkommens		nein	ja, analog zu 3.6.2.3
				3.6.3.4 Erhöhen des minimalen Vorkommens		ja	ja, analog zu 3.6.2.3
				3.6.3.5 Verringern des maximalen Vorkommens		ja	ja, analog zu 3.6.2.5
				3.6.3.6 Erhöhen des maximalen Vorkommens		nein	ja, analog zu 1.6.3.2 (von 1 auf unbounded)
	3.7 updmodule		3.7 Ändern einer Moduldeklaration	3.7.1 Umbenennen der Moduldeklaration		nein	ja, analog zu 3.6.1.1
				3.7.2 Ändern des Typs	3.7.2.1 import -> redefine	nein	ja, analog zu 3.7.2.9
					3.7.2.2 import -> include	nein	ja, analog zu 3.7.2.9
					3.7.2.3 import -> override	nein	ja, analog zu 3.7.2.9
					3.7.2.4 redefine -> import	nein	ja, analog zu 3.7.2.9
					3.7.2.5 redefine -> include	nein	ja, analog zu 3.7.2.9
					3.7.2.6 redefine -> override	nein	ja, analog zu 3.7.2.9
					3.7.2.7 include -> import	nein	ja, analog zu 3.7.2.9
					3.7.2.8 include -> redefine	nein	ja, analog zu 3.7.2.9
					3.7.2.9 include -> override	nein	ja, zeit.xsd
					3.7.2.10 override -> import	nein	ja, analog zu 3.7.2.9
					3.7.2.11 override -> include	nein	ja, analog zu 3.7.2.9
					3.7.2.12 override -> redefine	nein	ja, analog zu 3.7.2.9
	3.8 updconstraint		3.8 Ändern einer Constraintdeklaration	3.8.1 Umbenennen der Schlüssel-/Identitätsdeklaration		nein	ja, anbieterKey in anbieternameKey
				3.8.2 Ändern des Schlüsseltyps	3.8.2.1 key -> unique	nein	ja, routeKey
					3.8.2.2 key -> keyref	ja/nein	ja, analog zu 3.8.2.1
					3.8.2.3 keyref -> key	ja/nein	ja, analog zu 3.8.2.1
					3.8.2.4 keyref -> unique	ja/nein	ja, analog zu 3.8.2.1
					3.8.2.5 unique -> key	ja/nein	ja, analog zu 3.8.2.1
					3.8.2.6 unique -> keyref	ja/nein	ja, analog zu 3.8.2.1
				3.8.3 Ändern des Identitätstyps	3.8.3.1 id -> idref	ja	ja, analog zu 3.8.2.1
					3.8.3.2 idref -> id	ja	ja, analog zu 3.8.2.1
				3.8.4 Ändern des Constraintpfades	3.8.4.1 Ändern des selector-Pfades	ja/nein	ja, anbieterKey
					3.8.4.2 Ändern des field-Pfades	ja/nein	ja, analog zu 3.8.4.1
	3.9 updschema		3.9 Ändern einer Schemadeklaration	3.9.1 Ändern des Ziel-Namensraumes		nein	ja, analog zu 1.6.3.2
				3.9.2 Ändern des Ziel-Namensraumenspräfix		nein	ja, analog zu 3.9.4
				3.9.3 Ändern der Sprache (language)		nein	ja, analog zu 3.9.4
				3.9.4 Ändern der Version		nein	ja, von 1.0 auf 1.1
				3.9.5 Ändern der Elementform	3.9.5.1 unqualified -> qualified	nein	ja, analog zu 3.9.4
					3.9.5.2 qualified -> unqualified	nein	ja, analog zu 3.9.4
				3.9.6 Ändern der Attributform	3.9.6.1 unqualified -> qualified	nein	ja, analog zu 3.9.4
					3.9.6.2 qualified -> unqualified	nein	ja, analog zu 3.9.4
				3.9.7 Ändern der Abgeschlossenheit		ja/nein	ja, analog zu 3.5.3

Abbildung B.2: Kategorisierung der Schemaevolutionsschritte Teil 2/2 abgeleitet aus [38] und [112]

Anhang C

ELaX Übersetzungsregeln

Schemakomponente	Mapping-Beziehung	Match-kardinalität	Mapping-Regel			ELaX-Operation
1. xs:schema	1.1 V_s ↔ V_t	1:1	1.1.1 targetNamespace_s != targetNamespace_t			update schema change targetnamespace '*targetNamespace_t*';
			1.2.1 ¬targetNamespace_s AND targetNamespace_t			
			1.1.2 targetnamespaceprefix_s != targetnamespaceprefix_t			update schema change targetnamespaceprefix '*targetnamespaceprefix_t*';
			1.2.2 ¬targetnamespaceprefix_s AND targetnamespaceprefix_t			
			1.1.3 xml:lang_s != xml:lang_t			update schema change language '*xml:lang_t*';
			1.2.3 ¬xml:lang_s AND xml:lang_t			
			1.1.4 version_s != version_t			update schema change version '*version_t*';
			1.2.4 ¬version_s AND version_t			
			1.1.5 elementFormDefault_s != elementFormDefault_t			update schema change elementform '*elementFormDefault_t*';
			1.2.5 ¬elementFormDefault_s AND elementFormDefault_t			
	1.2 ¬V_s ← V_t		1.1.6 attributeFormDefault_s != attributeFormDefault_t			update schema change attributeform '*attributeFormDefault_t*';
			1.2.6 ¬attributeFormDefault_s AND attributeFormDefault_t			
			1.1.7 finalDefault_s != finalDefault_t			update schema change finaldefault '*finalDefault_t*';
			1.2.7 ¬finalDefault_s AND finalDefault_t			
			1.1.8 id_s != id_t			update schema change id '*id_t*';
			1.2.8 ¬id_s AND id_t			
			1.1.9 defaultAttributes_s != defaultAttributes_t			update schema change defaultattribute '*defaultAttributes_t*';
			1.2.9 ¬defaultAttributes_s AND defaultAttributes_t			
			1.1.10 xpathDefaultNamespace_s != xpathDefaultNamespace_t			update schema change xpathdefaultnamespace '*xpathDefaultNamespace_t*';
			1.2.10 ¬xpathDefaultNamespace_s AND xpathDefaultNamespace_t			
	1.3 V_s → ¬V_t	1:1	1.3.1 targetNamespace_s AND ¬targetNamespace_t			update schema change targetnamespace '';
			1.3.2 targetnamespaceprefix_s AND ¬targetnamespaceprefix_t			update schema change targetnamespaceprefix '';
			1.3.3 xml:lang_s AND ¬xml:lang_t			update schema change language '';
			1.3.4 version_s AND ¬version_t			update schema change version '';
			1.3.5 elementFormDefault_s AND ¬elementFormDefault_t			update schema change elementform '';
			1.3.6 attributeFormDefault_s AND ¬attributeFormDefault_t			update schema change attributeform '';
			1.3.7 finalDefault_s AND ¬finalDefault_t			update schema change finaldefault '';
			1.3.8 id_s AND ¬id_t			update schema change id '';
			1.3.9 defaultAttributes_s AND ¬defaultAttributes_t			update schema change defaultattribute '';
			1.3.10 xpathDefaultNamespace_s AND ¬xpathDefaultNamespace_t			update schema change xpathdefaultnamespace '';
2. xs:import, xs:redefine, xs:include, xs:override	2.1 V_s ↔ V_t	1:1	2.1.1 import_s_namespace != import_t_namespace			update module at '*/import_s|redefine_s|include_s| override_s)_locator*' change mode import with namespace '*import_t_namespace*';
			2.1.2 (import_s|redefine_s|include_s|override_s)_schemaLocation != (import_t|redefine_t|include_t|override_t)_schemaLocation			update module at '*/import_s|redefine_s|include_s| override_s)_locator*' change from '*/import_t|redefine_t| include_t|override_t)_schemaLocation*';
			2.1.3 (import_s|redefine_s|include_s|override_s)_id != (import_t|redefine_t|include_t|override_t)_id			update module at '*/import_s|redefine_s|include_s| override_s)_locator*' change id '*/import_t|redefine_t| include_t|override_t)_id*';
		n:m	2.1.4 M[n] = {(import_s|redefine_s|include_s|override_s)_1,..., (import_s|redefine_s|include_s|override_s)_n), N[m] = {(import_t|redefine_t|include_t|override_t)_1,..., (import_t|redefine_t|include_t|override_t)_m}	2.1.4 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	2.1.4.1 n = m	loop n: (import_s|redefine_s|include_s|override_s)_n ↔ (import_t|redefine_t|include_t|override_t)_n (analog zu 2.1.1, 2.1.2 und 2.1.3)
					2.1.4.2 n > m	analog zu 2.1.4.1 (P[m]!=∅, loop m, O[n]\\{(import_s|redefine_s|include_s|override_s)_m})
					2.1.4.3 n < m	loop n: delete module at '*/import_s|redefine_s|include_s| override_s)_n_locator*';
						analog zu 2.1.4.1 (O[n]!=∅, P[m]\\{(import_t| redefine_t| include_t|override_t)_n})
						loop n: add module from '*/import_t|redefine_t| include_t|override_t)_n_schemaLocation*' [...] id = '*/import_t|redefine_t| include_t| override_t)_n_id*'; [...]: mode import with namespace '*import_t_n_namespace*' [...]: mode redefine (<addst>|<addct>|<addattributegroupdef>) [...]: mode include [...]: mode override(<addst>| <addct>|<addattributegroupdef>| <addelementdef>|<addattribute>)
	2.2 ¬V_s ← V_t	1:1	2.2.1 ¬(import_s|redefine_s|include_s|override_s)_id AND (import_t|redefine_t|include_t|override_t)_id			analog zu 2.1.3
			2.2.2 (redefine_s|include_s|override_s) AND import_t	2.2.2.1 Content(redefine_s|include_s|override_s) = Content(import_t)		update module at '*/redefine_s|include_s|override_s)_locator*' change mode import with namespace '*import_t_namespace*';
			2.2.3 (import_s|include_s|override_s) AND redefine_t	2.2.3.1 Content(import_s|include_s|override_s) = Content(redefine_t)		update module at '*/import_s|include_s|override_s)_locator*' change mode redefine;
			2.2.4 (import_s|redefine_s|override_s) AND include_t	2.2.4.1 Content(import_s|redefine_s|override_s) = Content(include_t)		update module at '*/import_s|redefine_s|override_s)_locator*' change mode include;
			2.2.5 (import_s|include_s|redefine_s) AND override_t	2.2.5.1 Content(import_s|include_s|redefine_s) = Content(override_t)		update module at '*/import_s|include_s|redefine_s)_locator*' change mode override;
		n:m	2.2.6 M[n] = {(redefine_s|include_s|override_s)_1,..., (redefine_s|include_s|override_s)_n), N[m] = {import_t_1,..., import_t_m}		2.2.6.1 n = m	loop n: (redefine_s|include_s|override_s)_n ← import_t_n (analog zu 2.2.2.1)
					2.2.6.2 n > m	analog zu 2.2.6.1 (loop m, M[n]\\{(redefine_s|include_s|override_s)_m})
						analog zu 2.1.4.2 ('*/redefine_s|include_s|override_s)_n_locator*')
					2.2.6.3 n < m	analog zu 2.2.6.1 (N[m]\\(import_t_n))
						analog zu 2.1.4.3 (add, import_t_n)
			2.2.7 M[n] = {(import_s|include_s|override_s)_1,..., (import_s|include_s|override_s)_n), N[m] = {redefine_t_1,..., redefine_t_m}		2.2.7.1 n = m	loop n: (import_s|include_s|override_s)_n ← redefine_t_n (analog zu 2.2.3.1)
					2.2.7.2 n > m	analog zu 2.2.7.1 (loop m, M[n]\\{(import_s|include_s|override_s)_m})
						analog zu 2.1.4.2 ('*/import_s|include_s|override_s)_n_locator*')
					2.2.7.3 n < m	analog zu 2.2.7.1 (N[m]\\(redefine_t_n))
						analog zu 2.1.4.3 (add, redefine_t_n)
			2.2.8 M[n] = {(import_s|redefine_s|override_s)_1,..., (import_s|redefine_s|override_s)_n), N[m] = {include_t_1,..., include_t_m}		2.2.8.1 n = m	loop n: (import_s|redefine_s|override_s)_n ← include_t_n (analog zu 2.2.4.1)
					2.2.8.2 n > m	analog zu 2.2.8.1 (loop m, M[n]\\{(import_s|redefine_s|override_s)_m})
						analog zu 2.1.4.2 ('*/import_s|redefine_s|override_s)_n_locator*')
					2.2.8.3 n < m	analog zu 2.2.8.1 (N[m]\\(include_t_n))
						analog zu 2.1.4.3 (add, include_t_n)
			2.2.9 M[n] = {(import_s|include_s|redefine_s)_1,..., (import_s|include_s|redefine_s)_n), N[m] = {override_t_1,..., override_t_m}		2.2.9.1 n = m	loop n: (import_s|include_s|redefine_s)_n ← override_t_n (analog zu 2.2.5.1)
					2.2.9.2 n > m	analog zu 2.2.9.1 (loop m, M[n]\\{(import_s|include_s|redefine_s)_m})
						analog zu 2.1.4.2 ('*/import_s|include_s|redefine_s)_n_locator*')
					2.2.9.3 n < m	analog zu 2.2.9.1 (N[m]\\(override_t_n))
						analog zu 2.1.4.3 (add, override_t_n)
	2.3 V_s → ¬V_t	1:1	2.3.1 (import_s|redefine_s|include_s|override_s)_id AND ¬(import_t|redefine_t|include_t|override_t)_id			update module at '*/import_s|redefine_s|include_s| override_s)_locator*' change id '';
			2.3.2 import_s AND (redefine_t|include_t|override_t)	2.3.2.1 Content(import_s) = Content(redefine_t|include_t|override_t)		update module at '*import_s_locator*' change mode (redefine|include|override);
			2.3.3 redefine_s AND (import_t|include_t|override_t)	2.3.3.1 Content(redefine_s) = Content(import_t|include_t|override_t)		update module at '*redefine_s_locator*' change mode (import with namespace '*import_t_namespace*'|include|override);
			2.3.4 include_s AND (import_t|redefine_t|override_t)	2.3.4.1 Content(include_s) = Content(import_t|redefine_t|override_t)		update module at '*include_s_locator*' change mode (import with namespace '*import_t_namespace*'|redefine|override);
			2.3.5 override_s AND (import_t|include_t|redefine_t)	2.3.5.1 Content(override_s) = Content(import_t|include_t|redefine_t)		update module at '*override_s_locator*' change mode (import with namespace '*import_t_namespace*'|include|redefine);
		n:m	2.3.6 M[n] = {import_s_1,..., import_s_n}, N[m] = {(redefine_t|include_t|override_t)_1,..., (redefine_t|include_t|override_t)_m}		2.3.6.1 n = m	loop n: import_s_n → (redefine_t|include_t|override_t)_n (analog zu 2.3.2.1)
					2.3.6.2 n > m	analog zu 2.3.6.1 (loop m, M[n]\\{import_s_m})
						analog zu 2.1.4.2 (delete, import_s_n_locator)
					2.3.6.3 n < m	analog zu 2.3.6.1 (N[m]\\{(redefine_s|include_s|override_s)_n})
						analog zu 2.1.4.3 (add, (redefine_s|include_s|override_s)_n)
			2.3.7 M[n] = {redefine_s_1,..., redefine_s_n), N[m] = {(import_t|include_t|override_t)_1,..., (import_t|include_t|override_t)_m}		2.3.7.1 n = m	loop n: redefine_s_n → (import_t|include_t|override_t)_n (analog zu 2.3.3.1)
					2.3.7.2 n > m	analog zu 2.3.7.1 (loop m, M[n]\\{redefine_s_m})
						analog zu 2.1.4.2 (delete, redefine_s_n_locator)
					2.3.7.3 n < m	analog zu 2.3.7.1 (N[m]\\{(import_s|include_s|override_s)_n})
						analog zu 2.1.4.3 (add, (import_s|include_s|override_s)_n)
			2.3.8 M[n] = {include_s_1,..., include_s_n), N[m] = {(import_t|redefine_t|override_t)_1,..., (import_t|redefine_t|override_t)_m}		2.3.8.1 n = m	loop n: include_s_n → (import_t|redefine_t|override_t)_n (analog zu 2.3.4.1)
					2.3.8.2 n > m	analog zu 2.3.8.1 (loop m, M[n]\\{include_s_m})
						analog zu 2.1.4.2 (delete, include_s_n_locator)
					2.3.8.3 n < m	analog zu 2.3.8.1 (N[m]\\{(import_s|redefine_s|override_s)_n})
						analog zu 2.1.4.3 (add, (import_s|redefine_s|override_s)_n)
			2.3.9 M[n] = {override_s_1,..., override_s_n), N[m] = {(import_t|include_t|redefine_t)_1,..., (import_t|include_t|redefine_t)_m}		2.3.9.1 n = m	loop n: override_s_n → (import_t|include_t|redefine_t)_n (analog zu 2.3.5.1)
					2.3.9.2 n > m	analog zu 2.3.9.1 (loop m, M[n]\\{override_s_m})
						analog zu 2.1.4.2 (delete, override_s_n_locator)
					2.3.9.3 n < m	analog zu 2.3.9.1 (N[m]\\{(import_s|include_s|redefine_s)_n})
						analog zu 2.1.4.3 (add, (import_s|include_s|redefine_s)_n)
3. xs:annotation	3.1 V_s ↔ V_t	1:1	3.1.1 appinfo_s != appinfo_t			update annotation at '*annotation_s_locator*' change appinfo '*appinfo_t*';
			3.1.2 documentation_s != documentation_t			update annotation at '*annotation_s_locator*' change documentation '*documentation_t*';
			3.1.3 id_s != id_t			update annotation at '*annotation_s_locator*' change id '*id_t*';
			3.1.4 locator_s != locator_t			update annotation at '*annotation_s_locator*' change move into '*annotation_t_locator*';

