JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von
Isabella Spörl

Angefertigt am
Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

Beurteiler / Beurteilerin
o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl

Mitbetreuung
Dr. Christoph Schütz

Monat Jahr
Februar 2016

Erstellung, Verwaltung und Anpassung von Referenzmodellen für Data Warehouses unter Verwendung von Indyco Builder, BaseX und XQuery

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science

im Masterstudium

Wirtschaftsinformatik

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

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, am 25. Februar 2016

Isabella Spörl

Inhaltsverzeichnis

Abbildungsverzeichnis

Listenverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Verwendung von Referenzmodellen im Data Warehousing bietet vielerlei Vorteile und wird in Fachkreisen seit einigen Jahren häufig diskutiert. Schütz u. a. (2016) weisen darauf hin, dass besonders kleine und mittlere Unternehmen (KMUs) dadurch beim Einstieg in die Arbeit mit Data Warehouses und Business Intelligence (BI) Software im Allgemeinen unterstützt werden.

Verbesserung von Datenanalyse und Informationsmanagement, Unterstützung bei Entscheidungsprozessen und Kostenersparnisse stellen nach Scholz u. a. (2010) bedeutende Vorteile der Verwendung von Business Intelligence Systemen in KMUs dar. Scholz u. a. (2010) zeigen aber auch auf, dass diesen Vorteilen Schwierigkeiten bei der Benutzung solcher Werkzeuge, mangelnde Qualität von Daten und der Software selbst sowie die Einbindung von Business Intelligence in bestehende Systeme, zum Beispiel in Bezug auf Schnittstellen, sowie Datenkonflikte als Herausforderungen gegenüberstehen können. Weiters müssen laut Olszak und Ziemba (2012) auch die Kosten für Software, falsche Erwartungshaltungen der Benutzer und mangelnde Unterstützung durch die Unternehmensleitung bei der Einführung von Business Intelligence Systemen in KMUs berücksichtigt werden.

Gemäß Becker und Knackstedt (2004) steigern Referenzmodelle als „Ausgangslösungen für projektspezifische Realisierungen“ sowohl Effizienz als auch Effektivität von Data Warehousing. Fettke und Vom Brocke (2013) definieren dabei Referenzmodelle als Modelle, die „zur Konstruktion weiterer Modelle wiederverwendet“ werden, oder zumindest mit dieser Intention entwickelt wurden. Diese Referenzmodelle sollten laut Goeken und Knackstedt (2007) bereits bei der Gestaltung des Data Warehouse eingesetzt werden. Nach Knackstedt und Klose (2005) spielt die Konfigurierbarkeit der Referenzmodelle dabei eine bedeutende Rolle, da erst durch die Anpassung von Referenzmodellen auf konkrete Bedürfnisse ein optimaler Nutzen entsteht.

Die Verwendung von Referenzmodellen im Data Warehousing unterstützt nach Schütz u. a. (2016) jedoch nicht nur kleine und mittlere Betriebe, sondern durch Vereinheitlichung und Wiederverwendbarkeit auch internationale Konzerne. Weiters tragen Referenzmodelle zur Vereinheitlichung von Fachbegriffen und Leistungskennzahlen bei, wie beispielsweise bei Diamantini u. a. (2014).

Die vorliegende Arbeit beschreibt eine Implementierung basierend auf dem BIRD-Ansatz (Business Intelligence Reference Modeling for Data Warehouses) von Schütz u. a. (2016), der in Abschnitt 2.1 näher beschrieben wird. Die Verwendung von BIRD mithilfe dieser Implementierung erleichtert nicht nur die Erstellung, sondern auch die Verwaltung und Anpassung von Referenzmodellen für Data Warehouses. Für die Implementierung werden die Anwendung Indyco Builder der Firma Iconsulting (www.indyco.com) sowie BaseX (www.basex.org) in Verbindung mit XQuery verwendet. Näheres zu den verwendeten Technologien findet sich in Abschnitt 3 und Abschnitt 4.