Abbildung C.1: ELaX Übersetzungsregeln Teil 1/5 abgeleitet aus [112] und [109]

		n:m	3.1.5 M[n] = {annotation_s_1,..., annotation_s_n}, N[m] = {annotation_t_1,..., annotation_t_m}	3.1.5 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	3.1.5.1 n = m	loop n: annotation_s_n ↔ annotation_t_n (analog zu 3.1.1-3.1.4)
					3.1.5.2 n > m	analog zu 3.1.5.1 (P[m]!=∅, loop m, O[n]\(annotation_s_m))
						loop n: delete annotation at '*annotation_s_n_locator*';
					3.1.5.3 n < m	analog zu 3.1.5.1 (O[n]!=∅, P[m]\(annotation_t_n))
						loop n: add annotation appinfo '*appinfo_t_n*' documentation '*documentation_t_n*' id '*id_t_n*' in '*annotation_t_n_locator*';
	3.2 ¬V_s ← V_t	1:1	3.2.1 ¬appinfo_s AND appinfo_t			analog zu 3.1.1
			3.2.2 ¬documentation_s AND documentation_t			analog zu 3.1.2
			3.2.3 ¬id_s AND id_t			analog zu 3.1.3
		n:m	3.2.4 M[n] = ∅, N[m] = {annotation_t_1,..., annotation_t_m}		3.2.4.1 n < m	analog zu 3.1.5.3 (add)
	3.3 V_s → ¬V_t	1:1	3.3.1 appinfo_s AND ¬appinfo_t			update annotation at '*annotation_s_locator*' change appinfo '';
			3.3.2 documentation_s AND ¬documentation_t			update annotation at '*annotation_s_locator*' change documentation '';
			3.3.3 id_s AND ¬id_t			update annotation at '*annotation_s_locator*' change id '';
		n:m	3.3.4 M[n] = {annotation_s_1,..., annotation_s_n}, N[m] = ∅		3.3.4.1 n > m	analog zu 3.1.5.2 (delete)
4. xs:simpleType	4.1 V_s ↔ V_t	1:1	4.1.1 name_s != name_t			update simpletype name '*name_s*' change name '*name_t*';
			4.1.2 list_s != list_t			update simpletype name '*simpletype_s_name*' change mode list '*list_t_itemType*';
			4.1.3 M[n] = {union_s_1,..., union_s_n}, N[m] = {union_t_1,..., union_t_n}	4.1.3 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]		O[n]!=∅: update simpletype name '*simpletype_s_name*' change mode union *loop n:* remove '*union_s_n_memberTypes*';
						P[m]!=∅: update simpletype name '*simpletype_s_name*' change mode union *loop n:* insert '*union_t_m_memberTypes*';
			4.1.4 restriction_s != restriction_t	4.1.4.1 base_s != base_t		update simpletype name '*simpletype_s_name*' change mode restriction of '*base_t*';
				4.1.4.2 M[n] = {facet_s_1,..., facet_s_n}, N[m] = {facet_t_1,..., facet_t_n}	4.1.4.2 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	4.1.4.2.1 n = m	update simpletype name '*simpletype_s_name*' change mode restriction of '*base_s*' *loop n:* modify '*facet_t_n*' at '*facet_s_n_locator*';
						analog zu 4.1.4.2.1 (P[m]!=∅, loop m, O[n]\(facet_s_m))
					4.1.4.2.2 n > m	update simpletype name '*simpletype_s_name*' change mode restriction of '*base_s*' *loop n:* remove '*facet_s_n*' at '*facet_s_n_locator*';
						analog zu 4.1.4.2.1 (O[n]!=∅, P[m]\(facet_t_n))
					4.1.4.2.3 n < m	update simpletype name '*simpletype_s_name*' change mode restriction of '*base_s*' *loop n:* insert '*facet_t_n*' at '*facet_t_n_locator*';
			4.1.5 final_s != final_t			update simpletype name '*simpletype_s_name*' change final '*final_t*';
			4.1.6 id_s != id_t			update simpletype name '*simpletype_s_name*' change id '*id_t*';
		n:m	4.1.7 M[n] = {simpletype_s_1,..., simpletype_s_n}, N[m] = {simpletype_t_1,..., simpletype_t_m}	4.1.7 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	4.1.7.1 n = m	loop n: simpletype_s_n ↔ simpletype_t_n (analog zu 4.1.1-4.1.6)
					4.1.7.2 n > m	analog zu 4.1.7.1 (P[m]!=∅, loop m, O[n]\(simpletype_s_m))
						loop n: delete simpletype name '*simpletype_s_n_name*';
					4.1.7.3 n < m	analog zu 4.1.7.1 (O[n]!=∅, P[m]\(simpletype_t_n))
						loop n: add simpletype name '*simpletype_t_n_name*' mode '*simpletype_t_n_mode*' final = *final_t_n* id = '*id_t_n*';
	4.2 ¬V_s ← V_t	1:1	4.2.1 ¬final_s AND final_t			analog zu 4.1.5
			4.2.2 ¬id_s AND id_t			analog zu 4.1.6
			4.2.3 (list_s|union_s) AND restriction_t	4.2.3.1 Content(list_s|union_s) = Content(restriction_t)		update simpletype name '*simpletype_s_name*' change mode restriction of '*restriction_t_base*' (*loop n:* insert '*restriction_t_n_facet*' at '*restriction_t_n_locator*');
				4.2.3.2 Range(list_s|union_s)=Range(restriction_t)		
			4.2.4 (list_s|restriction_s) AND union_t	4.2.4.1 Content(list_s|restriction_s) = Content(union_t)		update simpletype name '*simpletype_s_name*' change mode union (*loop n:* insert '*union_t_n_memberTypes*');
				4.2.4.2 Range(list_s|restriction_s)=Range(union_t)		
			4.2.5 (union_s|restriction_s) AND list_t	4.2.5.1 Content(union_s|restriction_s) = Content(list_t)		update simpletype name '*simpletype_s_name*' change mode list '*list_t_itemType*';
				4.2.5.2 Range(union_s|restriction_s)=Range(list_t)		
		n:m	4.2.6 M[n] = {(list_s|union_s)_1,..., (list_s|union_s)_n}, N[m] = {restriction_t_1,..., restriction_t_m}		4.2.6.1 n = m	loop n: (list_s|union_s)_n ← restriction_t_n (analog zu 4.2.3)
					4.2.6.2 n > m	analog zu 4.2.6.1 (loop m, M[n]\((list_s|union_s)_m))
						analog zu 4.1.7.2 (delete)
					4.2.6.3 n < m	analog zu 4.2.6.1 (N[m]\(restriction_t_n))
						analog zu 4.1.7.3 (add, mode restriction of '*restriction_t_n_base*' with '*restriction_t_n_facet*')
			4.2.7 M[n] = {(list_s|restriction_s)_1,..., (list_s|restriction_s)_n}, N[m] = {union_t_1,..., union_t_m}		4.2.7.1 n = m	loop n: (list_s|restriction_s)_n ← union_t_n (analog zu 4.2.4)
					4.2.7.2 n > m	analog zu 4.2.7.1 (loop m, M[n]\((list_s|restriction_s)_m))
						analog zu 4.1.7.2 (delete)
					4.2.7.3 n < m	analog zu 4.2.7.1 (N[m]\(union_t_n))
						analog zu 4.1.7.3 (add, mode union '*union_t_n*_memberTypes')
			4.2.8 M[n] = {(union_s|restriction_s)_1,..., (union_s|restriction_s)_n}, N[m] = {list_t_1,..., list_t_m}		4.2.8.1 n = m	loop n: (union_s|restriction_s)_n ← list_t_n (analog zu 4.2.5)
					4.2.8.2 n > m	analog zu 4.2.8.1 (loop m, M[n]\((union_s|restriction_s)_m))
						analog zu 4.1.7.2 (delete)
					4.2.8.3 n < m	analog zu 4.2.8.1 (N[m]\(list_t_n))
						analog zu 4.1.7.3 (add, mode list '*list_t_n*_itemType')
	4.3 V_s → ¬V_t	1:1	4.3.1 final_s AND ¬final_t			update simpletype name '*simpletype_s_name*' change final '';
			4.3.2 id_s AND ¬id_t			update simpletype name '*simpletype_s_name*' change id '';
			4.3.3 restriction_s AND (list_t|union_t)	4.3.3.1 Content(restriction_s) = Content(list_t|union_t)		analog zu 4.2.4 und 4.2.5
				4.3.3.2 Range(restriction_s)=Range(list_t|union_t)		
			4.3.4 union_s AND (list_t|restriction_t)	4.3.4.1 Content(union_s) = Content(list_t|restriction_t)		analog zu 4.2.3 und 4.2.5
				4.3.4.2 Range(union_s)=Range(list_t|restriction_t)		
			4.3.5 list_s AND (union_t|restriction_t)	4.3.5.1 Content(list_s) = Content(union_t|restriction_t)		analog zu 4.2.3 und 4.2.4
				4.3.5.2 Range(list_s)=Range(union_t|restriction_t)		
		n:m	4.3.6 M[n] = {restriction_s_1,..., restriction_s_n}, N[m] = {(list_t|union_t)_1,..., (list_t|union_t)_m}		4.3.6.1 n = m	loop n: restriction_s_n → (list_t|union_t)_n (analog zu 4.3.3)
					4.3.6.2 n > m	analog zu 4.3.6.1 (loop m, M[n]\(restriction_s_m))
						analog zu 4.1.7.2 (delete)
					4.3.6.3 n < m	analog zu 4.3.6.1 (N[m]\(list_t|union_t)_n))
						analog zu 4.1.7.3 (add, mode ((list '*list_t_n*_itemType')|(union '*union_t_n*_memberTypes')))
			4.3.7 M[n] = {union_s_1,..., union_s_n}, N[m] = {(list_t|restriction_t)_1,..., (list_t|restriction_t)_m}		4.3.7.1 n = m	loop n: union_s_n → (list_t|restriction_t)_n (analog zu 4.3.4)
					4.3.7.2 n > m	analog zu 4.3.7.1 (loop m, M[n]\(union_s_m))
						analog zu 4.1.7.2 (delete)
					4.3.7.3 n < m	analog zu 4.3.7.1 (N[m]\((list_t|restriction_t)_n))
						analog zu 4.1.7.3 (add, mode ((list '*list_t_n*_itemType')|(restriction of '*restriction_t_n_base*' with '*restriction_t_n_facet*')))
			4.3.8 M[n] = {list_s_1,..., list_s_n}, N[m] = {(union_t|restriction_t)_1,..., (union_t|restriction_t)_m}		4.3.8.1 n = m	loop n: list_s_n → (union_t|restriction_t)_n (analog zu 4.3.5)
					4.3.8.2 n > m	analog zu 4.3.8.1 (loop m, M[n]\(list_s_m))
						analog zu 4.1.7.2 (delete)
					4.3.8.3 n < m	analog zu 4.3.8.1 (N[m]\((union_t|restriction_t)_n))
						analog zu 4.1.7.3 (add, mode ((union '*union_t_n*_memberTypes')|(restriction of '*restriction_t_n_base*' with '*restriction_t_n_facet*')))
5. xs:complexType	5.1 V_s ↔ V_t	1:1	5.1.1 name_s != name_t			update complextype name '*name_s*' change name '*name_t*';
			5.1.2 mixed_s != mixed_t			update complextype name '*complextype_s_name*' change mixed '*mixed_t*';
			5.1.3 final_s != final_t			update complextype name '*complextype_s_name*' change final '*final_t*';
			5.1.4 (simple|complex)Content_s_base != (simple|complex)Content_t_base			update complextype name '*complextype_s_name*' change mode (extension_cc|extension_sc|restriction_cc|restriction_sc) with base '*simpleContent_t_base*';
			5.1.5 restriction_sc_s != restriction_sc_t	5.1.5.1 M[n] = {facet_s_1,..., facet_s_n}, N[m] = {facet_t_1,..., facet_t_n}	5.1.5.1.1 n = m	update complextype name '*complextype_s_name*' change mode restriction_sc *loop n:* modify '*facet_t_n*' at '*facet_s_n_locator*' with base '*restriction_sc_s_base*';
						analog zu 5.1.5.1.1 (P[m]!=∅, loop m, O[n]\(facet_s_m))
					5.1.5.1.2 n > m	update complextype name '*complextype_s_name*' change mode restriction_sc *loop n:* remove '*facet_s_n*' at '*facet_s_n_locator*' with base '*restriction_sc_s_base*';
						analog zu 5.1.5.1.1 (O[n]!=∅, P[m]\(facet_t_n))
					5.1.5.1.3 n < m	update complextype name '*complextype_s_name*' change mode restriction_sc *loop n:* insert '*facet_t_n*' at '*facet_t_n_locator*' with base '*restriction_sc_s_base*';
			5.1.6 id_s != id_t			update complextype name '*complextype_s_name*' change id '*id_t*';
			5.1.7 defaultAttributesApply_s != defaultAttributesApply_t			update complextype name '*complextype_s_name*' change defaultattributesapply '*defaultAttributesApply_t*';
			5.1.8 assert_s != assert_t			update complextype name '*complextype_s_name*' change assert '*assert_t*' at locator '*assert_s_locator*';
		n:m	5.1.9 M[n] = {complextype_s_1,..., complextype_s_n}, N[m] = {complextype_t_1,..., complextype_t_m}	5.1.9 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	5.1.9.1 n = m	loop n: complextype_s_n ↔ complextype_t_n (analog zu 5.1.1-5.1.6)
					5.1.9.2 n > m	analog zu 5.1.9.1 (P[m]!=∅, loop m, O[n]\(complextype_s_m))
						loop n: delete complextype name '*complextype_s_n_name*';
					5.1.9.3 n < m	analog zu 5.1.9.1 (O[n]!=∅, P[m]\(complextype_t_n))
						loop n: add complextype name '*complextype_t_n_name*' mixed = '*mixed_t_n*' final = *final_t_n* mode '*complextype_t_n_mode*' id = '*id_t_n*' defaultattributesapply = '*defaultAttributesApply_t_n*' assert='*assert_t_n*';

Abbildung C.2: ELaX Übersetzungsregeln Teil 2/5 abgeleitet aus [112] und [109]

	5.2 ¬V_s ← V_t	1:1	5.2.1 ¬mixed_s AND mixed_t			analog zu 5.1.2
			5.2.2 ¬final_s AND final_t			analog zu 5.1.3
			5.2.3 ¬id_s AND id_t			analog zu 5.1.6
			5.2.4 ¬defaultAttributesApply_s AND defaultAttributesApply_t			analog zu 5.1.7
			5.2.5 ¬assert_s AND assert_t			analog zu 5.1.8
			5.2.6 (extension_cc_s|extension_sc_s| restriction_cc_s) AND restriction_sc_t	5.2.6.1 Content(extension_cc_s|extension_sc_s| restriction_cc_s) = Content(restriction_sc_t)		update complextype name '*complextype_s_name*' change mode restriction_sc (*loop n:* insert '*facet_t_n*' at '*facet_t_n_locator*') with base '*restriction_sc_t_base*';
				5.2.6.2 Range(extension_cc_s|extension_sc_s| restriction_cc_s) = Range(restriction_sc_t)		
			5.2.7 (extension_cc_s|extension_sc_s| restriction_sc_s) AND restriction_cc_t	5.2.7.1 Content(extension_cc_s|extension_sc_s| restriction_sc_s) = Content(restriction_cc_t)		update complextype name '*complextype_s_name*' change mode restriction_cc with base '*restriction_cc_t_base*';
				5.2.7.2 Range(extension_cc_s|extension_sc_s| restriction_sc_s) = Range(restriction_cc_t)		
			5.2.8 (extension_cc_s|restriction_cc_s| restriction_sc_s) AND extension _sc_t	5.2.8.1 Content(extension_cc_s|restriction_cc_s| restriction_sc_s) = Content(extension_sc_t)		update complextype name '*complextype_s_name*' change mode extension_sc with base '*extension_sc_t_base*';
				5.2.8.2 Range(extension_cc_s|restriction_cc_s| restriction_sc_s) = Range(extension_sc_t)		
			5.2.9 (extension_sc_s|restriction_cc_s| restriction_sc_s) AND extension _cc_t	5.2.9.1 Content(extension_sc_s|restriction_cc_s| restriction_sc_s) = Content(extension_cc_t)		update complextype name '*complextype_s_name*' change mode extension_cc with base '*extension_cc_t_base*';
				5.2.9.2 Range(extension_sc_s|restriction_cc_s| restriction_sc_s) = Range(extension_cc_t)		
		n:m	5.2.10 M[n] = {(extension_cc_s|extension_sc_s|restriction_cc_s)_1,..., (extension_cc_s|extension_sc_s|restriction_cc_s)_n}, N[m] = {restriction_sc_t_1,..., restriction_sc_t_m}		5.2.10.1 n = m	loop n: (extension_cc_s|extension_sc_s| restriction_cc_s)_n ← restriction_sc_t_n (analog zu 5.2.6)
					5.2.10.2 n > m	analog zu 5.2.10.1 (loop m, M[n]\\{(extension_cc_s| extension_sc_s|restriction_cc_s)_m})
						analog zu 5.1.9.2 (delete)
					5.2.10.3 n < m	analog zu 5.2.10.1 (N[m]\\(restriction_sc_t_n))
						analog zu 5.1.9.3 (add, mode restriction_sc ('*restriction_sc_t_n_*facet') with base '*restriction_sc_t_n_base*')
			5.2.11 M[n] = {(extension_cc_s|extension_sc_s|restriction_sc_s)_1,..., (extension_cc_s|extension_sc_s|restriction_sc_s)_n}, N[m] = {restriction_cc_t_1,..., restriction_cc_t_m}		5.2.11.1 n = m	loop n: (extension_cc_s|extension_sc_s| restriction_sc_s)_n ← restriction_cc_t_n (analog zu 5.2.7)
					5.2.11.2 n > m	analog zu 5.2.11.1 (loop m, M[n]\\{(extension_cc_s| extension_sc_s|restriction_sc_s)_m})
						analog zu 5.1.9.2 (delete)
					5.2.11.3 n < m	analog zu 5.2.11.1 (N[m]\\(restriction_cc_t_n))
						analog zu 5.1.9.3 (add, mode restriction_cc with base '*restriction_cc_t_n_base*')
			5.2.11 M[n] = {(extension_cc_s|restriction_cc_s|restriction_sc_s)_1,..., (extension_cc_s|restriction_cc_s|restriction_sc_s)_n), N[m] = {extension_sc_t_1,..., extension_sc_t_m}		5.2.11.1 n = m	loop n: (extension_cc_s|restriction_cc_s| restriction_sc_s)_n ← extension_sc_t_n (analog zu 5.2.8)
					5.2.11.2 n > m	analog zu 5.2.11.1 (loop m, M[n]\\{(extension_cc_s| restriction_cc_s|restriction_sc_s)_m})
						analog zu 5.1.9.2 (delete)
					5.2.11.3 n < m	analog zu 5.2.11.1 (N[m]\\(extension_sc_t_n))
						analog zu 5.1.9.3 (add, mode extension_sc with base '*extension_sc_t_n_base*')
			5.2.12 M[n] = {(extension_sc_s|restriction_cc_s|restriction_sc_s)_1,..., (extension_sc_s|restriction_cc_s|restriction_sc_s)_n), N[m] = {extension_cc_t_1,..., extension_cc_t_m}		5.2.12.1 n = m	loop n: (extension_sc_s|restriction_cc_s| restriction_sc_s)_n ← extension_cc_t_n (analog zu 5.2.9)
					5.2.12.2 n > m	analog zu 5.2.12.1 (loop m, M[n]\\{(extension_sc_s| restriction_cc_s|restriction_sc_s)_m})
						analog zu 5.1.9.2 (delete)
					5.2.12.3 n < m	analog zu 5.2.12.1 (N[m]\\(extension_cc_t_n))
						analog zu 5.1.9.3 (add, mode extension_cc with base '*extension_cc_t_n_base*')
	5.3 V_s → ¬V_t	1:1	5.3.1 mixed_s AND ¬mixed_t			update complextype name '*complextype_s_name*' change mixed '';
			5.3.2 final_s AND ¬final_t			update complextype name '*complextype_s_name*' change final '';
			5.3.3 id_s AND ¬id_t			update complextype name '*complextype_s_name*' change id '';
			5.3.4 defaultAttributesApply_s AND ¬defaultAttributesApply_t			update complextype name '*complextype_s_name*' change defaultattributesapply '';
			5.3.5 assert_s AND ¬assert_t			update complextype name '*complextype_s_name*' change assert '' at locator '*assert_s_locator*';
			5.3.6 restriction_sc_s AND (extension_cc_t|extension_sc_t| restriction_cc_t)	5.3.6.1 Content(restriction_sc_s) = Content(extension_cc_t|extension_sc_t|restriction_cc_t)		analog zu 5.2.7, 5.2.8 und 5.2.9
				5.3.6.2 Range(restriction_sc_s) = Range(extension_cc_t|extension_sc_t|restriction_cc_t)		
			5.3.7 restriction_cc_s AND (extension_cc_t|extension_sc_t| restriction_sc_t)	5.3.7.1 Content(restriction_cc_s) = Content(extension_cc_t|extension_sc_t|restriction_sc_t)		analog zu 5.2.6, 5.2.8 und 5.2.9
				5.3.7.2 Range(restriction_cc_s) = Range(extension_cc_t|extension_sc_t|restriction_sc_t)		
			5.3.8 extension _sc_s AND (extension_cc_t|restriction_cc_t| restriction_sc_t)	5.3.8.1 Content(extension _sc_s) = Content(extension_cc_t|restriction_cc_t|restriction_sc_t)		analog zu 5.2.6, 5.2.7 und 5.2.9
				5.3.8.2 Range(extension _sc_s) = Range(extension_cc_t|restriction_cc_t|restriction_sc_t)		
			5.3.9 extension _cc_s AND (extension_sc_t|restriction_cc_t| restriction_sc_t)	5.3.9.1 Content(extension _cc_s) = Content(extension_sc_t|restriction_cc_t|restriction_sc_t)		analog zu 5.2.6, 5.2.7 und 5.2.8
				5.3.9.2 Range(extension _cc_s) = Range(extension_sc_t|restriction_cc_t|restriction_sc_t)		
		n:m	5.3.10 M[n] = {restriction_sc_s_1,..., restriction_sc_s_n), N[m] = {(extension_cc_t|extension_sc_t|restriction_cc_t)_1,..., (extension_cc_t|extension_sc_t|restriction_cc_t)_m}		5.3.10.1 n = m	loop n: restriction_sc_s_n → (extension_cc_t| extension_sc_t|restriction_cc_t)_n (analog zu 5.3.6)
					5.3.10.2 n > m	analog zu 5.3.10.1 (loop m, M[n]\(restriction_sc_s_m))
						analog zu 5.1.9.2 (delete)
					5.3.10.3 n < m	analog zu 5.3.10.1 (N[m]\((extension_cc_t| extension_sc_t|restriction_cc_t)_n))
						analog zu 5.1.9.3 (add, mode ((extension_cc_t| extension_sc_t|restriction_cc_t) with base '*(extension_cc_t|extension_sc_t|restriction_cc_t)_n_base')*)
			5.3.11 M[n] = {restriction_cc_s_1,..., restriction_cc_s_n), N[m] = {(extension_cc_t|extension_sc_t|restriction_sc_t)_1,..., (extension_cc_t|extension_sc_t|restriction_sc_t)_m}		5.3.11.1 n = m	loop n: restriction_cc_s_n → (extension_cc_t|extension_sc_t| restriction_sc_t)_n (analog zu 5.3.7)
					5.3.11.2 n > m	analog zu 5.3.11.1 (loop m, M[n]\(restriction_cc_s_m))
						analog zu 5.1.9.2 (delete)
					5.3.11.3 n < m	analog zu 5.3.11.1 (N[m]\((extension_cc_t|extension_sc_t| restriction_sc_t)_n))
						analog zu 5.1.9.3 (add, mode ((extension_cc_t| extension_sc_t|restriction_sc_t('*restriction_sc_t_n_*facet')) with base '*(extension_cc_t|extension_sc_t|restriction_sc_t)_n_base')*)
			5.3.11 M[n] = {extension_sc_s_1,..., extension_sc_s_n), N[m] = {(extension_cc_t|restriction_cc_t|restriction_sc_t)_1,..., (extension_cc_t|restriction_cc_t|restriction_sc_t)_m}		5.3.11.1 n = m	loop n: extension_sc_s_n → (extension_cc_t|restriction_cc_t| restriction_sc_t)_n (analog zu 5.3.8)
					5.3.11.2 n > m	analog zu 5.3.11.1 (loop m, M[n]\(extension_sc_s_m))
						analog zu 5.1.9.2 (delete)
					5.3.11.3 n < m	analog zu 5.3.11.1 (N[m]\((extension_cc_t|restriction_cc_t| restriction_sc_t)_n))
						analog zu 5.1.9.3 (add, mode ((extension_cc_t| restriction_cc_t|restriction_sc_t('*restriction_sc_t_n_*facet')) with base '*(extension_cc_t|restriction_cc_t|restriction_sc_t)_n_base')*)
			5.3.12 M[n] = {extension_cc_s_1,..., extension_cc_s_n), N[m] = {(extension_sc_t|restriction_cc_t|restriction_sc_t)_1,..., (extension_sc_t|restriction_cc_t|restriction_sc_t)_m}		5.3.12.1 n = m	loop n: extension_cc_s_n → (extension_sc_t|restriction_cc_t| restriction_sc_t)_n (analog zu 5.3.9)
					5.3.12.2 n > m	analog zu 5.3.12.1 (loop m, M[n]\(extension_cc_s_m))
						analog zu 5.1.9.2 (delete)
					5.3.12.3 n < m	analog zu 5.3.12.1 (N[m]\((extension_sc_t|restriction_cc_t| restriction_sc_t)_n))
						analog zu 5.1.9.3 (add, mode ((extension_sc_t| restriction_cc_t|restriction_sc_t('*restriction_sc_t_n_*facet')) with base '*(extension_sc_t|restriction_cc_t|restriction_sc_t)_n_base')*)
6. group: xs:sequence, xs:choice und xs:all	6.1 V_s ↔ V_t	1:1	6.1.1 minOccurs_s != minOccurs_t			update group at '*group_s_locator*' change minoccurs '*minOccurs_t*';
			6.1.2 maxOccurs_s != maxOccurs_t			update group at '*group_s_locator*' change maxoccurs '*maxOccurs_t*';
			6.1.3 choice_s_groupDefault != choice_t_groupDefault			update group at '*group_s_locator*' change mode choice with '*choice_t_groupDefault*';
			6.1.4 id_s != id_t			update group at '*group_s_locator*' change id '*id_t*';
		n:m	6.1.5 M[n] = {group_s_1,..., group_s_n), N[m] = {group_t_1,..., group_t_m}	6.1.5 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	6.1.5.1 n = m	loop n: group_s_n ↔ group_t_n (analog zu 6.1.1-6.1.4)
					6.1.5.2 n > m	analog zu 6.1.5.1 (P[m]!=∅, loop m, O[n]\(group_s_m))
						loop n: delete group at '*group_s_n_locator*';
					6.1.5.3 n < m	analog zu 6.1.5.1 (O[n]!=∅, P[m]\(group_t_n))
						loop n: add group mode (sequence|choice|all) minoccurs '*minoccurs_t_n*' maxoccurs '*maxoccurs_t_n*' id '*id_t_n*' in '*group_t_n_locator*';
	6.2 ¬V_s ← V_t	1:1	6.2.1 ¬choice_s_groupDefault != choice_t_groupDefault			analog zu 6.1.3
			6.2.2 ¬id_s AND id_t			analog zu 6.1.4
			6.2.3 (sequence_s|choice_s) AND all_t	6.2.3.1 Content(sequence_s|choice_s) = Content(all_t)		update group at '*group_s_locator*' change mode all;
				6.2.3.2 Range(sequence_s|choice_s)=Range(all_t)		
			6.2.4 (sequence_s|all_s) AND choice_t	6.2.4.1 Content(sequence_s|all_s) = Content(choice_t)		update group at '*group_s_locator*' change mode choice;
				6.2.4.2 Range(sequence_s|all_s)=Range(choice_t)		
			6.2.5 (choice_s|all_s) AND sequence_t	6.2.5.1 Content(choice_s|all_s) = Content(sequence_t)		update group at '*group_s_locator*' change mode sequence;
				6.2.5.2 Range(choice_s|all_s)=Range(sequence_t)		
		n:m	6.2.6 M[n] = {(sequence_s|choice_s)_1,..., (sequence_s|choice_s)_n), N[m] = {all_t_1,..., all_t_m}		6.2.6.1 n = m	loop n: (sequence_s|choice_s)_n ← all_t_n (analog zu 6.2.3)
					6.2.6.2 n > m	analog zu 6.2.6.1 (loop m, M[n]\\{(sequence_s|choice_s)_m})
						analog zu 6.1.5.2 (delete)
					6.2.6.3 n < m	analog zu 6.2.6.1 (N[m]\(all_t_n))
						analog zu 6.1.5.3 (add, mode all)
			6.2.7 M[n] = {(sequence_s|all_s)_1,..., (sequence_s|all_s)_n), N[m] = {choice_t_1,..., choice_t_m}		6.2.7.1 n = m	loop n: (sequence_s|all_s)_n ← choice_t_n (analog zu 6.2.4)
					6.2.7.2 n > m	analog zu 6.2.7.1 (loop m, M[n]\\{(sequence_s|all_s)_m})

Abbildung C.3: ELaX Übersetzungsregeln Teil 3/5 abgeleitet aus [112] und [109]

					analog zu 6.1.5.2 (delete)
				6.2.7.3 n < m	analog zu 6.2.7.1 (N[m]\(choice_t_n))
					analog zu 6.1.5.3 (add, mode choice)
			6.2.8 M[n] = {(choice_s|all_s)_1,..., (choice_s|all_s)_n}, N[m] = {sequence_t_1,..., sequence_t_m}	6.2.8.1 n = m	loop n: (choice_s|all_s)_n ← sequence_t_n (analog zu 6.2.5)
				6.2.8.2 n > m	analog zu 6.2.8.1 (loop m, M[n]\(choice_s|all_s)_m))
					analog zu 6.1.5.2 (delete)
				6.2.8.3 n < m	analog zu 6.2.8.1 (N[m]\(sequence_t_n))
					analog zu 6.1.5.3 (add, mode sequence)
	6.3 V_s → ¬V_t		6.3.1 choice_s_groupDefault != ¬choice_t_groupDefault		update group at '*group_s_locator*' change mode choice with '';
			6.3.2 id_s AND ¬id_t		update group at '*group_s_locator*' change id '';
		1:1	6.3.3 all_s AND (sequence_t|choice_t)	6.3.3.1 Content(all_s) = Content(sequence_t|choice_t)	analog zu 6.2.4 und 6.2.5
				6.3.3.2 Range(all_s)=Range(sequence_t|choice_t)	
			6.3.4 choice_s AND (sequence_t|all_t)	6.2.4.1 Content(choice_s) = Content(sequence_t|all_t)	analog zu 6.2.3 und 6.2.5
				6.2.4.2 Range(choice_s)=Range(sequence_t|all_t)	
			6.3.5 sequence_s AND (choice_t|all_t)	6.2.5.1 Content(sequence_s) = Content(choice_t|all_t)	analog zu 6.2.3 und 6.2.4
				6.2.5.2 Range(sequence_s)=Range(choice_t|all_t)	
		n:m	6.3.6 M[n] = {all_s_1,..., all_s_n}, N[m] = {(sequence_t|choice_t)_1,..., (sequence_t|choice_t)_m}	6.3.6.1 n = m	loop n: all_s_n → (sequence_t|choice_t)_n (analog zu 6.3.3)
				6.3.6.2 n > m	analog zu 6.3.6.1 (loop m, M[n]\(all_s_m))
					analog zu 6.1.5.2 (delete)
				6.3.6.3 n < m	analog zu 6.3.6.1 (N[m]\(sequence_t|choice_t)_n))
					analog zu 6.1.5.3 (add, mode (sequence|choice))
			6.3.7 M[n] = {choice_s_1,..., choice_s_n), N[m] = {(sequence_t|all_t)_1,..., (sequence_t|all_t)_m}	6.3.7.1 n = m	loop n: choice_s_n → (sequence_t|all_t)_n (analog zu 6.3.4)
				6.3.7.2 n > m	analog zu 6.3.7.1 (loop m, M[n]\(choice_s_m))
					analog zu 6.1.5.2 (delete)
				6.3.7.3 n < m	analog zu 6.3.7.1 (N[m]\(sequence_t|all_t)_n))
					analog zu 6.1.5.3 (add, mode (sequence|all))
			6.3.8 M[n] = {sequence_s_1,.., sequence_s_n), N[m] = {(choice_t|all_t)_1,..., (choice_t|all_t)_m}	6.3.8.1 n = m	loop n: sequence_s_n → (choice_t|all_t)_n (analog zu 6.3.5)
				6.3.8.2 n > m	analog zu 6.3.8.1 (loop m, M[n]\(sequence_s_m))
					analog zu 6.1.5.2 (delete)
				6.3.8.3 n < m	analog zu 6.3.8.1 (N[m]\(choice_t|all_t)_n))
					analog zu 6.1.5.3 (add, mode (choice|all))
7. xs:element: global, reference und any	7.1 V_s ↔ V_t		7.1.1 global_s_name != global_t_name		update element name '*global_s_name*' change name '*global_t_name*';
			7.1.2 global_s_type != global_t_type		update element name '*global_s_name*' change type '*global_t_type*';
			7.1.3 global_s_default != global_t_default		update element name '*global_s_name*' change default '*global_t_default*';
			7.1.4 global_s_fixed != global_t_fixed		update element name '*global_s_name*' change fixed '*global_t_fixed*';
			7.1.5 global_s_final != global_t_final		update element name '*global_s_name*' change final '*global_t_final*';
			7.1.6 global_s_nillable != global_t_nillable		update element name '*global_s_name*' change nillable '*global_t_nillable*';
			7.1.7 reference_s_ref != reference_t_ref		update elementref '*reference_s_ref*' (at '*reference_s_locator*'| '*reference_s_refposition*') change ref '*reference_t_ref*';
			7.1.8 (reference_s|any_s)_minOccurs != (reference_t|any_t)_minOccurs		update elementref '*reference_s_ref*' (at '*reference_s_locator*'| '*reference_s_refposition*') change minoccurs '*reference_t_minOccurs*';
					update any at '*any_s_locator*' change minoccurs '*any_t_minOccurs*';
			7.1.9 (reference_s|any_s)_maxOccurs_s != (reference_t|any_t)_maxOccurs		update elementref '*reference_s_ref*' (at '*reference_s_locator*'| '*reference_s_refposition*') change maxoccurs '*reference_t_maxOccurs*';
					update any at '*any_s_locator*' change maxoccurs '*any_t_maxOccurs*';
			7.1.10 any_s_notQName != any_t_notQName		update any at '*any_s_locator*' change not '*any_t_notQName*';
			7.1.11 any_s_namespace != any_t_namespace		update any at '*any_s_locator*' change namespace '*any_t_namespace*';
			7.1.12 any_s_notNamespace != any_t_notNamespace		update any at '*any_s_locator*' change namespace not '*any_t_notNamespace*';
			7.1.13 any_s_processContents != any_t_processContents		update any at '*any_s_locator*' change processcontent '*any_t_processContents*';
			7.1.14 (global_s|reference_s|any_s)_id != (global_t|reference_t|any_t)_id		update element name '*global_s_name*' change id '*global_t_id*';
					update elementref '*reference_s_ref*' (at '*reference_s_locator*'| '*reference_s_refposition*') change id '*reference_t_id*';
					update any at '*any_s_locator*' change id '*any_t_id*';
			7.1.15 reference_s_position != reference_t_position		update elementref '*reference_s_ref*' (at '*reference_s_locator*'| '*reference_s_refposition*') move to '*reference_t_position*';
		n:m	7.1.16 M[n] = {element_s_1,..., element_s_n), N[m] = {element_t_1,..., element_t_m}	7.1.16 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	7.1.16.1 n = m	loop n: element_s_n ↔ element_t_n (analog zu 7.1.1-7.1.15)
					7.1.16.2 n > m	analog zu 7.1.16.1 (P[m]!=∅, loop m, O[n]\(element_s_m))
						loop n: delete element name '*global_s_n_name*';
						loop n: delete elementref '*reference_s_n_ref*' (at '*reference_s_n_locator*'|'*reference_s_n_refposition*');
						loop n: delete any at '*any_s_n_locator*';
					7.1.16.3 n < m	analog zu 7.1.16.1 (O[n]!=∅, P[m]\(element_t_n))
						loop n: add element name '*global_t_n_name*' type '*global_t_n_type*' (default '*global_t_n_default*' | fixed '*global_t_n_fixed*') final '*global_t_n_final*' nillable '*global_t_n_nillable*' id '*global_t_n_id*';
						loop n: add elementref '*reference_t_n_ref*' minoccurs '*reference_t_n_minOccurs*' maxoccurs '*reference_t_n_maxOccurs*' id '*reference_t_n_id*' '*reference_t_n_position*;
						loop n: add any not '*any_t_n_notQName*' namespace '*any_t_n_namespace*' not '*any_t_n_notNamespace*' processcontent '*any_t_n_processContents*' minoccurs '*any_t_n_minOccurs*' maxoccurs '*any_t_n_maxOccurs*' id '*any_t_n_id*' in '*any_t_n_locator*';
	7.2 ¬V_s ← V_t		7.2.1 ¬global_s_default AND global_t_default		analog zu 7.1.3
			7.2.2 ¬global_s_fixed AND global_t_fixed		analog zu 7.1.4
		1:1	7.2.3 ¬global_s_final AND global_t_final		analog zu 7.1.5
			7.2.4 ¬any_s_notQName AND any_t_notQName		analog zu 7.1.10
			7.2.5 ¬any_s_notNamespace AND any_t_notNamespace		analog zu 7.1.12
			7.2.6 ¬(global_s|reference_s|any_s)_id AND (global_t|reference_t|any_t)_id		analog zu 7.1.14
		n:m	7.2.7 M[n] = ∅, N[m] = {(global_t|reference_t|any_t)_1,..., (global_t|reference_t|any_t)_m}	7.2.7.1 n < m	analog zu 7.1.16.3 (add)
	7.3 V_s → ¬V_t		7.3.1 global_s_default AND ¬global_t_default		update element name '*global_s_name*' change default '';
			7.3.2 global_s_fixed AND ¬global_t_fixed		update element name '*global_s_name*' change fixed '';
			7.3.3 global_s_final AND ¬global_t_final		update element name '*global_s_name*' change final '';
		1:1	7.3.4 any_s_notQName AND ¬any_t_notQName		update any at '*any_s_locator*' change not '';
			7.3.5 any_s_notNamespace AND ¬any_t_notNamespace		update any at '*any_s_locator*' change namespace not '';
			7.3.6 (global_s|reference_s|any_s)_id AND ¬(global_t|reference_t|any_t)_id		update element name '*global_s_name*' change id '';
					update elementref '*reference_s_ref*' (at '*reference_s_locator*'|'*reference_s_refposition*') change id '';
					update any at '*any_s_locator*' change id '';
		n:m	7.3.7 M[n] = {(global_s|reference_s|any_s)_1,..., (global_s|reference_s|any_s)_n), N[m] = ∅	7.3.7.1 n > m	analog zu 7.1.16.2 (delete)
8. xs:attributeGroup und xs:attribute: globalAG, referenceAG, globalA, referenceA und any	8.1 V_s ↔ V_t		8.1.1 (globalAG_s|globalA_s)_name != (globalAG_t|globalA_t)_name		update (attributegroup|attribute) name '*(globalAG_s|globalA_s)_name*' change name '*(globalAG_t|globalA_t)_name*';
			8.1.2 M[n] = {globalAG_Content_s_1,..., globalAG_Content_s_n), N[m] = {globalAG_Content_t_1,..., globalAG_Content_t_n}	8.1.2 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	O[n]!=∅: update attributegroup name '*globalAG_s_name*' change (*loop n:* delete <delattributeref>_s_n) (delete <delattributewildcard>_s);
					P[m]!=∅: update attributegroup name '*globalAG_s_name*' change (*loop n:* add <addattributeref>_t_n) (add <addattributewildcard>_t);
			8.1.3 globalA_s_type != globalA_t_type		update attribute name '*globalA_s_name*' change type '*globalA_t_type*';
			8.1.4 (globalA_s|referenceA_s)_default != (globalA_t|referenceA_t)_default		update attribute name '*globalA_s_name*' change default '*globalA_t_default*';
					update attributeref '*referenceA_s_ref*' at '*referenceA_s_locator*' change default '*referenceA_t_default*';
			8.1.5 (globalA_s|referenceA_s)_fixed != (globalA_t|referenceA_t)_fixed		update attribute name '*globalA_s_name*' change fixed '*globalA_t_fixed*';
					update attributeref '*referenceA_s_ref*' at '*referenceA_s_locator*' change fixed '*referenceA_t_fixed*';
		1:1	8.1.6 globalA_s_inheritable != globalA_t_inheritable		update attribute name '*globalA_s_name*' change inheritable '*globalA_t_inheritable*';
			8.1.7 (referenceAG_s|referenceA_s)_ref != (referenceAG_t|referenceA_t)_ref		update (attributegroupref|attributeref) '*(referenceAG_s| referenceA_s)_ref*' at '*(referenceAG_s|referenceA_s)_locator*' change ref '*(referenceAG_t|referenceA_t)_ref*';
			8.1.8 referenceA_s_use != referenceA_t_use		update attributeref '*referenceA_s_ref*' at '*referenceA_s_locator*' change use '*referenceA_t_use*';
			8.1.9 any_s_notQName != any_t_notQName		update anyattribute at '*any_s_locator*' change not '*any_t_notQName*';
			8.1.10 any_s_namespace != any_t_namespace		update anyattribute at '*any_s_locator*' change namespace '*any_t_namespace*';
			8.1.11 any_s_notNamespace != any_t_notNamespace		update anyattribute at '*any_s_locator*' change namespace not '*any_t_notNamespace*';
			8.1.12 any_s_processContents != any_t_processContents		update anyattribute at '*any_s_locator*' change processcontent '*any_t_processContents*';
			8.1.13 (globalAG_s|referenceAG_s|globalA_s|referenceA_s|any_s)_id != (globalAG_t|referenceAG_t|globalA_t|referenceA_t|any_t)_id		update (attributegroup|attribute) name '*(globalAG_s| globalA_s)_name*' change id '*(globalAG_t|globalA_t)_id*;
					update (attributegroupref|attributeref) '*(referenceAG_s| referenceA_s)_ref*' at '*(referenceAG_s|referenceA_s)_locator*' change id '*(referenceAG_t|referenceA_t)_id*';
					update anyattribute at '*any_s_locator*' change id '*any_t_id*';

Abbildung C.4: ELaX Übersetzungsregeln Teil 4/5 abgeleitet aus [112] und [109]