Im Weiteren ist die vorliegende Arbeit folgendermaßen aufgebaut: Während Abschnitt 2 sich mit dem theoretischen Hintergrund, konkret dem BIRD-Ansatz und ähnlichen Arbeiten zum Thema beschäftigt, beschreibt Abschnitt 3 die Verwendung der erstellten BIRD-Implementierung aus Benutzersicht. In Abschnitt 4 findet sich eine detaillierte Beschreibung der Implementierung an sich, bevor die Arbeit mit einem kurzen Fazit in Abschnitt 5 schließt.

2 Hintergrund

Bevor auf die erstellte Implementierung des BIRD-Ansatzes von Schütz u. a. (2016) eingegangen wird, wird in diesem Abschnitt der Ansatz selbst erläutert. Außerdem werden ähnliche Arbeiten, die sich mit Referenzmodellierung für Data Warehouses und verwandten Themen beschäftigen, beleuchtet.

2.1 BIRD

Basierend auf einem bereits zuvor entstandenen Konzept zur Anpassung von Referenzmodellen für Data Warehouses von Schütz und Schrefl (2014) entwickelten Schütz u. a. (2016) mit BIRD (Business Intelligence Reference Modeling for Data Warehouses) einen Ansatz zur Referenzmodellierung für relationales OLAP (Online Analytical Processing). Dieser umfasst neben multidimensionalen Referenzmodellen auch Referenzmodelle für Analysesituationen und auf diesen basierenden Analysegraphen, und sieht die fach- und bedarfsspezifische Anpassung dieser drei Konzepte vor.

Um BIRD gut verständlich zu gestalten und darzulegen, greifen Schütz u. a. (2016) dabei das von Golfarelli u. a. (1998) vorgestellte dimensionale Faktmodell (DFM – Dimensional Fact Model) auf und verwenden SQL für die Definition von Leistungskennzahlen. Zur grafischen Darstellung des BIRD-Metamodells werden UML-Klassendiagramme verwendet. Auf den nächsten Seiten werden die Grundkonzepte von BIRD erläutert, wobei an dieser Stelle darauf hingewiesen wird, dass alle in Abschnitt 2.1.1 bis Abschnitt 2.1.6 dargelegten Inhalte auf der Arbeit von Schütz u. a. (2016) basieren.

2.1.1 Metamodell für multidimensionale Referenzmodelle

Abbildung 1 zeigt das UML-Klassenmodell des Metamodells für multidimensionale Referenzmodelle. Die Basis des Modells bildet eine Faktklasse, die eine oder mehrere Kennzahlen (Measures) enthält. Jede Faktklasse und jede Kennzahl verfügen dabei über einen eindeutigen Namen. Für jede Kennzahl können mehrere Aggregationsoperationen angegeben werden. Außerdem kann eine Kennzahl mehreren Faktklassen zugeordnet werden, wobei die Zuordnung einer Kennzahl zu einer Faktklasse als verpflichtend gekennzeichnet werden kann. Das heißt, es kann festgelegt werden, dass die Kennzahl bei einer eventuellen Anpassung des multidimensionalen Referenzmodells nicht abgewählt werden darf. Zu den verschiedenen Typen von Kennzahlen findet sich Näheres im weiteren Verlauf dieses Abschnitts, zur Anpassung multidimensionaler Referenzmodelle in Abschnitt 2.1.2.

Neben den Kennzahlen verweist jede Faktklasse auf eine oder mehrere Dimensionen. Jede Dimension umfasst wiederum ein oder mehrere Levels (Ebenen), die in Hierarchien angeordnet werden. Je höher ein Level in einer Hierarchie platziert ist, desto gröber ist seine Granularität. Mehrere Hierarchien innerhalb einer Dimension beschreiben dabei alternative Aggregationspfade. Die Reihenfolge der Levels wird durch hierarchiebezogene Superlevels angegeben, wobei der Superlevel eines Levels den nächsthöheren Level innerhalb der Hierarchie darstellt. Ein Level ohne Superlevel entspricht dem obersten Level der Hierarchie. Die Beziehungen zwischen Levels innerhalb einer Hierarchie dürfen nicht zyklisch sein.