			8.1.14 (referenceAG_s|referenceA_s)_locator != (referenceAG_t|referenceA_t)_locator			update (attributegroupref|attributeref) '*(referenceAG_s| referenceA_s)_ref*' at '*(referenceAG_s|referenceA_s)_locator* move into '*(referenceAG_t|referenceA_t)_locator*';
		n:m	8.1.15 M[n] = {(attributeGroup_s|attribute_s)_1,..., (attributeGroup_s|attribute_s)_n), N[m] = {(attributeGroup_t|attribute_t)_1,..., (attributeGroup_t|attribute_t)_m}	8.1.15 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	8.1.15.1 n = m	loop n: (attributeGroup_s|attribute_s)_n ↔ (attributeGroup_t|attribute_t)_n (analog zu 8.1.1-8.1.14)
					8.1.15.2 n > m	analog zu 8.1.15.1 (P[m]!=∅, loop m, O[n]\\(attributeGroup_s|attribute_s)_m))
						loop n: delete (attributegroup|attribute) name '*(globalAG_s|globalA_s)_n_name*';
						loop n: delete (attributegroupref|attribute) '*(referenceAG_s| referenceA_s)_ref*' at '*(referenceAG_s|referenceA_s)_locator*';
						loop n: delete anyattribute at '*any_s_n_locator*';
					8.1.15.3 n < m	analog zu 8.1.15.1 (O[n]!=∅, P[m]\\(attributegroupref_t|attribute_t)_n))
						loop n: add attributegroup name '*globalAG_t_n_name*' id '*globalAG_t_n_id*' with *loop x* <addattributeref>_x <addattributewildcard>; x>0
						loop n: add attributegroupref '*referenceAG_t_n_name*' id '*referenceAG_t_n_id*' in '*referenceAG_t_n_locator*';
						loop n: add attribute name '*globalA_t_n_name*' type '*globalA_t_n_type*' (default '*globalA_t_n_default*' | fixed '*globalA_t_n_fixed*') inheritable '*globalA_t_n_inheritable*' id '*globalA_t_n_id*';
						loop n: add attributeref name '*referenceA_t_n_name*' (default '*referenceA_t_n_default*' | fixed '*referenceA_t_n_fixed*') use '*referenceA_t_n_use*' id '*referenceA_t_n_id*' in '*referenceA_t_n_locator*';
						loop n: add anyattribute not '*any_t_n_notQName*' namespace '*any_t_n_namespace*' not '*any_t_n_notNamespace*' processcontent '*any_t_n_processContents*' id '*any_t_n_id*' in '*any_t_n_locator*';
	8.2 ¬V_s ← V_t		8.2.1 ¬globalAG_s_any AND globalAG_t_any			update attributegroup name '*globalAG_s_name*' change add <addattributewildcard>_globalAG_t_any;
		1:1	8.2.2 ¬(globalA_s|referenceA_s)_default AND (globalA_t|referenceA_t)_default			analog zu 8.1.4
			8.2.3 ¬(globalA_s|referenceA_s)_fixed AND (globalA_t|referenceA_t)_fixed			analog zu 8.1.5
			8.2.4 ¬any_s_notQName AND any_t_notQName			analog zu 8.1.9
			8.2.5 ¬any_s_namespace AND any_t_namespace			analog zu 8.1.10
			8.2.6 ¬any_s_notNamespace AND any_t_notNamespace			analog zu 8.1.11
			8.2.7 ¬(globalAG_s|referenceAG_s|globalA_s|referenceA_s|any_s)_id AND (globalAG_t|referenceAG_t|globalA_t|referenceA_t|any_t)_id			analog zu 8.1.13
		n:m	8.2.8 M[n] = ∅, N[m] = {(attributeGroup_t|attribute_t)_1,..., (attributeGroup_t|attribute_t)_m}		8.2.8.1 n < m	analog zu 8.1.15.3 (add)
	8.3 V_s → ¬V_t		8.3.1 globalAG_s_any AND ¬globalAG_t_any			update attributegroup name '*globalAG_s_name*' change delete <delattributewildcard>_globalAG_t_any;
		1:1	8.3.2 (globalA_s|referenceA_s)_default AND ¬(globalA_t|referenceA_t)_default			update attribute name '*globalA_s_name*' change default '';
						update attributeref '*referenceA_s_ref*' at '*referenceA_s_locator*' change default '';
			8.3.3 (globalA_s|referenceA_s)_fixed AND ¬(globalA_t|referenceA_t)_fixed			update attribute name '*globalA_s_name*' change fixed '';
						update attributeref '*referenceA_s_ref*' at '*referenceA_s_locator*' change fixed '';
			8.3.4 any_s_notQName AND ¬any_t_notQName			update anyattribute at '*any_s_locator*' change not '';
			8.3.5 any_s_namespace AND ¬any_t_namespace			update anyattribute at '*any_s_locator*' change namespace '';
			8.3.6 any_s_notNamespace AND ¬any_t_notNamespace			update anyattribute at '*any_s_locator*' change namespace not '';
			8.3.7 (globalAG_s|referenceAG_s|globalA_s|referenceA_s|any_s)_id AND ¬(globalAG_t|referenceAG_t|globalA_t|referenceA_t|any_t)_id			update (attributegroup|attribute) name '*(globalAG_s| globalA_s)_name*' change id '';
						update (attributegroupref|attributeref) '*(referenceAG_s| referenceA_s)_ref*' at '*(referenceAG_s|referenceA_s)_locator* change id '';
						update anyattribute at '*any_s_locator*' change id '';
		n:m	8.3.8 M[n] = {(attribute_s|attributeGroup_s)_1,..., (attribute_s|attributeGroup_s)_n), N[m] = ∅		8.3.8.1 n > m	analog zu 8.1.15.2 (delete)
9. xs:key, xs:unique und xs:keyref	9.1 V_s ↔ V_t		9.1.1 (key_s|unique_s|keyref_s)_name != (key_t|unique_t|keyref_t)_name			update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator*' change name '*(key_t|unique_t|keyref_t)_name*';
			9.1.2 keyref_s_ref != keyref_t_refer			update constraint name '*keyref_s_name*' at '*keyref_s_locator*' change type keyref refer '*keyref_t_refer*';
		1:1	9.1.3 M[n] = {(selector_s|field_s)_1,..., (selector_s|field_s)_n), N[m] = {(selector_t|field_t)_1,..., (selector_t|field_t)_n) }	9.1.3 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]		O[n]!=∅: update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator* change *loop n:* remove <updconstraintpath> s_n;
						P[m]!=∅: update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator* change *loop n:* insert <updconstraintpath>_t_n;
			9.1.4 (key_s|unique_s|keyref_s)_id != (key_t|unique_t|keyref_t)_id			update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator*' change id '*(key_t|unique_t|keyref_t)_id*';
			9.1.5 (key_s|unique_s|keyref_s)_locator != (key_t|unique_t|keyref_t)_locator			update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator*' change id '*(key_t|unique_t|keyref_t)_locator*';
		n:m	9.1.6 M[n] = {(key_s|unique_s|keyref_s)_1,..., (key_s|unique_s|keyref_s)_n), N[m] = {(key_t|unique_t|keyref_t)_1,..., (key_t|unique_t|keyref_t)_m}	9.1.6 O[n] = M[n] \ N[m], P[m] = N[m] \ M[n]	9.1.6.1 n = m	loop n: (key_s|unique_s|keyref_s)_n ↔ (key_t|unique_t|keyref_t)_n (analog zu 9.1.1-9.1.5)
					9.1.6.2 n > m	analog zu 9.1.6.1 (P[m]!=∅, loop m, O[n]\\((key_s|unique_s|keyref_s)_m))
						loop n: delete constraint name '*(key_s|unique_s|keyref_s)_n_name*';
					9.1.6.3 n < m	analog zu 9.1.6.1 (O[n]!=∅, P[m]\\((key_t|unique_t|keyref_t)_n))
						loop n: add constraint name '*(key_t|unique_t|keyref_t)_n_name*' type (key|unique|(keyref refer '*keyref_t_n_refer*')) with <addconstraintpath> id '*(key_t|unique_t|keyref_t)_n_id*' in '*(key_t|unique_t|keyref_t)_n_locator*';
	9.2 ¬V_s ← V_t		9.2.1 ¬(key_s|unique_s|keyref_s)_id AND (key_t|unique_t|keyref_t)_id			analog zu 9.1.4
		1:1	9.2.2 (key_s|unique_s) AND keyref_t	9.2.2.1 Content(key_s|unique_s) = Content(keyref_t)		update constraint name '*(key_s|unique_s)_name*' at '*(key_s|unique_s)_locator*' change type keyref refer '*keyref_t_refer*';
				9.2.2.2Range(key_s|unique_s)=Range(keyref_t)		
			9.2.3 (key_s|keyref_s) AND unique_t	9.2.3.1 Content(key_s|keyref_s)= Content(unique_t)		update constraint name '*(key_s|keyref_s)_name*' at '*(key_s|keyref_s)_locator*' change type unique;
				9.2.3.2 Range(key_s|keyref_s)= Range(unique_t)		
			9.2.4 (unique_s|keyref_s) AND key_t	9.2.4.1 Content(unique_s|keyref_s) = Content(key_t)		update constraint name '*(unique_s|keyref_s)_name*' at '*(unique_s|keyref_s)_locator*' change type key;
				9.2.4.2 Range(unique_s|keyref_s) = Range(key_t)		
		n:m	9.2.5 M[n] = {(key_s|unique_s)_1,..., (key_s|unique_s)_n), N[m] = {keyref_t_1,..., keyref_t_m}		9.2.5.1 n = m	loop n: (key_s|unique_s)_n ← keyref_t_n (analog zu) 9.2.2)
					9.2.5.2 n > m	analog zu 9.2.5.1 (loop m, M[n]\\((key_s|unique_s)_m))
						analog zu 9.1.6.2 (delete, '*(key_s|unique_s)_n*')
					9.2.5.3 n < m	analog zu 9.2.5.1 (N[m]\\(keyref_t_n))
						analog zu 9.1.6.3 (add, keyref_t_n)
			9.2.6 M[n] = {(key_s|keyref_s)_1,..., (key_s|keyref_s)_n), N[m] = {unique_t_1,..., unique_t_m}		9.2.6.1 n = m	loop n: (key_s|keyref_s)_n ← unique_t_n (analog zu 9.2.3)
					9.2.6.2 n > m	analog zu 9.2.6.1 (loop m, M[n]\\((key_s|keyref_s)_m))
						analog zu 9.1.6.2 (delete, '*(key_s|keyref_s)_n*')
					9.2.6.3 n < m	analog zu 9.2.6.1 (N[m]\\(unique_t_n))
						analog zu 9.1.6.3 (add, unique_t_n)
			9.2.7 M[n] = {(unique_s|keyref_s)_1,..., (unique_s|keyref_s)_n), N[m] = {key_t_1,..., key_t_m}		9.2.7.1 n = m	loop n: (unique_s|keyref_s)_n ← key_t_n (analog zu 9.2.4)
					9.2.7.2 n > m	analog zu 9.2.7.1 (loop m, M[n]\\((unique_s|keyref_s)_m))
						analog zu 9.1.6.2 (delete, '*(unique_s|keyref_s)_n*')
					9.2.7.3 n < m	analog zu 9.2.7.1 (N[m]\\(key_t_n))
						analog zu 9.1.6.3 (add, key_t_n)
	9.3 V_s → ¬V_t		9.3.1 (key_s|unique_s|keyref_s)_id AND ¬(key_t|unique_t|keyref_t)_id			update constraint name '*(key_s|unique_s|keyref_s)_name*' at '*(key_s|unique_s|keyref_s)_locator*' change id '';
		1:1	9.3.2 keyref_s AND (key_t|unique_t)	9.3.2.1 Content(keyref_s) = Content(key_t|unique_t)		update constraint name '*keyref_s_name*' at '*keyref_s_locator*' change type (key|unique);
				9.3.2.2 Range(keyref_s) = Range(key_t|unique_t)		
			9.3.3 unique_s AND (key_t|keyref_t)	9.3.3.1 Content(unique_s) = Content(key_t|keyref_t)		update constraint name '*unique_s_name*' at '*unique_s_locator*' change type (key|keyref refer '*keyref_t_refer*');
				9.3.3.2 Range(unique_s) = Range(key_t|keyref_t)		
			9.3.4 key_s AND (unique_t|keyref_t)	9.3.4.1 Content(key_s) = Content(unique_t|keyref_t)		update constraint name '*key_s_name*' at '*key_s_locator*' change type (unique|keyref refer '*keyref_t_refer*');
				9.3.4.2 Range(key_s) = Range(unique_t|keyref_t)		
		n:m	9.3.5 M[n] = {keyref_s_1,..., keyref_s_n}, N[m] = {(key_t|unique_t)_1,..., (key_t|unique_t)_m},		9.3.5.1 n = m	loop n: keyref_s_n → (key_t|unique_t)_n (analog zu) 9.3.2)
					9.3.5.2 n > m	analog zu 9.3.5.1 (loop m, M[n]\\(keyref_s_m))
						analog zu 9.1.6.2 (delete, '*(keyref_s_n)*')
					9.3.5.3 n < m	analog zu 9.3.5.1 (N[m]\\((key_t|unique_t)_n))
						analog zu 9.1.6.3 (add, (key_t|unique_t)_n)
			9.3.6 M[n] = {unique_s_1,..., unique_s_n), N[m] = {(key_s|keyref_s)_1,..., (key_s|keyref_s)_m}		9.3.6.1 n = m	loop n: unique_s_n → (key_t|keyref_t)_n (analog zu 9.3.3)
					9.3.6.2 n > m	analog zu 9.3.6.1 (loop m, M[n]\\(unique_s_m))
						analog zu 9.1.6.2 (delete, '*unique_s_n*')
					9.3.6.3 n < m	analog zu 9.3.6.1 (N[m]\\((key_t|keyref_t)_n))
						analog zu 9.1.6.3 (add, (key_t|keyref_t)_n)
			9.3.7 M[n] = {key_t_1,..., key_t_n), N[m] = {(unique_s|keyref_s)_1,..., (unique_s|keyref_s)_m}		9.3.7.1 n = m	loop n: key_s_n → (unique_t|keyref_t)_n (analog zu 9.3.4)
					9.3.7.2 n > m	analog zu 9.3.7.1 (loop m, M[n]\\(key_s_m))
						analog zu 9.1.6.2 (delete, '*key_s_n*')
					9.3.7.3 n < m	analog zu 9.3.7.1 (N[m]\\((unique_t|keyref_t)_n))
						analog zu 9.1.6.3 (add, (unique_t|keyref_t)_n)

Abbildung C.5: ELaX Übersetzungsregeln Teil 5/5 abgeleitet aus [112] und [109]

Anhang D

ELaX Evaluierung

ELaX-Operation			Schemaevolutionsschritt			Umsetzung im Beispiel	Umsetzung im CodeXMapper	Umsetzung in CodeX
1. add	1.2 add attribute group	1.2.1 addattribute groupdef	1.2.1 Hinzufügen einer Attributgruppen Definition			ja, tour	√add attributegroup name 'tour' with attributeref 'start' use 'required' in '/node()/node()[@name="tour"]/.' attributeref 'ziel' use 'required' in '/node()/node()[@name="tour"]/.' ;	√add attributegroup name 'tour' with attributeref 'start' use 'required' in '/node()/node()[@name="tour"]/.' attributeref 'ziel' use 'required' in '/node()/node()[@name="tour"]/.' ;
		1.2.2 addattribute	1.2.2 Hinzufügen einer Attributdeklaration			ja, start, ziel	√add attribute name 'start' type 'xs:string' ;	√add attribute name 'start' type 'xs:string' ;
							√add attribute name 'ziel' type 'xs:string' ;	√add attribute name 'ziel' type 'xs:string' ;
							add attribute name 'akronym' type 'xs:string' ;	
		1.2.3 add attributeref	1.2.3 Hinzufügen einer Attribut-deklaration Referenz	1.2.3.2 "optional"		ja, rythmus (zeitType)	√add attributeref 'rythmus' use 'optional' in '/node()/node()/node()[@name="zeitType"]/.' ;	√add attributeref 'rythmus' use optional in '/node()/node()/node()[@name="zeitType"]/.' ;
				1.2.3.3 "required"		ja, start und ziel (tour), anbieterID (anbieterneuType)	*√ start und ziel bereits in 1.2.1 eingefügt*	*√start und ziel bereits in 1.2.1 eingefügt*
							√ add attributeref 'anbieterID' use 'required' in '/node()/node()[@name="anbieterneuType"]/node()/.' ;	√add attributeref 'anbieterID' use required in '/node()/node()[@name="anbieterneuType"]/node()/node()/.' ;
							add attributeref 'akronym' use 'required' in '/node()/node()[@name="nameType"]/node()/.' ;	
		1.2.4 addattribute groupref	1.2.4 Hinzufügen einer Attributgruppen Definition Referenz			ja, tour (zeitType, routenType)	√add attributegroupref 'tour'in'/node()/node()[@name="routenType"]/.' ;	√add attributegroupref 'tour' in '/node()/node()[@name="routenType"]/.' ;
							√add attributegroupref 'tour' in '/node()/node()/node()[@name="zeitType"]/.' ;	√add attributegroupref 'tour'in'/node()/node()/node()[@name="zeitType"]/.' ;
	1.3 addgroup		1.3 Hinzufügen einer Gruppen-definition	1.3.1 Sequenz	1.3.1.2 minOccurs > 0 maxOccurs > 0	ja, zeitType, routenfahrplanType, fernbusanbieterType	***(√)**zeitType wird beim Einfügen in 1.7.4 erledigt* √add group mode sequence in '/node()/node()/node()[@name="zeitType"]/.' ;	***(√)**zeitType wird beim Einfügen in 1.7.4 erledigt* √add group mode sequence in '/node()/node()/node()[@name="zeitType"]/.' ;
							√add complextype name 'routenfahrplanType' ; √add group mode sequence in '/node()/node()[@name="routenfahrplanType"]/.' ;	√add complextype name 'routenfahrplanType' ; √add group mode sequence in '/node()/node()[@name="routenfahrplanType"]/.' ;
							√add complextype name 'fernbusanbieterType' ; √add group mode sequence in '/node()/node()[@name="fernbusanbieterType"]/.' ;	√add complextype name 'fernbusanbieterType' ; √add group mode sequence in '/node()/node()[@name="fernbusanbieterType"]/.' ;
							add group mode sequence in '/node()/node()[@name="anbieterneuType"]/node()/.' ;	*add group mode sequence in '/node()/node()[@name="anbieterneuType"]/node()/node()/.' ;*
							add complextype name 'routenType' ; add group mode sequence in '/node()/node()[@name="routenType"]/.' ;	
							add complextype name 'ausstattungType' ; add group mode sequence in '/node()/node()[@name="ausstattungType"]/.' ;	
	1.4 addst		1.4 Hinzufügen einer einfachen Typdefinition in Hierarchie als	1.4.1 Einschränkung		ja, gründungsjahrType, fehlType	√add simpletype name 'gründungsjahrType' mode restriction of 'xs:gYear' with mininclusive '1900' ;	√add simpletype name 'gründungsjahrType' mode restriction of 'xs:gYear' with mininclusive '1900' ;
							Xadd fehlType	√add simpletype name 'fehlType' mode restriction of 'xs:string' with pattern 'keine Angabe' ;
				1.4.3 Vereinigung		ja, geldneuType	√add simpletype name 'geldneuType' mode union 'geldType' 'fehlType' ;	√add simpletype name 'geldneuType' mode union 'geldType' 'fehlType' ;
	1.5 addct		1.5 Hinzufügen einer komplexen Typdefinition in Hierarchie als	1.5.4 Erweiterung: complexContent		ja, anbieterneuType	√add complextype name 'anbieterneuType' mode extension_cc with base 'anbieterType' ; *mit add group benötigte Sequenz in 1.3.1.2 eingefügt*	√add complextype name 'anbieterneuType' mode extension_cc with base 'anbieterType' ; *√ mit add group benötigte Sequenz in 1.3.1.2 eingefügt*
							add complextype name 'nameType' mode extension_cc with base 'beschreibungType' ;	
	1.6 add element	1.6.1 add elementdef	1.6.1 Hinzufügen einer Elementdeklaration			ja, routen, fernbusanbieter	√add element name 'routen' type 'routenType' ;	√add element name 'routen' type 'routenType' ;
							√add element name 'fernbusanbieter' type 'fernbusanbieterType' ;	√add element name 'fernbusanbieter' type 'fernbusanbieterType' ;
							Xadd element name 'name' type 'nameType' ;	
							add element name 'routenfahrplan' type 'routenfahrplanType' ;	
							add element name 'ausstattung' type 'ausstattungType' nillable 'true' ;	
		1.6.2 add elementref	1.6.2 Hinzufügen einer Element-deklaration Referenz in	1.6.2.1 Sequenz	1.6.2.1.1 minOccurs = 0 maxOccurs ≥ 0	ja, hinfahrt und rückfahrt (routenfahrplanType)	*(√)mit update elementref 'rückfahrt' in 3.6.2.1 erledigt*	√add elementref 'hinfahrt' minoccurs '0' as first into '/node()/node()[@name="routenfahrplanType"]/node()/.' ;
							(√)mit update elementref 'hinfahrt' in 3.6.2.1 eledigt	√add elementref 'rückfahrt' minoccurs '0' as last into '/node()/node()[@name="routenfahrplanType"]/node()/.' ;
					1.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja, fernbusanbieter (fernbusverkehrType), anbieter(fernbusanbieterType), ausstattung (anbieterneuType), ab und an (zeitType)	*(√)mit update fernbuslinien in 3.6.2.1 erledigt add elementref 'routen' maxoccurs 'unbounded' in '/node()/node()[@name="fernbusverkehrType"]/node()/.' ;*	√add elementref 'fernbusanbieter' at last into '/node()/node()[@name="fernbusverkehrType"]/node()/.' ;
							(√)anbieter wird in neue Position verschoben in 3.6.2.5 erledigt	√add elementref 'anbieter' maxoccurs 'unbounded' in '/node()/node()[@name="fernbusanbieterType"]/node()/.' ;
							√add elementref 'ausstattung' in '/node()/node()[@name="anbieterneuType"]/node()/node()/.' ;	√add elementref 'ausstattung' in '/node()/node()[@name="anbieterneuType"]/node()/node()/node()/.' ;
							√add elementref 'ab' in '/node()/node()/node()[@name="zeitType"]/node()/.' ;	√add elementref 'ab' as first into '/node()/node()/node()[@name="zeitType"]/node()/.' ;
							√add elementref 'an' in '/node()/node()/node()[@name="zeitType"]/node()/.' ;	√add elementref 'an' as last into '/node()/node()/node()[@name="zeitType"]/node()/.' ;
							add elementref 'fernbusanbieter-ref' in '/node()/node()[@name="routeType"]/node()/.' ;	
							add elementref 'routenfahrplan' in '/node()/node()[@name="routeType"]/node()/.' ;	
		1.6.3 addelement wildcard	1.6.3 Hinzufügen einer Wildcard	1.6.3.2 minOccurs > 0 maxOccurs > 0		ja, any (ausstattungType)	√add any namespace 'http://www.ausstattung.org' maxoccurs 'unbounded' in '/node()/node()[@name="ausstattungType"]/node()/.' ;	√add any namespace 'http://www.ausstattung.org' maxoccurs 'unbounded' in '/node()//node()[@name="ausstattungType"]/node()/.' ;
	1.7 addmodule		1.7Hinzufügen einer Moduldeklaration	1.7.4 override		ja, analog zu 3.7.2.9	*add module from 'zeit.xsd' mode override complextype name 'zeitType' ;*	*add module from 'zeit.xsd' mode override complextype name 'zeitType' ;*
	1.8 addconstraint		1.8 Hinzufügen einer Constraint-deklaration	1.8.1 Hinzufügen eines Primärschlüssels	1.8.1.1 key	ja, routenKey (fernbusverkehr)	√add constraint name 'routenKey' type 'key' with selector './routen' field '@start' field '@ziel' in '/node()/node()[@name="fernbusverkehr"]/.' ;	√add constraint name 'routenKey' type key with selector './routen' field '@start' field '@ziel' in '/node()/node()[@name="fernbusverkehr"]/.' ;
				1.8.2 Hinzufügen eines Fremdschlüssels (keyref)		ja, startzielKey und zielstartKey	√add constraint name 'startzielKey' type 'keyref' refer 'routeKey' with selector './/hinfahrt/zeit' field '@start' field '@ziel' in '/node()/node()[@name="fernbusverkehr"]/.' ;	√add constraint name 'startzielKey' type keyref refer 'routeKey' with selector './/hinfahrt/zeit' field '@start' field '@ziel' in '/node()/node()[@name="fernbusverkehr"]/.' ;
							√add constraint name 'zielstartKey' type 'keyref' refer 'routeKey' with selector './/rückfahrt/zeit' field '@ziel' field '@start' in '/node()/node()[@name="fernbusverkehr"]/.' ;	√add constraint name 'zielstartKey' type keyref refer 'routeKey' with selector './/rückfahrt/zeit' field '@ziel' field '@start' in '/node()/node()[@name="fernbusverkehr"]/.' ;
				1.8.3 Hinzufügen einer Identität		ja, anbieterID	√add attribute name 'anbieterID' type 'xs:ID' ;	√add attribute name 'anbieterID' type 'xs:ID' ;
				1.8.4 Hinzufügen einer Identitätsreferenz		ja, fernbusanbieter-ref	√add element name 'fernbusanbieter-ref' type 'xs:IDREF' ;	√add element name 'fernbusanbieter-ref' type 'xs:IDREF' ;
2. delete	2.2 del attribute group	2.2.2 delattribute	2.2.2 Entfernen einer Attributdeklaration			ja, MwSt.	√delete attribute name 'MwSt.' ;	√delete attribute name 'MwSt.' ;
							delete attribute name 'abkürzung' ;	
		2.2.3 delattributeref	2.2.3 Entfernen einer Attributdekla-ration Referenz	2.2.3.3 "required"		ja, MwSt. (preiseType)	√delete attributeref 'MwSt.' at '/node()/node()[@name="preiseType"]/.' ;	√delete attributeref 'MwSt.' at '/node()/node[@name="preiseType"]/.' ;
							delete attributeref 'abkürzung' at '/node()/node()[@name="nameType"]/node()/.' ;	
	2.3 delgroup		2.3 Entfernen einer Gruppendefinition	2.3.1 Sequenz	2.3.1.2 minOccurs > 0 maxOccurs > 0	ja, fernbuslinieType	√delete complextype name 'fernbuslinieType' ;	√delete complextype name 'fernbuslinieType' ;
							delete group at '/node()/node()[@name="fernbuslinieType"]/.' ;	
							delete complextype name 'nameType' ;	
							delete complextype name 'ausstattungType' ;	
							delete group at '/node()/node()[@name="ausstattungType"]/.' ;	
							delete complextype name 'fahrplanType' ;	
							delete group at '/node()/node()[@name="fahrplanType"]/.' ;	
	2.4 delst		2.4 Entfernen einer einfachen Typdefinition aus Hierarchie	2.4.1 Einschränkung		ja, MwSt.Type, serviceType, sitzabstandType, wahlType	√delete simpletype name 'MwSt.Type' ;	√delete simpletype name 'MwSt.Type' ;
							√delete simpletype name 'serviceType' ;	√delete simpletype name 'serviceType' ;
							√delete simpletype name 'sitzabstandType' ;	√delete simpletype name 'sitzabstandType' ;
							√delete simpletype name 'wahlType' ;	√delete simpletype name 'wahlType' ;
				2.4.2 Liste		ja, cateringType	√delete simpletype name 'cateringType' ;	√delete simpletype name 'cateringType' ;
	2.6 del element	2.6.1 delelementdef	2.6.1 Entfernen einer Elementdeklaration			ja, fernbuslinie, catering, sitzabstand, steckdosen, wlan, beschreibung	√delete element name 'fernbuslinie' ;	√delete element name 'fernbuslinie' ;
							√delete element name 'catering' ;	√delete element name 'catering' ;
							√delete element name 'sitzabstand' ;	√delete element name 'sitzabstand' ;
							√delete element name 'steckdosen' ;	√delete element name 'steckdosen' ;
							√delete element name 'wlan' ;	√delete element name 'wlan' ;
							√delete element name 'beschreibung' ;	√delete element name 'beschreibung' ;
							Xdelete element name 'name' ;	
							delete element name 'fahrplan' ;	
							delete element name 'ausstattung' ;	
		2.6.2 delelementref	2.6.2 Entfernen einer Elementdeklaration Referenz aus	2.6.2.1 Sequenz	2.6.2.1.2 minOccurs > 0 maxOccurs > 0	ja, catering, sitzabstand, steckdosen, wlan (ausstattungType)	√delete elementref 'catering' at '/node()/node()[@name="ausstattungType"]/node()/.' ;	√delete elementref 'catering' at '/node()//node()[@name="ausstattungType"]/node()/.' ;
							√delete elementref 'sitzabstand' at '/node()/node()[@name="ausstattungType"]/node()/.' ;	√delete elementref 'sitzabstand' at '/node()//node()[@name="ausstattungType"]/node()/.' ;
							√delete elementref 'steckdosen' at '/node()/node()[@name="ausstattungType"]/node()/.' ;	√delete elementref 'steckdosen' at '/node()//node()[@name="ausstattungType"]/node()/.' ;
							√delete elementref 'wlan' at '/node()/node()[@name="ausstattungType"]/node()/.' ;	√delete elementref 'wlan' at '/node()//node()[@name="ausstattungType"]/node()/.' ;
							delete elementref 'ausstattung' at	

Abbildung D.1: ELaX Evaluierung Teil 1/2 abgeleitet aus B

							'/node()/node()[@name="fernbuslinieType"]/node()/.' ; *delete elementref 'fahrplan' at '/node()/node()[@name="fernbuslinieType"]/node()/.' ;* *delete elementref 'beschreibung' at '/node()/node()[@name="fernbuslinieType"]/node()/.' ;*	
	2.7 delmodule		2.7 Entfernen einer Moduldeklaration	2.7.3 include		ja, analog zu 3.7.2.9	*delete module at '/node()/node()[1]' ;*	*delete module at '/node()/.' ;*
3. update	3.1 updannotation		3.1 Ändern einer Anmerkung	3.1.1 Ändern einer documentation		ja, von 2012 auf 2013	√update annotation at '/node()/.' change documentation 'Übersicht über die Fernbuslinien in Deutschland Stand: 2013' ;	√update annotation at '/node()/.' documentation 'Übersicht über die Fernbuslinien in Deutschland Stand: 2013' ;
	3.2 upd attribute group	3.2.2 upd attribute	3.2.2 Ändern einer Attribut-deklaration	3.2.2.1 Umbenennen einer Attributdeklaration		ja,abkürzung in akronym	**(√)***mit delete abkürzung in 2.2.2 und add akronym in 1.2.2 erledigt*	√update attribute name 'abkürzung' change name 'akronym' ;
				3.2.2.2 Ändern der Typdefinition		ja, routeID	√update attribute name 'routeID' change type 'xs:ID' ;	√update attribute name 'routeID' change type 'xs:ID' ;
				3.2.2.3 Ändern des fixed-Wertes		ja, last_schema_change	√update attribute name 'last_schema_change' change fixed '01.01.2013' ;	√update attribute name 'last_schema_change' change fixed '01.01.2013' ;
		3.2.3 upd attribute ref	3.2.3 Ändern einer Attributdeklaration Referenz	3.2.3.1 Ändern des Referenznamen		ja, abkürzung in akronym (nameType)	**(√)***mit delete abkürzung in 1.2.3.3 und add akronym in 2.2.3.3 erledigt*	√update attributeref 'abkürzung' at '/node()/node[@name="nameType"]/node()/node()/.' change ref 'akronym';
				3.2.3.4 Ändern der Attributverwendung	3.2.3.4.1 "optional" -> "required"	ja, routeID (routeType)	√update attributeref 'routeID' at '/node()/node()[@name="routeType"]/.' change use 'required' ;	√update attributeref 'routeID' at '/node()/node()[@name="routeType"]/.' change use required ;
	3.3 updgroup		3.3 Ändern einer Gruppendefinition	3.3.1 Ändern des Typs	3.3.1.5 Menge -> Sequenz	ja, preiseType	√update group at '/node()/node()[@name="preiseType"]/.' change mode sequence minoccurs '0' maxoccurs 'unbounded' ;	√update group at '/node()/node[@name="preiseType"]/.' change mode sequence minoccurs '0' maxoccurs 'unbounded' ;
				3.3.2 Verringern des minimalen Vorkommens		ja, preiseType	**(√)***mit 3.3.1.5 erledigt*	**(√)***mit 3.3.1.5 erledigt*
				3.3.5 Erhöhen des maximalen Vorkommens		ja, preiseType	**(√)***mit 3.3.1.5 erledigt*	**(√)***mit 3.3.1.5 erledigt*
	3.4 updst		3.4 Ändern einer einfachen Typdefinition	3.4.2 Ändern des Typs	3.4.2.6 Vereinigung -> Liste	ja, buslinienType	√update simpletype name 'buslinienType' change mode list 'buslinieType' ;	√update simpletype name 'buslinienType' change mode list 'buslinieType' ;
				3.4.4 Entfernen einer einfachen Typdefinition aus einer Vereinigung		ja, buslinienType	**(√)***mit 3.4.2.6 erledigt*	**(√)***mit 3.4.2.6 erledigt*
				3.4.7 Modifizieren einer Einschränkung		ja, buslinieType	√update simpletype name 'buslinieType' change mode restriction of 'xs:string' modify enumeration 'Joost-Ostsee-Express' at '/node()/node()[@name="buslinieType"]/node()/node()[1]' modify enumeration 'Wörlitz-Tourist' at '/node()/node()[@name="buslinieType"]/node()/node()[2]' modify enumeration 'Bayern-Express&P.Kühn' at '/node()/node()[@name="buslinieType"]/node()/node()[3]' modify enumeration 'Eurolines-Scandinavia' at '/node()/node()[@name="buslinieType"]/node()/node()[4]' ;	√update simpletype name 'buslinieType' change mode restriction of 'xs:string' modify enumeration 'Joost-Ostsee-Express' at '/node()/node()[@name="buslinieType"]/node()/node()[1]' modify enumeration 'Wörlitz-Tourist' at '/node()/node()[@name="buslinieType"]/node()/node()[2]' modify enumeration 'Bayern-Express&P.Kühn' at '/node()/node()[@name="buslinieType"]/node()/node()[3]' modify enumeration 'Eurolines-Scandinavia' at '/node()/node()[@name="buslinieType"]/node()/node()[4]' ;
	3.5 updct		3.5 Ändern einer komplexen Typdefinition	3.5.1 Umbenennen der Typdefinition		ja, fahrplanType in routenType	**(√)***mit delete fahrplanType in 2.3.1.2 und add routenType in 1.3.1.2 erledigt*	√update complextype name 'fahrplanType' change name 'routenType' ;
				3.5.2Ändern des Inhalts-typs auf	3.5.2.1 mixed = "true"	ja, beschreibungType	√update complextype name 'beschreibungType' change mixed 'true' ;	√update complextype name 'beschreibungType' change mixed true ;
				3.5.3 Ändern der Abgeschlossenheit		ja, anbieterType	√update complextype name 'anbieterType' change final 'restriction' ;	√update complextype name 'anbieterType' change final restriction ;
				3.5.4 Ändern des Inhalts (und Basis)	3.5.4.9 Extension: simple Content -> Extension: complex Content	ja, nameType	**(√)** *mit delete nameType in 2.3.1.2 und add nameType in 1.5.4 erledigt*	√update complextype name 'nameType' change mode extension_cc with base 'beschreibungType' ;
	3.6 upd element	3.6.1 upd element def	3.6.1 Ändern einer Elementdeklaration	3.6.1.1 Umbenennen einer Elementdeklaration		ja, fahrplan in routenfahrplan	**(√)** *mit delete fahrplan in 2.6.1 und add routenfahrplan in 1.6.1 erledigt*	√update element name 'fahrplan' change name 'routenfahrplan' ;
				3.6.1.2 Ändern der Typdefinition		ja, routenfahrplan, anbieter, sparpreise, erwachsene, kinder, gruppen, tageskarte, fahrradmitnahme, gründungsjahr	**(√)** *mit delete fahrplan in 2.6.1 und add routenfahrplan in 1.6.1 erledigt*	√update element name 'fahrplan' change type 'routenType' ;
							√update element name 'anbieter' change type 'anbieterneuType' ;	√update element name 'anbieter' change type 'anbieterneuType' ;
							√update element name 'sparpreise' change type 'geldneuType' ;	√update element name 'sparpreise' change type 'geldneuType' ;
							√update element name 'erwachsene' change type 'geldneuType' ;	√update element name 'erwachsene' change type 'geldneuType' ;
							√update element name 'kinder' change type 'geldneuType' ;	√update element name 'kinder' change type 'geldneuType' ;
							√update element name 'gruppen' change type 'geldneuType' ;	√update element name 'gruppen' change type 'geldneuType' ;
							√update element name 'tageskarte' change type 'geldneuType' ;	√update element name 'tageskarte' change type 'geldneuType' ;
							√update element name 'fahrradmitnahme' change type 'geldneuType' ;	√update element name 'fahrradmitnahme' change type 'geldneuType' ;
							√update element name 'gründungsjahr' change type 'gründungsjahrType' ;	*√update element name 'gründungsjahr' change type 'gründungsjahrType' ;*
				3.6.1.6 Ändern der Nullwert fähigkeit	3.6.1.6.2 nillable= "true" einfügen: minOccurs ≥ 0 maxOccurs ≥ 0	ja, ausstattung	**(√)***mit delete ausstattung in 2.6.1 und add ausstattung in 1.6.1 erledigt*	√update element name 'ausstattung' change nillable true ;
		3.6.2 updelementref	3.6.2 Ändern einer Elementdeklaration Referenz	3.6.2.1 Ändern des Referenznamen		ja, fernbuslinie in routen (fernbusverkehrType), hinfahrt in fernbusanbieter-ref und rückfahrt in routenfahrplan (routeType)	**(√)** *mit add routen in 1.6.2.1.2 erledigt* *update elementref 'fernbuslinie' at '/node()/node()[@name="fernbusverkehrType"]/node()/.' change ref 'fernbusanbieter' maxoccurs '';*	√update elementref 'fernbuslinie' at '/node()/node()[@name="fernbusverkehrType"]/node()/.' change ref 'routen' maxoccurs 'unbounded' ;
							(√) *mit add fernbusanbieter-ref in 1.6.2.1.2* *update elementref 'hinfahrt' at '/node()/node()[@name="routeType"]/node()/.' change move to '/node()/node()[@name="routenfahrplanType"]/node()/.' ;*	√update elementref 'hinfahrt' at '/node()/node()[@name="routeType"]/node()/.' change ref 'fernbusanbieter-ref' minoccurs '' ;
							(√)*mit add routenfahrplan in 1.6.2.1.2 erledigt* *update elementref 'rückfahrt' at '/node()/node()[@name="routeType"]/node()/.' change move to '/node()/node()[@name="routenfahrplanType"]/node()/.' ;*	√update elementref 'rückfahrt' at '/node()/node()[@name="routeType"]/node()/.' change ref 'routenfahrplan' minoccurs '' ;
							√update elementref 'route' at '/node()/node()[@name="fahrplanType"]/node()/.' change move to '/node()/node()[@name="routenType"]/node()/.' ;	
				3.6.2.3 Erhöhen des minimalen Vorkommens		ja, fernbusanbieter-ref und routenfahrplan (routeType), sparpreise erwachsene, kinder, gruppen, tageskarte, fahrradmitnahme (preiseType)	**(√)***mit add fernanbusanbierter-ref und routenfahrplan in* 1.6.2.1.2 *erledigt*	**(√)***mit update fernanbusanbierter-ref und routenfahrplan in 3.6.2.1 erledigt*
							√update elementref 'sparpreise' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ;	√update elementref 'sparpreise' at '/node()/node[@name="preiseType"]/node()/.' change minoccurs '' ;
							√update elementref 'erwachsene' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ;	√update elementref 'erwachsene' at '/node()/node[@name="preiseType"]/node()/.' change minoccurs '' ;
							√update elementref 'kinder' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ;	√update elementref 'kinder' at '/node()/node[@name="preiseType"]/node()/.' change minoccurs '' ;
							√update elementref 'gruppen' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ;	√update elementref 'gruppen' at '/node()/node[@name="preiseType"]/node()/.' change minoccurs '' ;
							√update elementref 'tageskarte' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ;	√update elementref 'tageskarte' at '/node()/node[@name="preiseType"]/node()/.' change minoccurs '' ;
							√update elementref 'fahrradmitnahme' at '/node()/node()[@name="preiseType"]/node()/.' change minoccurs '' ;	√update elementref 'fahrradmitnahme' at '/node()/node[@name="preiseType"]/node()/.' change minoccurs '' ;
				3.6.2.5 Erhöhen des maximalen Vorkommens		ja, routen (fernbusverkehrType)	*update elementref 'anbieter' at '/node()/node()[@name="fernbuslinieType"]/node()/.' change maxoccurs 'unbounded' move to '/node()/node()[@name="fernbusanbieterType"]/node()/.' ;*	*in 3.6.2.1 mit erledigt*
	3.7 updmodule		3.7 Ändern einer Moduldeklaration	3.7.2 Ändern des Typs	3.7.2.9 include -> override	ja, zeit.xsd	**(√)***mit delete include in 2.7.3 und add override in 1.7.4*	**(√)***mit delete include in 2.7.3 und add override in 1.7.4*
	3.8 updconstraint		3.8 Ändern einer Constraint-deklaration	3.8.1 Umbenennen der Schlüssel-/Identitätsdeklaration		ja, anbieterKey in anbieternameKey	**(√)***mit 3.8.4.1 erledigt*	**(√)***mit 3.8.4.1 erledigt*
				3.8.2 Ändern des Typs	3.8.2.1 key -> unique	ja, routeKey	√update constraint name 'routeKey' at '/node()/node()[@name="fernbusverkehr"] /.' change type 'unique' remove selector './fernbuslinie/fahrplan/route' insert selector './routen/route' ;	√update constraint name 'routeKey' at '/node()/node[@name="fernbusverkehr"]/.' change type unique remove selector './fernbuslinie/fahrplan/route' insert selector './routen/route' ;
				3.8.4 Ändern des Constraintpfades	3.8.4.1 Ändern des selector-Pfades	ja, anbieterKey	√update constraint name 'anbieterKey' at '/node()/node()[@name="fernbusverkehr"]/.' change name 'anbieternameKey' insert selector './fernbusanbieter/anbieter' insert field 'name/@akronym' at '/node()/node()[@name="fernbusverkehr"]/node()[@name="anbieternameKey"]/node()[2]' ;	√update constraint name 'anbieterKey' at '/node()/node[@name="fernbusverkehr"]/.' change name 'anbieternameKey' remove selector './fernbuslinie/anbieter' insert selector './fernbusanbieter/anbieter' ;
	3.9 updschema		3.9 Ändern einer Schemadeklaration	3.9.4 Ändern der Version		ja, von 1.0 auf 1.1	√update schema change version '1.1' ;	√update schema change version '1.1' ;
√ = umgesetzt, (√) = teilweise oder andersweitig umgesetzt, X = nicht umgesetzt								