Jede Dimension enthält eine oder mehrere Hierarchien, wobei jede Hierarchie und auch jeder Level mit einer bestimmten Dimension verknüpft sind. Der Level mit der feinsten Granularität innerhalb einer Dimension wird als Basislevel bezeichnet. Alle Hierarchien müssen auf diesem Level aufbauen.

Die Zuordnung einer Dimension oder einer Hierarchie zu einer Faktklasse kann ebenfalls als verpflichtend angegeben werden. Wie bei der Zuordnung der Kennzahlen zu den Faktklassen bedeutet dies, dass die Dimension respektive die Hierarchie bei einer eventuellen Anpassung des

multidimensionalen Referenzmodells nicht abgewählt werden darf. Zusätzlich kann außerdem die Zuordnung einer Hierarchie zu einer Dimension als verpflichtend gekennzeichnet werden. Sowohl Dimensionen als auch Hierarchien können jeweils mehreren Faktklassen zugeordnet werden.

Einem Level können ein oder mehrere Attribute zugeordnet werden, wobei nicht jeder Level über ein Attribut verfügen muss. Auch hier kann die Zuordnung als verpflichtend angegeben werden, was bedeutet, dass bei einer eventuellen Anpassung des multidimensionalen Referenzmodells das Attribut nicht abgewählt werden darf, sofern nicht auch der entsprechende Level abgewählt wird.

Jede Dimension kann durch einen eindeutigen Namen identifiziert werden. Innerhalb einer Dimension sind auch die Namen der Hierarchien, Levels und Attribute eindeutig. Aus der Kombination aus Dimensionsname und Hierarchie-, Level- beziehungsweise Attributname ergibt sich eine eindeutige ID. Der Name der Dimension und der Name der Hierarchie, des Levels beziehungsweise des Attributs werden dabei durch einen Punkt getrennt (`Dimensionsname.Hierarchiename`, `Dimensionsname.Levelname`, `Dimensionsname.Attributname`), wobei nach Konvention Dimensions- und Hierarchienamen mit einem Großbuchstaben, Level- und Attributnamen mit einem Kleinbuchstaben beginnen.