Abbildung D.2: ELaX Evaluierung Teil 2/2 abgeleitet aus B

Anhang E

XML-Dokumente

E.1 Quellschema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="
    unqualified" version="1.0">
 <xs:annotation>
  <xs:documentation xml:lang="de">
   Übersicht über die Fernbuslinien in Deutschland Stand: 2012
  </xs:documentation>
 </xs:annotation>
 <xs:include schemaLocation = "zeit.xsd"/>
 <xs:element name="fernbusverkehr" type="fernbusverkehrType">
  <xs:key name="anbieterKey">
   <xs:selector xpath="./fernbuslinie/anbieter"/>
   <xs:field xpath="name"/>
  </xs:key>
  <xs:key name="routeKey">
   <xs:selector xpath="./fernbuslinie/fahrplan/route"/>
   <xs:field xpath="von"/>
   <xs:field xpath="nach"/>
  </xs:key>
 </xs:element>
 <xs:complexType name="fernbusverkehrType">
  <xs:sequence>
   <xs:element ref="fernbuslinie" maxOccurs="10"/>
  </xs:sequence>
  <xs:attribute ref="last_schema_change" use="required"/>
 </xs:complexType>
 <xs:element name="fernbuslinie" type="fernbuslinieType"/>
 <xs:complexType name="fernbuslinieType">
  <xs:sequence>
   <xs:element ref="anbieter"/>
   <xs:element ref="fahrplan"/>
   <xs:element ref="ausstattung"/>
   <xs:element ref="beschreibung"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="anbieter" type="anbieterType"/>
 <xs:complexType name="anbieterType" final="#all">
  <xs:sequence>
   <xs:element ref="name"/>
   <xs:element ref="gründungsjahr"/>
   <xs:element ref="unternehmungsform"/>
   <xs:element ref="homepage"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="name" type="nameType"/>
 <xs:complexType name="nameType">
  <xs:simpleContent>
   <xs:extension base="xs:string">
    <xs:attribute ref="abkürzung" use="required"/>
   </xs:extension>
```

```
 </xs:simpleContent>
</xs:complexType>
<xs:attribute name="abkürzung" type="xs:string"/>
<xs:element name="gründungsjahr" type="xs:gYear"/>
<xs:element name="unternehmungsform" type="unternehmungsformType"/>
<xs:simpleType name="unternehmungsformType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="GmbH"/>
  <xs:enumeration value="AG"/>
  <xs:enumeration value="KG"/>
  <xs:enumeration value="GmbH&amp;Co.KG"/>
 </xs:restriction>
</xs:simpleType>
<xs:element name="homepage" type="xs:anyURI"/>
<xs:element name="fahrplan" type="fahrplanType"/>
<xs:complexType name="fahrplanType">
 <xs:sequence>
  <xs:element ref="route" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
<xs:element name="route" type="routeType"/>
<xs:complexType name="routeType">
 <xs:sequence>
  <xs:element ref="von"/>
  <xs:element ref="über" minOccurs="0"/>
  <xs:element ref="nach"/>
  <xs:element ref="preise"/>
  <xs:element ref="hinfahrt" minOccurs="0"/>
  <xs:element ref="rückfahrt" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute ref="routeID" use="optional"/>
 <xs:attribute ref="buslinien" use="required"/>
</xs:complexType>
<xs:element name="von" type="xs:string"/>
<xs:element name="über" type="xs:string"/>
<xs:element name="nach" type="xs:string"/>
<xs:element name="preise" type="preiseType"/>
<xs:complexType name="preiseType">
 <xs:all>
  <xs:element ref="sparpreise" minOccurs="0"/>
  <xs:element ref="erwachsene" minOccurs="0"/>
  <xs:element ref="kinder" minOccurs="0"/>
  <xs:element ref="gruppen" minOccurs="0"/>
  <xs:element ref="tageskarte" minOccurs="0"/>
  <xs:element ref="fahrradmitnahme" minOccurs="0"/>
 </xs:all>
 <xs:attribute ref="währung" use="required"/>
 <xs:attribute ref="MwSt." use="required"/>
</xs:complexType>
<xs:element name="sparpreise" type="geldType"/>
<xs:simpleType name="geldType">
 <xs:restriction base="xs:decimal">
  <xs:fractionDigits value="2"/>
  <xs:minExclusive value="0"/>
 </xs:restriction>
</xs:simpleType>
<xs:element name="erwachsene" type="geldType"/>
<xs:element name="kinder" type="geldType"/>
<xs:element name="gruppen" type="geldType"/>
<xs:element name="tageskarte" type="geldType"/>
<xs:element name="fahrradmitnahme" type="geldType"/>
<xs:attribute name="währung" type="währungType"/>
<xs:simpleType name="währungType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="EUR"/>
  <xs:enumeration value="CHF"/>
 </xs:restriction>
</xs:simpleType>
<xs:attribute name="MwSt." type="MwSt.Type"/>
<xs:simpleType name="MwSt.Type">
 <xs:restriction base="xs:string">
  <xs:enumeration value="inklusive"/>
  <xs:enumeration value="nicht inklusive"/>
 </xs:restriction>
</xs:simpleType>
<xs:attribute name="routeID" type="xs:string"/>
```

```
 <xs:attribute name="buslinien" type="buslinienType"/>
 <xs:simpleType name="buslinienType">
  <xs:union memberTypes="buslinieType xs:string"/>
 </xs:simpleType>
 <xs:simpleType name="buslinieType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="Joost's Ostsee-Express"/>
   <xs:enumeration value="Wörlitz Tourist"/>
   <xs:enumeration value="Bayern Express &amp; P. Kühn"/>
   <xs:enumeration value="Eurolines Scandinavia"/>
   <xs:enumeration value="Touring"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="ausstattung" type="ausstattungType"/>
 <xs:complexType name="ausstattungType">
  <xs:sequence>
   <xs:element ref="catering"/>
   <xs:element ref="sitzabstand"/>
   <xs:element ref="steckdosen"/>
   <xs:element ref="wlan"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="catering" type="cateringType"/>
 <xs:simpleType name="cateringType">
  <xs:list itemType="serviceType"/>
 </xs:simpleType>
 <xs:simpleType name="serviceType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="Snacks"/>
   <xs:enumeration value="Kaffee"/>
   <xs:enumeration value="Kaltgetränke"/>
   <xs:enumeration value="nein"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="sitzabstand" type="sitzabstandType"/>
 <xs:simpleType name="sitzabstandType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="groß"/>
   <xs:enumeration value="klein"/>
   <xs:enumeration value="nein"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="steckdosen" type="wahlType"/>
 <xs:simpleType name="wahlType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="ja"/>
   <xs:enumeration value="nein"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="wlan" type="wahlType"/>
 <xs:element name="beschreibung" type="beschreibungType"/>
 <xs:complexType name="beschreibungType" mixed="false">
  <xs:sequence>
   <xs:element ref="kurzbeschreibung"/>
   <xs:element ref="langbeschreibung" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="kurzbeschreibung" type="xs:string"/>
 <xs:element name="langbeschreibung" type="xs:string"/>
 <xs:attribute name="last_schema_change" fixed="01.01.2012"/>
</xs:schema>
```

Listing E.1: Quellschema: Fernbusverkehr.xsd

E.2 Quellinstanz

```
<?xml version="1.0" encoding="UTF-8"?>
<fernbusverkehr xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
     Fernbusverkehr.xsd" last_schema_change="01.01.2012">
<fernbuslinie>
 <anbieter>
```

```
 <name abkürzung="MFB">MeinFernbus</name>
 <gründungsjahr>2011</gründungsjahr>
 <unternehmungsform>GmbH</unternehmungsform>
 <homepage>http://meinfernbus.de/</homepage>
</anbieter>
<fahrplan>
 <route routeID="rou1" buslinien="Joost's Ostsee-Express Wörlitz Tourist">
  <von>Warnemünde</von>
  <über>Rostock</über>
  <nach>Berlin</nach>
  <preise währung="EUR" MwSt.="inklusive">
   <sparpreise>11.00</sparpreise>
   <erwachsene>21.00</erwachsene>
   <kinder>12.50</kinder>
   <tageskarte>35.00</tageskarte>
   <gruppen>16.50</gruppen>
   <fahrradmitnahme>9.00</fahrradmitnahme>
  </preise>
  <hinfahrt>
   <zeit rythmus="täglich">
    <abfahrt>
     <stunde>06</stunde>
     <minute>40</minute>
    </abfahrt>
    <ankunft>
     <stunde>09</stunde>
     <minute>45</minute>
    </ankunft>
   </zeit>
   <zeit rythmus="täglich">
    <abfahrt>
     <stunde>11</stunde>
     <minute>10</minute>
    </abfahrt>
    <ankunft>
     <stunde>14</stunde>
     <minute>15</minute>
    </ankunft>
   </zeit>
   <zeit rythmus="täglich">
    <abfahrt>
     <stunde>15</stunde>
     <minute>10</minute>
    </abfahrt>
    <ankunft>
     <stunde>18</stunde>
     <minute>15</minute>
    </ankunft>
   </zeit>
   <zeit rythmus="täglich">
    <abfahrt>
     <stunde>19</stunde>
     <minute>10</minute>
    </abfahrt>
    <ankunft>
     <stunde>22</stunde>
     <minute>15</minute>
    </ankunft>
   </zeit>
   <zeit rythmus="FR SA SO">
    <abfahrt>
     <stunde>17</stunde>
     <minute>10</minute>
    </abfahrt>
    <ankunft>
     <stunde>20</stunde>
     <minute>15</minute>
    </ankunft>
   </zeit>
  </hinfahrt>
  <rückfahrt>
   <zeit rythmus="täglich">
    <ab>07:00:00+01:00</ab>
    <an>10:00:00+01:00</an>
   </zeit>
   <zeit rythmus="täglich">
```

```
    <ab>11:00:00+01:00</ab>
    <an>14:00:00+01:00</an>
   </zeit>
   <zeit rythmus="täglich">
    <ab>15:00:00+01:00</ab>
    <an>18:00:00+01:00</an>
   </zeit>
   <zeit rythmus="täglich">
    <ab>19:30:00+01:00</ab>
    <an>22:30:00+01:00</an>
   </zeit>
   <zeit rythmus="SA SO">
    <ab>09:00:00+01:00</ab>
    <an>12:00:00+01:00</an>
   </zeit>
   <zeit rythmus="FR">
    <ab>13:00:00+01:00</ab>
    <an>16:00:00+01:00</an>
   </zeit>
  </rückfahrt>
 </route>
</fahrplan>
<ausstattung>
 <catering>Snacks Kaffee </catering>
 <sitzabstand>groß</sitzabstand>
 <steckdosen>ja</steckdosen>
 <wlan>ja</wlan>
</ausstattung>
<beschreibung>
 <kurzbeschreibung>Der 2011 in Berlin gegründete Verbund MeinFernbus will nach eigenen Angaben beliebtester und
      bekanntester Anbieter von Fernbuslinien werden.</kurzbeschreibung>
 <langbeschreibung>Das Unternehmen bemüht sich um hohe Kundenorientierung. Bei MeinFernbus besonders
      hervorzuheben sind die Möglichkeit der Fahrradmitnahme,
 kostenloses Wifi und der SMS-Service bei Verspätungen.</langbeschreibung>
</beschreibung>
</fernbuslinie>
<fernbuslinie>
<anbieter>
 <name abkürzung="BLB">Berlin Linien Bus</name>
 <gründungsjahr>1947</gründungsjahr>
 <unternehmungsform>GmbH</unternehmungsform>
 <homepage>https://www.berlinlinienbus.de/</homepage>
</anbieter>
<fahrplan>
 <route routeID="rou1" buslinien="Bayern Express &amp;amp; P. Kühn Eurolines Scandinavia Touring">
  <von>Rostock</von>
  <über>Nykøbing</über>
  <nach>Kopenhagen</nach>
  <preise währung="EUR" MwSt.="inklusive">
   <sparpreise>22.00</sparpreise>
   <erwachsene>35.00</erwachsene>
   <kinder>18.00</kinder>
   <tageskarte>66.00</tageskarte>
  </preise>
  <hinfahrt>
   <zeit rythmus="täglich">
    <abfahrt>
     <stunde>10</stunde>
     <minute>25</minute>
    </abfahrt>
    <ankunft>
     <stunde>15</stunde>
     <minute>15</minute>
    </ankunft>
   </zeit>
   <zeit rythmus="täglich">
    <abfahrt>
     <stunde>14</stunde>
     <minute>40</minute>
    </abfahrt>
    <ankunft>
     <stunde>19</stunde>
     <minute>30</minute>
    </ankunft>
   </zeit>
   <zeit rythmus="täglich">
```

```
    <abfahrt>
     <stunde>19</stunde>
     <minute>10</minute>
    </abfahrt>
    <ankunft>
     <stunde>23</stunde>
     <minute>53</minute>
    </ankunft>
   </zeit>
   <zeit rythmus="MO DO FR SO">
    <abfahrt>
     <stunde>10</stunde>
     <minute>25</minute>
    </abfahrt>
    <ankunft>
     <stunde>15</stunde>
     <minute>15</minute>
    </ankunft>
   </zeit>
  </hinfahrt>
  <rückfahrt>
   <zeit rythmus="täglich">
    <ab>06:30:00+01:00</ab>
    <an>11:15:00+01:00</an>
   </zeit>
   <zeit rythmus="täglich">
    <ab>11:00:00+01:00</ab>
    <an>15:30:00+01:00</an>
   </zeit>
   <zeit rythmus="täglich">
    <ab>17:00:00+01:00</ab>
    <an>22:15:00+01:00</an>
   </zeit>
   <zeit rythmus="MO DO FR SO">
    <ab>17:00:00+01:00</ab>
    <an>22:15:00+01:00</an>
   </zeit>
  </rückfahrt>
 </route>
 <route routeID="rou2" buslinien="Bayern Express &amp;amp; P. Kühn">
  <von>Rostock</von>
  <über>Linstow</über>
  <nach>Berlin</nach>
  <preise währung="EUR" MwSt.="inklusive">
   <sparpreise>11.00</sparpreise>
   <erwachsene>26.00</erwachsene>
   <kinder>20.00</kinder>
   <tageskarte>52.00</tageskarte>
  </preise>
  <hinfahrt>
   <zeit rythmus="täglich">
    <abfahrt>
     <stunde>11</stunde>
     <minute>00</minute>
    </abfahrt>
    <ankunft>
     <stunde>14</stunde>
     <minute>00</minute>
    </ankunft>
   </zeit>
   <zeit rythmus="täglich">
    <abfahrt>
     <stunde>15</stunde>
     <minute>15</minute>
    </abfahrt>
    <ankunft>
     <stunde>18</stunde>
     <minute>30</minute>
    </ankunft>
   </zeit>
   <zeit rythmus="MO FR SA SO">
    <abfahrt>
     <stunde>16</stunde>
     <minute>45</minute>
    </abfahrt>
    <ankunft>
```

```
      <stunde>19</stunde>
      <minute>30</minute>
     </ankunft>
    </zeit>
    <zeit rythmus="MO DO FR SO">
     <abfahrt>
      <stunde>22</stunde>
      <minute>00</minute>
     </abfahrt>
     <ankunft>
      <stunde>01</stunde>
      <minute>00</minute>
     </ankunft>
    </zeit>
   </hinfahrt>
   <rückfahrt>
    <zeit rythmus="täglich">
     <ab>16:15:00+01:00</ab>
     <an>19:25:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich">
     <ab>12:00:00+01:00</ab>
     <an>14:55:00+01:00</an>
    </zeit>
    <zeit rythmus="MO FR SA SO">
     <ab>07:30:00+01:00</ab>
     <an>10:15:00+01:00</an>
    </zeit>
    <zeit rythmus="MO DO FR SO">
     <ab>07:45:00+01:00</ab>
     <an>10:40:00+01:00</an>
    </zeit>
   </rückfahrt>
  </route>
 </fahrplan>
 <ausstattung>
  <catering>nein</catering>
  <sitzabstand>nein</sitzabstand>
  <steckdosen>nein</steckdosen>
  <wlan>nein</wlan>
 </ausstattung>
 <beschreibung>
  <kurzbeschreibung>Berlin Linien Bus (BLB) ist zurzeit der Verbund mit dem größten innerdeutschen Liniennetz.</
       kurzbeschreibung>
  <langbeschreibung>Die meisten Linien führen von oder nach Berlin und werden seit vor dem Mauerfall betrieben.
  Hauptgesellschafter des Unternehmens ist ein Tochterunternehmen der Deutschen Bahn.</langbeschreibung>
 </beschreibung>
</fernbuslinie>
</fernbusverkehr>
```

Listing E.2: Quellinstanz zu Fernbusverkehr.xsd: Fernbuslinien.xml

E.3 Zielschema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="
      unqualified" version="1.1">
 <xs:annotation>
  <xs:documentation xml:lang="de">
   Übersicht über die Fernbuslinien in Deutschland Stand: 2013
  </xs:documentation>
 </xs:annotation>
 <xs:override schemaLocation="zeit.xsd">
  <xs:complexType name="zeitType">
   <xs:sequence>
    <xs:element ref="ab"/>
    <xs:element ref="an"/>
   </xs:sequence>
   <xs:attribute ref="rythmus" use="optional"/>
   <xs:attributeGroup ref="tour"/>
  </xs:complexType>
```

```
</xs:override>
<xs:element name="fernbusverkehr" type="fernbusverkehrType">
 <xs:key name="anbieternameKey">
  <xs:selector xpath="./fernbusanbieter/anbieter"/>
  <xs:field xpath="name/@akronym"/>
 </xs:key>
 <xs:key name="routenKey">
  <xs:selector xpath="./routen"/>
  <xs:field xpath="@start"/>
  <xs:field xpath="@ziel"/>
 </xs:key>
 <xs:unique name="routeKey">
  <xs:selector xpath="./routen/route"/>
  <xs:field xpath="von"/>
  <xs:field xpath="nach"/>
 </xs:unique>
 <xs:keyref name="startzielKey" refer="routeKey">
  <xs:selector xpath=".//hinfahrt/zeit"/>
  <xs:field xpath="@start"/>
  <xs:field xpath="@ziel"/>
 </xs:keyref>
 <xs:keyref name="zielstartKey" refer="routeKey">
  <xs:selector xpath=".//rückfahrt/zeit"/>
  <xs:field xpath="@ziel"/>
  <xs:field xpath="@start"/>
 </xs:keyref>
</xs:element>
<xs:complexType name="fernbusverkehrType">
 <xs:sequence>
  <xs:element ref="routen" maxOccurs="unbounded"/>
  <xs:element ref="fernbusanbieter"/>
 </xs:sequence>
 <xs:attribute ref="last_schema_change" use="required"/>
</xs:complexType>
<xs:element name="routen" type="routenType"/>
<xs:complexType name="routenType">
 <xs:sequence>
  <xs:element ref="route" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attributeGroup ref="tour"/>
</xs:complexType>
<xs:element name="route" type="routeType"/>
<xs:complexType name="routeType">
 <xs:sequence>
  <xs:element ref="von"/>
  <xs:element ref="über" minOccurs="0"/>
  <xs:element ref="nach"/>
  <xs:element ref="preise"/>
  <xs:element ref="fernbusanbieter-ref"/>
  <xs:element ref="routenfahrplan"/>
 </xs:sequence>
 <xs:attribute ref="routeID" use="required"/>
 <xs:attribute ref="buslinien" use="required"/>
</xs:complexType>
<xs:element name="von" type="xs:string"/>
<xs:element name="über" type="xs:string"/>
<xs:element name="nach" type="xs:string"/>
<xs:element name="preise" type="preiseType"/>
<xs:complexType name="preiseType">
 <xs:sequence minOccurs="0" maxOccurs="unbounded">
  <xs:element ref="sparpreise"/>
  <xs:element ref="erwachsene"/>
  <xs:element ref="kinder"/>
  <xs:element ref="gruppen"/>
  <xs:element ref="tageskarte"/>
  <xs:element ref="fahrradmitnahme"/>
 </xs:sequence>
 <xs:attribute ref="währung" use="required"/>
</xs:complexType>
<xs:element name="sparpreise" type="geldneuType"/>
<xs:simpleType name="geldneuType">
 <xs:union memberTypes="geldType fehlType"/>
</xs:simpleType>
<xs:simpleType name="geldType">
 <xs:restriction base="xs:decimal">
  <xs:fractionDigits value="2"/>
```

```
  <xs:minExclusive value="0"/>
 </xs:restriction>
</xs:simpleType>
<xs:simpleType name="fehlType">
 <xs:restriction base="xs:string">
  <xs:pattern value="keine Angabe"/>
 </xs:restriction>
</xs:simpleType>
<xs:element name="erwachsene" type="geldneuType"/>
<xs:element name="kinder" type="geldneuType"/>
<xs:element name="gruppen" type="geldneuType"/>
<xs:element name="tageskarte" type="geldneuType"/>
<xs:element name="fahrradmitnahme" type="geldneuType"/>
<xs:attribute name="währung" type="währungType"/>
<xs:simpleType name="währungType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="EUR"/>
  <xs:enumeration value="CHF"/>
 </xs:restriction>
</xs:simpleType>
<xs:element name="fernbusanbieter-ref" type="xs:IDREF"/>
<xs:element name="routenfahrplan" type="routenfahrplanType"/>
<xs:complexType name="routenfahrplanType">
 <xs:sequence>
  <xs:element ref="hinfahrt" minOccurs="0"/>
  <xs:element ref="rückfahrt" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
<xs:attribute name="routeID" type="xs:ID"/>
<xs:attribute name="buslinien" type="buslinienType"/>
<xs:simpleType name="buslinienType">
 <xs:list itemType="buslinieType"/>
</xs:simpleType>
<xs:simpleType name="buslinieType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Joost-Ostsee-Express"/>
  <xs:enumeration value="Wörlitz-Tourist"/>
  <xs:enumeration value="Bayern-Express&amp;amp;P.Kühn"/>
  <xs:enumeration value="Eurolines-Scandinavia"/>
  <xs:enumeration value="Touring"/>
 </xs:restriction>
</xs:simpleType>
<xs:attributeGroup name="tour">
 <xs:attribute ref="start" use="required"/>
 <xs:attribute ref="ziel" use="required"/>
</xs:attributeGroup>
<xs:attribute name="start" type="xs:string"/>
<xs:attribute name="ziel" type="xs:string"/>
<xs:element name="fernbusanbieter" type="fernbusanbieterType"/>
<xs:complexType name="fernbusanbieterType">
 <xs:sequence>
  <xs:element ref="anbieter" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
<xs:element name="anbieter" type="anbieterneuType"/>
<xs:complexType name="anbieterneuType">
 <xs:complexContent>
  <xs:extension base="anbieterType">
   <xs:sequence>
    <xs:element ref="ausstattung"/>
   </xs:sequence>
   <xs:attribute ref="anbieterID" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="anbieterType" final="restriction">
 <xs:sequence>
  <xs:element ref="name"/>
  <xs:element ref="gründungsjahr"/>
  <xs:element ref="unternehmungsform"/>
  <xs:element ref="homepage"/>
 </xs:sequence>
</xs:complexType>
<xs:element name="name" type="nameType"/>
<xs:complexType name="nameType">
 <xs:complexContent>
```

```
   <xs:extension base="beschreibungType">
    <xs:attribute ref="akronym" use="required"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
 <xs:attribute name="akronym" type="xs:string"/>
 <xs:element name="gründungsjahr" type="gründungsjahrType"/>
 <xs:simpleType name="gründungsjahrType">
  <xs:restriction base="xs:gYear">
   <xs:minInclusive value="1900"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="unternehmungsform" type="unternehmungsformType"/>
 <xs:simpleType name="unternehmungsformType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="GmbH"/>
   <xs:enumeration value="AG"/>
   <xs:enumeration value="KG"/>
   <xs:enumeration value="GmbH&amp;Co.KG"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="homepage" type="xs:anyURI"/>
 <xs:element name="ausstattung" type="ausstattungType" nillable="true"/>
 <xs:complexType name="ausstattungType">
  <xs:sequence>
   <xs:any namespace="http://www.ausstattung.org" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="beschreibungType" mixed="true">
  <xs:sequence>
   <xs:element ref="kurzbeschreibung"/>
   <xs:element ref="langbeschreibung" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="kurzbeschreibung" type="xs:string"/>
 <xs:element name="langbeschreibung" type="xs:string"/>
 <xs:attribute name="anbieterID" type="xs:ID"/>
 <xs:attribute name="last_schema_change" fixed="01.01.2013"/>
</xs:schema>
```

Listing E.3: Zielschema: Fernbusverkehr_Evolution.xsd

E.4 Zielinstanz

```
<?xml version="1.0" encoding="UTF-8"?>
<fernbusverkehr xmlns:extra="http://www.ausstattung.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi
      :schemaLocation="http://www.ausstattung.org ausstattung.xsd" xsi:noNamespaceSchemaLocation="
     Fernbusverkehr_evolution.xsd" last_schema_change="01.01.2013">
<routen start="Rostock" ziel="Berlin">
 <route routeID="rouRB1" buslinien="Joost-Ostsee-Express Wörlitz-Tourist">
  <von>Warnemünde</von>
  <über>Rostock</über>
  <nach>Berlin</nach>
  <preise währung="EUR">
   <sparpreise>11.00</sparpreise>
   <erwachsene>21.00</erwachsene>
   <kinder>12.50</kinder>
   <gruppen>16.50</gruppen>
   <tageskarte>35.00</tageskarte>
   <fahrradmitnahme>9.00</fahrradmitnahme>
  </preise>
  <fernbusanbieter-ref>a01</fernbusanbieter-ref>
  <routenfahrplan>
   <hinfahrt>
    <zeit rythmus="täglich" start="Warnemünde" ziel="Berlin">
     <ab>06:40:00+01:00</ab>
     <an>09:45:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Warnemünde" ziel="Berlin">
     <ab>11:10:00+01:00</ab>
     <an>14:15:00+01:00</an>
```

```
    </zeit>
    <zeit rythmus="täglich" start="Warnemünde" ziel="Berlin">
     <ab>15:10:00+01:00</ab>
     <an>18:15:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Warnemünde" ziel="Berlin">
     <ab>19:10:00+01:00</ab>
     <an>22:15:00+01:00</an>
    </zeit>
    <zeit rythmus="FR SA SO" start="Warnemünde" ziel="Berlin">
     <ab>17:10:00+01:00</ab>
     <an>20:15:00+01:00</an>
    </zeit>
   </hinfahrt>
   <rückfahrt>
    <zeit rythmus="täglich" start="Berlin" ziel="Warnemünde">
     <ab>07:00:00+01:00</ab>
     <an>10:00:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Berlin" ziel="Warnemünde">
     <ab>11:00:00+01:00</ab>
     <an>14:00:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Berlin" ziel="Warnemünde">
     <ab>15:00:00+01:00</ab>
     <an>18:00:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Berlin" ziel="Warnemünde">
     <ab>19:30:00+01:00</ab>
     <an>22:30:00+01:00</an>
    </zeit>
    <zeit rythmus="SA SO" start="Berlin" ziel="Warnemünde">
     <ab>09:00:00+01:00</ab>
     <an>12:00:00+01:00</an>
    </zeit>
    <zeit rythmus="FR" start="Berlin" ziel="Warnemünde">
     <ab>13:00:00+01:00</ab>
     <an>16:00:00+01:00</an>
    </zeit>
   </rückfahrt>
  </routenfahrplan>
 </route>
 <route routeID="rouRB2" buslinien="Bayern-Express&amp;amp;P.Kühn">
  <von>Rostock</von>
  <über>Linstow</über>
  <nach>Berlin</nach>
  <preise währung="EUR">
   <sparpreise>11.00</sparpreise>
   <erwachsene>26.00</erwachsene>
   <kinder>keine Angabe</kinder>
   <gruppen>16.50</gruppen>
   <tageskarte>52.00</tageskarte>
   <fahrradmitnahme>keine Angabe</fahrradmitnahme>
  </preise>
  <fernbusanbieter-ref>a02</fernbusanbieter-ref>
  <routenfahrplan>
   <hinfahrt>
    <zeit rythmus="täglich" start="Rostock" ziel="Berlin">
     <ab>11:00:00+01:00</ab>
     <an>14:00:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Rostock" ziel="Berlin">
     <ab>15:15:00+01:00</ab>
     <an>18:30:00+01:00</an>
    </zeit>
    <zeit rythmus="MO FR SA SO" start="Rostock" ziel="Berlin">
     <ab>16:45:00+01:00</ab>
     <an>19:30:00+01:00</an>
    </zeit>
    <zeit rythmus="MO DO FR SO" start="Rostock" ziel="Berlin">
     <ab>22:00:00+01:00</ab>
     <an>01:00:00+01:00</an>
    </zeit>
   </hinfahrt>
   <rückfahrt>
    <zeit rythmus="täglich" start="Berlin" ziel="Rostock">
```

```
    <ab>16:15:00+01:00</ab>
    <an>19:25:00+01:00</an>
   </zeit>
   <zeit rythmus="täglich" start="Berlin" ziel="Rostock">
    <ab>12:00:00+01:00</ab>
    <an>14:55:00+01:00</an>
   </zeit>
   <zeit rythmus="MO FR SA SO" start="Berlin" ziel="Rostock">
    <ab>07:30:00+01:00</ab>
    <an>10:15:00+01:00</an>
   </zeit>
   <zeit rythmus="MO DO FR SO" start="Berlin" ziel="Rostock">
    <ab>07:45:00+01:00</ab>
    <an>10:40:00+01:00</an>
   </zeit>
  </rückfahrt>
 </routenfahrplan>
</route>
</routen>
<routen start="Rostock" ziel="Kopenhagen">
 <route routeID="rouRK1" buslinien="Bayern-Express&amp;amp;P.Kühn Eurolines-Scandinavia Touring">
  <von>Rostock</von>
  <über>Nykøbing</über>
  <nach>Kopenhagen</nach>
  <preise währung="EUR">
   <sparpreise>22.00</sparpreise>
   <erwachsene>35.00</erwachsene>
   <kinder>18.00</kinder>
   <gruppen>keine Angabe</gruppen>
   <tageskarte>66.00</tageskarte>
   <fahrradmitnahme>keine Angabe</fahrradmitnahme>
  </preise>
  <fernbusanbieter-ref>a02</fernbusanbieter-ref>
  <routenfahrplan>
   <hinfahrt>
    <zeit rythmus="täglich" start="Rostock" ziel="Kopenhagen">
     <ab>10:25:00+01:00</ab>
     <an>15:15:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Rostock" ziel="Kopenhagen">
     <ab>14:40:00+01:00</ab>
     <an>19:30:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Rostock" ziel="Kopenhagen">
     <ab>19:10:00+01:00</ab>
     <an>23:53:00+01:00</an>
    </zeit>
    <zeit rythmus="MO DO FR SO" start="Rostock" ziel="Kopenhagen">
     <ab>10:25:00+01:00</ab>
     <an>15:15:00+01:00</an>
    </zeit>
   </hinfahrt>
   <rückfahrt>
    <zeit rythmus="täglich" start="Kopenhagen" ziel="Rostock">
     <ab>06:30:00+01:00</ab>
     <an>11:15:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Kopenhagen" ziel="Rostock">
     <ab>11:00:00+01:00</ab>
     <an>15:30:00+01:00</an>
    </zeit>
    <zeit rythmus="täglich" start="Kopenhagen" ziel="Rostock">
     <ab>17:00:00+01:00</ab>
     <an>22:15:00+01:00</an>
    </zeit>
    <zeit rythmus="MO DO FR SO" start="Kopenhagen" ziel="Rostock">
     <ab>17:00:00+01:00</ab>
     <an>22:15:00+01:00</an>
    </zeit>
   </rückfahrt>
  </routenfahrplan>
 </route>
</routen>
<fernbusanbieter>
 <anbieter anbieterID="a01">
```

```
  <name akronym="MFB">MeinFernbus<kurzbeschreibung>Der 2011 in Berlin gegründete Verbund MeinFernbus will nach
       eigenen Angaben beliebtester und bekanntester Anbieter von Fernbuslinien werden.</kurzbeschreibung></name>
  <gründungsjahr>2011</gründungsjahr>
  <unternehmungsform>GmbH</unternehmungsform>
  <homepage>http://meinfernbus.de/</homepage>
  <ausstattung>
   <extra:catering>Snacks Kaffee </extra:catering>
   <extra:sitzabstand>groß</extra:sitzabstand>
   <extra:steckdosen>ja</extra:steckdosen>
   <extra:wlan>ja</extra:wlan>
  </ausstattung>
 </anbieter>
 <anbieter anbieterID="a02">
  <name akronym="BLB">Berlin Linien Bus<kurzbeschreibung>Berlin Linien Bus (BLB) ist zurzeit der Verbund mit dem
       größten innerdeutschen Liniennetz.</kurzbeschreibung></name>
  <gründungsjahr>1947</gründungsjahr>
  <unternehmungsform>GmbH</unternehmungsform>
  <homepage>https://www.berlinlinienbus.de/</homepage>
  <ausstattung xsi:nil="true"/>
 </anbieter>
</fernbusanbieter>
</fernbusverkehr>
```

Listing E.4: Zielinstanz zu Fernbusverkehr_Evolution.xsd: Fernbuslinien_Evolution.xml

E.5 XML-Schema Zeit

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="
     unqualified">
 <xs:annotation>
  <xs:documentation xml:lang="de">
    Beschreibt die Hinfahrts- und Rückfahrtszeiten
  </xs:documentation>
 </xs:annotation>
 <xs:element name="hinfahrt" type="pendlerType"/>
 <xs:complexType name="pendlerType">
  <xs:sequence>
   <xs:element ref="zeit" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="zeit" type="zeitType"/>
 <xs:complexType name="zeitType">
  <xs:choice>
   <xs:sequence>
    <xs:element ref="abfahrt"/>
    <xs:element ref="ankunft"/>
   </xs:sequence>
   <xs:sequence>
    <xs:element ref="ab"/>
    <xs:element ref="an"/>
   </xs:sequence>
  </xs:choice>
  <xs:attribute ref="rythmus" use="optional"/>
 </xs:complexType>
 <xs:element name="abfahrt" type="uhrType"/>
 <xs:complexType name="uhrType">
  <xs:sequence>
   <xs:element ref="stunde"/>
   <xs:element ref="minute"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="stunde" type="xs:integer"/>
 <xs:element name="minute" type="xs:integer"/>
 <xs:element name="ankunft" type="uhrType"/>
 <xs:element name="ab" type="xs:time"/>
 <xs:element name="an" type="xs:time"/>
 <xs:attribute default="täglich" name="rythmus" type="rythmusType"/>
 <xs:simpleType name="rythmusType">
  <xs:list itemType="tagType"/>
 </xs:simpleType>
```

```
 <xs:simpleType name="tagType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="MO"/>
   <xs:enumeration value="DI"/>
   <xs:enumeration value="MI"/>
   <xs:enumeration value="DO"/>
   <xs:enumeration value="FR"/>
   <xs:enumeration value="SA"/>
   <xs:enumeration value="SO"/>
   <xs:enumeration value="täglich"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="rückfahrt" type="pendlerType"/>
</xs:schema>
```

Listing E.5: XML-Schema Zeit: zeit.xsd

E.6 XML-Schema Ausstattung

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.ausstattung.org" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://
     www.ausstattung.org" elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:annotation>
  <xs:documentation xml:lang="de">
    Beschreibt verschiedene Ausstattungsmöglichkeiten von Reisebussen
  </xs:documentation>
 </xs:annotation>
 <xs:element name="catering" type="cateringType"/>
 <xs:simpleType name="cateringType">
  <xs:list itemType="serviceType"/>
 </xs:simpleType>
 <xs:simpleType name="serviceType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="Snacks"/>
   <xs:enumeration value="Kaffee"/>
   <xs:enumeration value="Kaltgetränke"/>
   <xs:enumeration value="nein"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="sitzabstand" type="sitzabstandType"/>
 <xs:simpleType name="sitzabstandType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="groß"/>
   <xs:enumeration value="klein"/>
   <xs:enumeration value="nein"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="steckdosen" type="wahlType"/>
 <xs:simpleType name="wahlType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="ja"/>
   <xs:enumeration value="nein"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="wlan" type="wahlType"/>
</xs:schema>
```

Listing E.6: XML-Schema Ausstattung: ausstattung.xsd

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[1] B. Alexe, L. Chiticariu, R. J. Miller, and W.-C. Tan. Muse: Mapping Understanding and deSign by Example. In *ICDE*, pages 10–19, 2008.

[2] B. Alexe, B. ten Cate, P.G. Kolaitis, and W.-C. Tan. EIRENE: Interactive Design and Refinement of Schema Mappings via Data Examples. *VLDB 4(12)*, 2011.

[3] A. Algergawy. Management of XML data by means of schema matching. Dissertation, Universität Magdeburg, 2010.

[4] A. Algergawy, R. Nayak, and G. Saake. XML Schema Element Similarity Measures: A Schema Matching Context. In *OTM*, pages 1246–1253. Springer, 2009.

[5] A. Algergawy, R. Nayak, and G. Saake. Element similarity measures in XML schema matching. *Information Sciences 180(24)*, pages 4975–4998, 2010.

[6] A. Algergawy, E. Schallehn, and G. Saake. A Schema Matching-based Approach to XML Schema Clustering. In *iiWAS*, pages 131–136, 2008.

[7] A. Algergawy, E. Schallehn, and G. Saake. Improving XML schema matching performance using Prüfer sequences. *Data Knowl. Eng. 68(8)*, pages 728–747, 2009.

[8] Y. An, A. Borgida, and J. Mylopoulos. Discovering and Maintaining Semantic Mappings between XML Schemas and Ontologies. *J. Computing Sci. Eng 2(1)*, pages 44–73, 2008.

[9] M. Arenas, J. Pérez, and C. Riveros. The Recovery of a Schema Mapping: Bringing Exchanged Data Back. *ACM TODS 34(4)*, pages 22–32, 2009.

[10] P. Arnold. *COMA 3.0 CE Program Description*. University of Leipzig, 2012.

[11] P.C. Arocena, A. Fuxman, and R. J. Miller. Composing Local-As-View Mappings: Closure and Applications. In *ICDT*, pages 209–218, 2010.

[12] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and Ontology Matching with COMA++. In *SIGMOD*, pages 906–908, 2005.

[13] F. Ayaz. *XML und XSLT mit C++*. mitp, 2006.

[14] M. Beg, L. Charlin, and J. So. MAXSM: A Multi-Heuristic Approach to XML Schema Matching. Technical report, University of Waterloo, 2006.

[15] Z. Bellahsene and F. Duchateau. Tuning for Schema Matching. In Z. Bellahsene, A. Bonifati, and E. Rahm, editors, *Schema Matching and Mapping*, pages 293–316. Springer, 2011.

[16] P. A. Bernstein, S. Melnik, and J. E. Churchill. Incremental Schema Matching. In *VLDB*, pages 1167–1170, 2006.

[17] P.A. Bernstein, J. Madhavan, and E. Rahm. Generic Schema Matching, Ten Years Later. In *VLDB*, pages 695–701, 2011.

[18] P.A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-Strength Schema Matching. In *SIGMOD*, pages 38–43, 2004.

[19] P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes W3C Recommendation 5 April 2012. Technical report, W3C, `http://www.w3.org/TR/xmlschema11-2/`, 2012.

[20] P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster. Putting Context into Schema Matching. In *VLDB*, pages 307–318, 2006.

[21] A. Bonifati, A. Mecca, G.and Pappalardo, S. Raunich, and G. Summa. Schema Mapping Verification: The Spicy Way. In *EDBT*, pages 85–96, 2008.

[22] A. Bonifati, A. Mecca, G.and Pappalardo, S. Raunich, and G. Summa. The Spicy System: Towards a Notion of Mapping Quality. In *SIGMOD*, pages 1289–1294, 2008.

[23] S. Bossung, H. Stoeckle, J. Grundy, R. Amor, and J. Hosking. Automated Data Mapping Specifications via Schema Heuristics and User Interaction. In *ASE*, pages 208–217, 2004.

[24] A. Boukottaya and C. Vanoirbeek. Schema Matching for Transforming Structured Documents. In *DocEng*, pages 101–110, 2005.

[25] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). Technical report, W3C, http://www.w3.org/TR/xml/, 2008.

[26] T. Brinkhoff, O. Herden, H. Knolle, K. Kudraß, T. Meyer-Wegener, T. Rakow, N. Ritter, K.-U. Sattler, P. Sauer, B. Schiefer, H. Schöning, and C. Türker. *Taschenbuch Datenbanken.* Fachbuchverlag Leipzig im Carl Hanser Verlag, 2007.

[27] N. Chen, J. He, W. Wang, and Z. Chen. Extended FRAG-BASE schema-matching method for multi-version open GIS Web services retrieval. *International Journal of Geographical Information Science 25(7)*, pages 1045–1068, 2011.

[28] N. Chen, J. He, C. Yang, and C. Wang. A node semantic similarity schema-matching method for multi-version Web Coverage Service retrieval. *International Journal of Geographical Information Science 26(6)*, pages 1051–1072, 2012.

[29] Y.-P. P. Chen, S. Promparmote, and F. Maire. MDSM: Microarray database schema matching using the Hungarian method. *Information Sciences 176(19)*, pages 2771–2790, 2006.

[30] L. Chiticariu, M. A. Hernández, P. G. Kolaitis, and L. Popa. Semi-Automatic Schema Integration in Clio. In *VLDB*, pages 1326–1329, 2007.

[31] K. T. Claypool, V. Hegde, and N. Tansalarak. QMatch - A Hybrid Match Algorithm for XML Schemas. In *ICDE*, pages 1281–1291, 2005.

[32] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *IIWeb*, pages 73–78, 2003.

[33] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A Comparison of String Metrics for Matching Names and Records. In *KDD Workshop on Data Cleaning and Object Consolidation*, 2003.

[34] I. F. Cruz, F. P. Antonelli, and C. Stroe. Efficient Selection of Mappings and Automatic Quality-driven Combination of Matching Methods. In *OM*, pages 49–60, 2009.

[35] I. F. Cruz and W. Sunna. Structural Alignment Methods with Applications to Geospatial Ontologies. *Transactions in GIS 12(6)*, pages 683–711, 2008.

[36] I. F. Cruz, W. Sunna, N. Makar, and S. Bathala. A visual tool for ontology alignment to enable geospatial interoperability. *Journal of Visual Languages & Computing (18)*, (3):230–254, 2007.

[37] I.F. Cruz, F.P. Antonelli, and C. Stroe. AgreementMaker: Efficient Matching for Large Real-World Schemas and Ontologies. In *VLDB*, pages 1586–1589, 2009.

[38] J. Deffke. XML-Schema Evolution: Evolution in der Praxis. Bachelorarbeit, Universität Rostock, 2012.

[39] R. Dhamankar, Y. Lee, A-H. Doan, A. Y. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *SIGMOD*, pages 383–394, 2004.

[40] H. H. Do, S. Melnik, and E. Rahm. Comparison of Schema Matching Evaluations. In *Web, Web-Services, and Database Systems*, pages 221–237. Springer, 2003.

[41] H. H. Do and E. Rahm. COMA - A system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.

[42] A-H. Doan, P. Domingos, and A. Y. Halevy. Reconciling the Schemas of Disparate Data Sources: A Machine-Learning Approach. In *SIGMOD*, pages 509–520, 2001.

[43] F. Duchateau, Z. Bellahsene, and R. Coletta. A Flexible Approach for Planning Schema Matching Algorithms. In *OTM*, pages 249–264. Springer, 2008.

[44] F. Duchateau, R. Coletta, Z. Bellahsene, and R. J. Miller. YAM: a Schema Matcher Factory. In *CIKM*, pages 2079–2080, 2009.

[45] M. Ehrig and S. Staab. QOM - Quick Ontology Mapping. In *Int. Conf. Semantic Web (ICSW)*, pages 683–697, 2004.

[46] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate Record Detection: A Survey. *TKDE 19(1)*, pages 1–16, 2007.

[47] H. Elmeleegy, M. Ouzzani, and A.K. Elmagarmid. Usage-Based Schema Matching. In *ICDE*, pages 20–29, 2008.

[48] D. Engmann and S. Massmann. Instance Matching with COMA++. 2007.

[49] R. Fagin, L. M. Haas, M. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema Mapping Creation and Data Exchange. Technical report, IBM Almaden Research Center; University of Toronto and University of Trento, 2009.

[50] S.M. Falconer and N.F. Noy. Interactive Techniques to Support Ontology Matching. In Z. Bellahsene, A. Bonifati, and E. Rahm, editors, *Schema Matching and Mapping*, pages 29–52. Springer, 2011.

[51] L. Feiyu. State of the Art : Automatic Ontology Matching. Technical report, School of Engineering, Jönköping University, 2007.

[52] A. Gal. Managing Uncertainty in Schema Matching with Top-K Schema Mappings. *Data Semantics 6*, pages 90–114, 2006.

[53] A. Gal, T. Sagi, M. Weidlich, E. Levy, V. Shafran, Z. Miklós, and N. Q. V. Hung. Making Sense of Top-K Matchings: A Unified Match Graph for Schema Matching. In *IIWeb*, IIWeb '12, pages 6:1–6:6, 2012.

[54] F. Giunchiglia and P. Shvaiko. Semantic matching. Technical report, University of Trento, 2003.

[55] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic schema matching. In *OTM Conf. (CoopIS)*, pages 347–365, 2005.

[56] F. Giunchiglia, M. Yatskevich, and P. Shvaiko. Semantic Matching: Algorithms and Implementation. In *Journal on Data Semantics IX*, pages 1–38. Springer, 2007.

[57] Altova GmbH. *Benutzer- und Referenzhandbuch Altova® DiffDog® 2013 Enterprise Edition*, 2012.

[58] Altova GmbH. *Benutzer- und Referenzhandbuch MapForce® 2013 Enterprise Edition*, 2012.

[59] Altova GmbH. Vergleich von XML-Schemas, `http://www.altova.com/de/diffdog/xml-schema-diff-tool.html`, 2013.

[60] Altova GmbH. MapForce - grafisches Tool für Datenmapping, -konvertierung und -integration, `http://www.altova.com/de/mapforce.html`, 2013.

[61] Altova GmbH. RaptorXML, `http://www.altova.com/de/raptorxml.html`, 2013.

[62] J. Gong, R. Cheng, and D. W. Cheung. Efficient management of uncertainty in XML schema matching. *The VLDB Journal 21(3)*, pages 385–409, 2012.

[63] A. Gross, M. Hartung, T. Kirsten, and E. Rahm. On Matching Large Life Science Ontologies in Parallel. In *7th Int. Conf. Data Integration in the Life Sciences (DILS)*, pages 35–49, 2010.

[64] M. Hartung, J. Terwilliger, and E. Rahm. Recent Advances in Schema and Ontology Evolution. In *Schema Matching and Mapping*, pages 149–190. Springer, 2011.

[65] B. He and K. C.-C. Chang. Statistical Schema Matching across Web Query Interfaces. In *SIGMOD*, pages 217–228, 2003.

[66] B. He and K. C.-C. Chang. Automatic complex schema matching across Web query interfaces: A correlation mining approach. *ACM Trans. Database Syst. 31(1)*, pages 346–395, 2006.

[67] H. He, W. Meng, C. T. Yu, and Z. Wu. Automatic integration of Web search interfaces with WISE-Integrator. In *VLDB*, pages 256–273, 2004.

[68] W. Hu, Y. Qu, and G. Cheng. Matching large ontologies: A divide-and-conquer-approach. *Data Knowl. Eng. 67(1)*, pages 140–160, 2008.

[69] IBM. Datenbanken entwerfen und modellieren: Zuordnungseditor, `http://publib.boulder.ibm.com/infocenter/idm/v2r1/index.jsp?topic=/com.ibm.datatools.metadata.mapping.ui.doc/topics/iiymdmslexamp.html`, 2013.

[70] J. Kang and J. F. Naughton. Schema Matching Using Interattribute Dependencies. *IEEE Trans. Knowl. Data Eng. 20(10)*, pages 1393–1407, 2008.

[71] C. Kaping. Transformation von Modellierungsstilen. Bachelorarbeit, Universität Rostock, 2013.

[72] A. Karcher. Die eXtensible Markup Language XML. Skript zur Vorlesung "Einführung in die Wirtschaftsinformatik 3", Universität der Bundeswehr München, Frühjahrs-Trimester 2003.

[73] Z. Kedad and X. Xue. Mapping discovery for XML data integration. In *OTM*, pages 166–182, 2005.

[74] Y. O. Kılıç and M. N. Aydin. Automatic XML Schema Matching. In *EMCIS*, 2009.

[75] J. Kim, Y. Peng, N. Ivezic, and J. Shin. Semantic-based Optimal XML Schema Matching. In *ICEME*, 2010.

[76] J. Kim, Y. Peng, N. Ivezic, and J. Shin. An Optimization Approach for Semantic-based XML Schema Matching. *International Journal of Trade, Economics and Finance 2(1)*, pages 78–86, 2011.

[77] M. Klettke. Conceptual XML Schema Evolution - the CoDEX approach for Design and Redesign. Technical report, Universität Greifswald, 2007.

[78] M. Klettke. Modellierung, Bewertung und Evolution von XML-Dokumentkollektionen. Habilitation, Universität Rostock, 2007.

[79] J. Klímek. XML schema evolution. Diplomarbeit, Charles University in Prague, 2009.

[80] H. Köhler, X. Zhou, S. Sadiq, Y. Shu, and K. Taylor. Sampling Dirty Data for Matching Attributes. In *SIGMOD*, pages 63–74, 2010.

[81] M. Kwietniewski, J. Gryz, S. Hazlewood, and P. Van Run. Transforming XML Documents as Schemas Evolve. *VLDB 3(1-2)*, pages 1577–1580, 2010.

[82] M.L. Lee, L.H. Yang, W. Hsu, and X. Yang. XClust: Clustering XML Schemas for Effective Integration. In *CIKM*, pages 292–299, 2002.

[83] Y. Lee, M. Sayyadian, A. Doan, and A. Rosenthal. eTuner: tuning schema matching software using synthetic scenarios. In *VLDB*, pages 97–122, 2007.

[84] F. Legler and F. Naumann. A Classification of Schema Mappings and Analysis of Mapping Tools. In *BTW*, 2007.

[85] J. Li, J. Tang, Y. Li, and Q. Luo. RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. *IEEE Trans. Knowl. Data Eng. 21(8)*, pages 1218–1232, 2009.

[86] J. Lu, S. Wang, and J. Wang. An Experiment on the Matching and Reuse of XML Schemas. In *ICWE*, pages 273–284, 2005.

[87] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid - extended version. In *VLDB*, pages 49–58, 2001.

[88] J. Madhavan, P.A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based Schema Matching. In *ICDE*, pages 57–68, 2005.

[89] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *VLDB*, pages 49–58, 2001.

[90] M. Magnani, N. Rizopoulos, P. McBrien, and D. Montesi. Schema Integration based on Uncertain Semantic Mappings. In *ER*, pages 31–45, 2005.

[91] E. Maler. Schema Design Rules for UBL..and Maybe for You. In *XML 2002 Proceedings by deepX*, 2002.

[92] B. Marnette, G. Mecca, and P. Papotti. Scalable Data Exchange with Functional Dependencies. *VLDB 3(1-2)*, pages 105–116, 2010.

[93] B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro. ++Spicy: an Open-Source Tool for Second-Generation Schema Mapping and Data Exchange. *Clio 19(21)*, 2011.

[94] S. Massmann. Analyse der Wiederverwendung zum Schema-Matching. Diplomarbeit, Universität Leipzig, 2005.

[95] S. Massmann and E. Rahm. Evaluating Instance-based Matching of Web Directories. In *WebDB*, 2008.

[96] S. Massmann, S. Raunich, D. Aumueller, P. Arnold, and E. Rahm. Evolution of the COMA Match System. In *Ontology Matching 6*, 2011.

[97] R. McCann, W. Shen, and A. Doan. Matching Schemas in Online Communities: A Web 2.0 Approach. In *ICDE*, pages 110–119, 2008.

[98] G. Mecca and P. Papotti. Schema Mapping and Data Exchange Tools: Time for the Golden Age. *it-Information Technology 54(3)*, pages 105–113, 2012.

[99] G. Mecca, P. Papotti, and S. Raunich. Core Schema Mappings. In *SIGMOD*, pages 655–668, 2009.

[100] G. Mecca, P. Papotti, and S. Raunich. Core schema mappings: Scalable core computations in data exchange. *Information Systems*, 2012.

[101] G. Mecca, P. Papotti, S. Raunich, and M. Buoncristiano. Concise and Expressive Mappings with +Spicy. *VLDB 2(2)*, pages 1582–1585, 2009.

[102] M.M. Meijer. On a method for XML Schema matching. Technical report, University of Twente, 2008.

[103] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm (Extended Technical Report). Technical report, Stanford, 2001.

[104] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *ICDE*, pages 117–128, 2002.

[105] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A Programming Platform for Model Management. In *SIGMOD*, pages 193–204, 2003.

[106] P. Mork, L. Seligman, A. Rosenthal, J. Korb, and C. Wolf. The Harmony Integration Workbench. In *Journal on Data Semantics XI*, pages 65–93. 2008.

[107] O. Motschiunigg. Evaluierung von Clio zur Transformation von Metamodellen. Diplomarbeit, Universität Wien, 2008.

[108] A. Nandi and P.A. Bernstein. HAMSTER: Using Search Clicklogs for Schema and Taxonomy Matching. In *VLDB*, pages 181–192, 2009.

[109] T. Nösinger. ELaX (Evolution Language for XML-Schema), `www.ls-dbis.de/elax`, 2013.

[110] T. Nösinger, M. Klettke, and A. Heuer. Evolution von XML-Schemata auf konzeptioneller Ebene - Übersicht: Der CodeX-Ansatz zur Lösung des Gültigkeitsproblems. In *Grundlagen von Datenbanken*, pages 29–34, 2012.

[111] T. Nösinger, M. Klettke, and A. Heuer. Automatisierte Modelladaptionen durch Evolution -RELaX in the Garden of Eden. Technical report, Universität Rostock, 2013.

[112] T. Nösinger, M. Klettke, and A. Heuer. XML Schema Transformations - The ELaX Approach. In *DEXA*, pages 293–302, 2013.

[113] D. Pansch. Verfahren zur Datenintegration strukturell heterogener Quellen im Kontext der eHumanities. Bachelorarbeit, Universität Leipzig, 2010.

[114] E. Peukert, H. Berthold, and E. Rahm. Rewrite Techniques for Performance Optimization of Schema Matching Processes. In *EDBT*, pages 433–464, 2010.

[115] E. Peukert, J. Eberius, and E. Rahm. AMC - A Framework for Modelling and Comparing Matching Systems as Matching Processes. In *ICDE*, pages 1304–1307, 2011.

[116] E. Peukert, J. Eberius, and E. Rahm. A Self-Configuring Schema Matching System. In *ICDE*, pages 306–317, 2012.

[117] E. Peukert, S. Massmann, and K. Koenig. Comparing Similarity Combination Methods for Schema Matching. In *GI-Jahrestagung*, volume 175, pages 692–701, 2010.

[118] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.

[119] L. Qian, M. J. Cafarella, and H. V. Jagadish. Sample-Driven Schema Mapping. In *SIGMOD*, pages 73–84, 2012.

[120] A. Radwan, L. Popa, I. R. Stanoi, and A. Younis. Top-K Generation of Integrated Schemas Based on Directed and Weighted Correspondences. In *SIGMOD*, pages 641–654, 2009.

[121] A. Raffio, D. Braga, S. Ceri, P. Papotti, and M. A. Hernandez. Clip: a Visual Language for Explicit Schema Mappings. In *ICDE*, pages 30–39, 2008.

[122] E. Rahm. Towards Large-Scale Schema and Ontology Matching. In Z. Bellahsene, A. Bonifati, and E. Rahm, editors, *Schema Matching and Mapping*, pages 3–28. Springer, 2011.

[123] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. In *VLDB*, pages 334–350, 2001.

[124] E. Rahm, H. H. Do, and S. Massmann. Matching Large XML Schemas. *ACM SIGMOD Record 33(4)*, pages 26–31, 2004.

[125] A. Rajesh and S. K. Srivatsa. XML Schema Matching - Using Structural Information. *International Journal of Computer Applications 8(2)*, pages 34–41, 2010.

[126] G.G. Robertson, M.P. Czerwinski, and J. E. Churchill. Visualization of Mappings Between Schemas. In *SIGCHI*, pages 431–439, 2005.

[127] B. Saha, I. Stanoi, and K.L. Clarkson. Schema Covering: a Step Towards Enabling Reusein Information Integration. In *ICDE*, pages 285–296, 2010.

[128] K. Saleem, Z. Bellahsene, and E. Hunt. Performance Oriented Schema Matching. In *Database and Expert Systems Applications*, pages 844–853, 2007.

[129] K. Saleem, Z. Bellahsene, and E. Hunt. PORSCHE: Performance ORiented SCHEma Mediation. *Information Systems 33(7-8)*, pages 637–657, 2008.

[130] K. Saruladha, G. Aghila, and B. Sathiya. A Comparative Analysis of Ontology and Schema Matching Systems. *International Journal of Computer Applications 34(8)*, pages 14–21, 2011.

[131] H. Schöning. *XML und Datenbanken Konzepte und Systeme*. Hanser, 2003.

[132] L. Seligman, P. Mork, A.Y. Halevy, K. Smith, M. J. Carey, K. Chen, C. Wolf, J. Madhavan, A. Kannan, and D. Burdick. OpenII: An Open Source Information Integration Toolkit. In *SIGMOD*, pages 1057–1060, 2010.

[133] P. Shvaiko. A Classification of Schema-Based Matching Approaches. *Data Semantics 6*, pages 146–171, 2005.

[134] P. Shvaiko and P. Euzenat. A Survey of Schema-based Matching Approaches. *Journal on data Semantics, IV*, pages 146–171, 2005.

[135] M. Smiljanic. XML schema matching balancing efficiency and effectiveness by means of clustering. Dissertation, University Twente, 2006.

[136] K. Smith, P. Mork, L. Seligman, P. Leveille, B. Yost, M. Li, and C. Wolf. Unity: Speeding the Creation of Community Vocabularies for Information Integration and Reuse. In *IRI*, pages 129–135, 2011.

[137] C. Stroe. The AgreementMaker User Interface, `http://agreementmaker.org/wiki/index.php/User_Interface`, 2013.

[138] H. Stuckenschmidt, J. Noessner, and F. Fallahi. A Study in User-centric Data Integration. In *ICEIS*, pages 5–14, 2012.

[139] W. Su, J. Wang, and F.H. Lochovsky. Holistic Schema Matching for Web Query Interface. In *EDBT*, pages 77–94, 2006.

[140] N. Tansalarak and K. T. Claypool. QMatch - Using Paths to Match XML Schemas. *Data & Knowledge Engineering 60(2)*, pages 260–282, 2007.

[141] J. Tekli, , R. Chbeir, and K. Yetongnon. An overview on XML similarity: Background, current trends and future directions. *Computer Science Review 3(3)*, pages 151–173, 2009.

[142] J. Tekli, R. Chbeir, and K. Yetongnon. Extensible User-based XML Grammar Matching. In *Conceptual Modeling-ER*, pages 294–314. Springer, 2009.

[143] H. Q. Thang and V. S. Nam. XML Schema Automatic Matching Solution. *International Journal of Electrical, Computer, and Systems Engineering 4(1)*, pages 68–74, 2010.

[144] H.S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures W3C Recommendation 5 April 2012. Technical report, W3C, `http://www.w3.org/TR/xmlschema11-1/`, 2012.

[145] P. T. T. Thuy, Y.-K. Lee, and S. Lee. Semantic and structural similarities between XML Schemas for integration of ubiquitous healthcare data. *Personal and Ubiquitous Computing*, pages 1–9, 2012.

[146] T. Tiedt. Schemaevolution und Adaption von XML-Dokumenten und XQuery-anfragen. Diplomarbeit, Universität Rostock, 2005.

[147] C. Türker. XML und Datenbanken. Skript zur Vorlesung "XML und Datenbank", Eidgenössische Technische Hochschule Zürich, Wintersemester 2004/2005.

[148] Y. Velegrakis, R. J. Miller, and L. Popa. Preserving mapping consistency under schema changes. *The VLDB Journal 13(3)*, pages 274–293, 2004.

[149] E. Wilson, S. Vibhute, C. Bhatia, R. Jain, L. Perniu, S. Raveendramurthy, and R. Samuel. *Getting Started with InfoSphere Data Architect.* DB2 on Campus book series, 2011.

[150] M. Wimmer, M. Seidl, P. Brosch, H. Kargl, and G. Kappel. On Realizing a Framework for Self-tuning Mappings. In *Objects, Components, Models and Patterns*, pages 1–16. Springer, 2009.

[151] H. Wu, T. W. Ling, G. Dobbie, Z. Bao, and L. Xu. Reducing graph matching to tree matching for XML queries with ID references. In *Database and Expert Systems Applications*, pages 391–406, 2010.

[152] Z. Yang, J. Yu, and M. Kitsuregawa. Fast Algorithms for Top-k Approximate String Matching. In *AAAI*, pages 1467–1473, 2010.

[153] C. Yu and L. Popa. Semantic Adaptation of Schema Mappings when Schemas Evolve. In *VLDB*, pages 1006–1017, 2005.

[154] V. Zhdanova and P. Shvaiko. Community-Driven Ontology Matching. In *ESWC*, pages 34–49, 2006.

Thesen

1. Großer Pool zur Auswahl von Matching- und Mappingalgorithmen. Dabei ist die richtige Auswahl der Matcher und Bestimmung der Ablaufreihenfolge nicht trivial.
2. Komplett automatisierte Matching- und Mappingansätze basieren aktuell auf Integration der Vorteile mehrerer Tools. Können bisher ohne Nutzerinvolvierung nicht alle Mappings korrekt erkennen.
3. Matching- und Mappingverfahren wurden in vielen Forschungsprototypen und kommerziellen Systemen implementiert. Im XML-Bereich variieren die Ansätze sehr stark nach Anwendungsfall, sodass keine allumfassende Lösung existiert.
4. Eine Visualisierung automatisch generierter Mappings helfen dem Nutzer dabei, Fehler im Mappingergebnis schneller aufzufinden und zu beheben. Werden zudem die generierten Evolutionsoperationen eingeblendet, dann erhöht dies die Verständlichkeit der Mappings und die Produktivität des Nutzers wird weiter gesteigert.
5. Die automatische Generierung von Evolutionsoperationen, insbesondere bei manueller Einführung bzw. Veränderung von Mappings, erleichtert die Anpassung auch ohne Kenntnis der Evolutionssprache.
6. Eine falsche Reihenfolge der generierten Evolutionsoperationen führt zu einem Misserfolg der Schemaevolution.