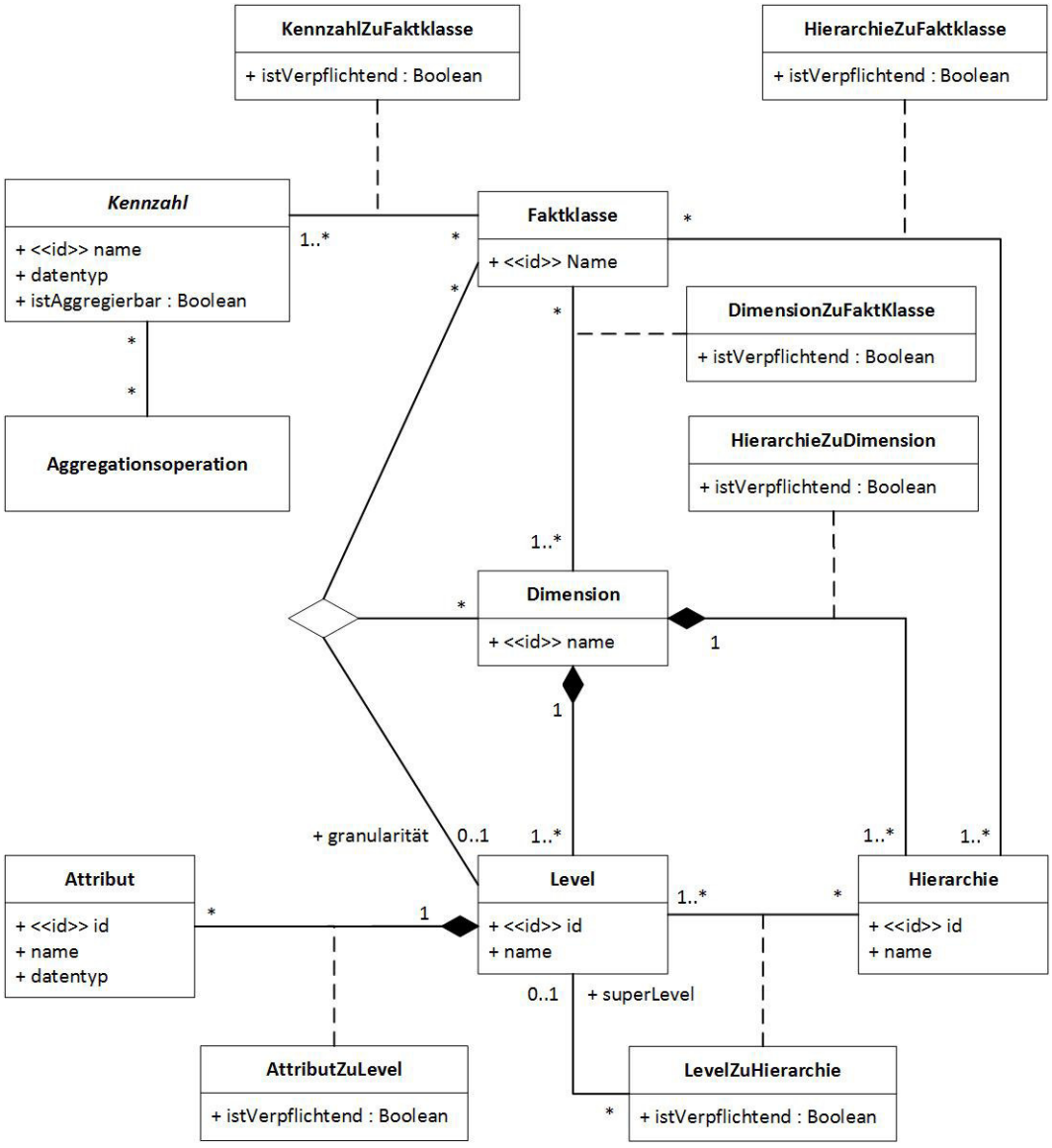

Abbildung 1: Metamodell für multidimensionale Referenzmodelle (Schütz u. a. 2016)

Die verschiedenen Typen von Kennzahlen im Metamodell des multidimensionalen Referenzmodells und deren Beziehungen zueinander sind in Abbildung 2 als UML-Klassendiagramm

dargestellt. Es gibt Basiskennzahlen und berechnete Kennzahlen, wobei letztere sich wiederum in arithmetische Kennzahlen, kumulative Kennzahlen und Vergleichskennzahlen unterscheiden. Die Bezeichnung der Kennzahltypen gibt dabei die Art der Berechnung der Kennzahl an, wobei den Basiskennzahlen keine Berechnung zugrunde liegt.

Die Berechnungsregel einer berechneten Kennzahl kann auf andere Kennzahlen oder Attribute der Faktklasse verweisen und ist in SQL verfasst. Sie entspricht einem Fragment des `SELECT`-Teils einer SQL-Abfrage. (`SELECT Berechnungsregel AS Kennzahlname`). Innerhalb der Berechnungsregel wird den Kennzahlnamen der Bezeichner `Fact`, getrennt durch einen Punkt vorangesetzt. Dieser Bezeichner ersetzt den Namen der Faktklasse und wird verwendet, um die Zuordnung von Kennzahlen zu verschiedenen Faktklassen zu erleichtern. Attribute werden mit ihrer eindeutigen ID wie oben beschrieben bestehend aus Dimensionsname und Attributname angegeben.

Jede Kennzahl beziehungsweise jedes Attribut kann in mehreren Berechnungsregeln, das heißt für mehrere berechnete Kennzahlen, verwendet werden. Wieder kann bei der Zuordnung festgelegt werden, ob der Verweis auf eine Kennzahl oder ein Attribut verpflichtend ist. Ist dies der Fall, muss die Kennzahl respektive das Attribut bei einer eventuellen Neudefinition der Berechnungsregel im Zuge einer Anpassung des multidimensionalen Referenzmodells auch in der neuen Definition der Berechnungsregel verwendet werden.

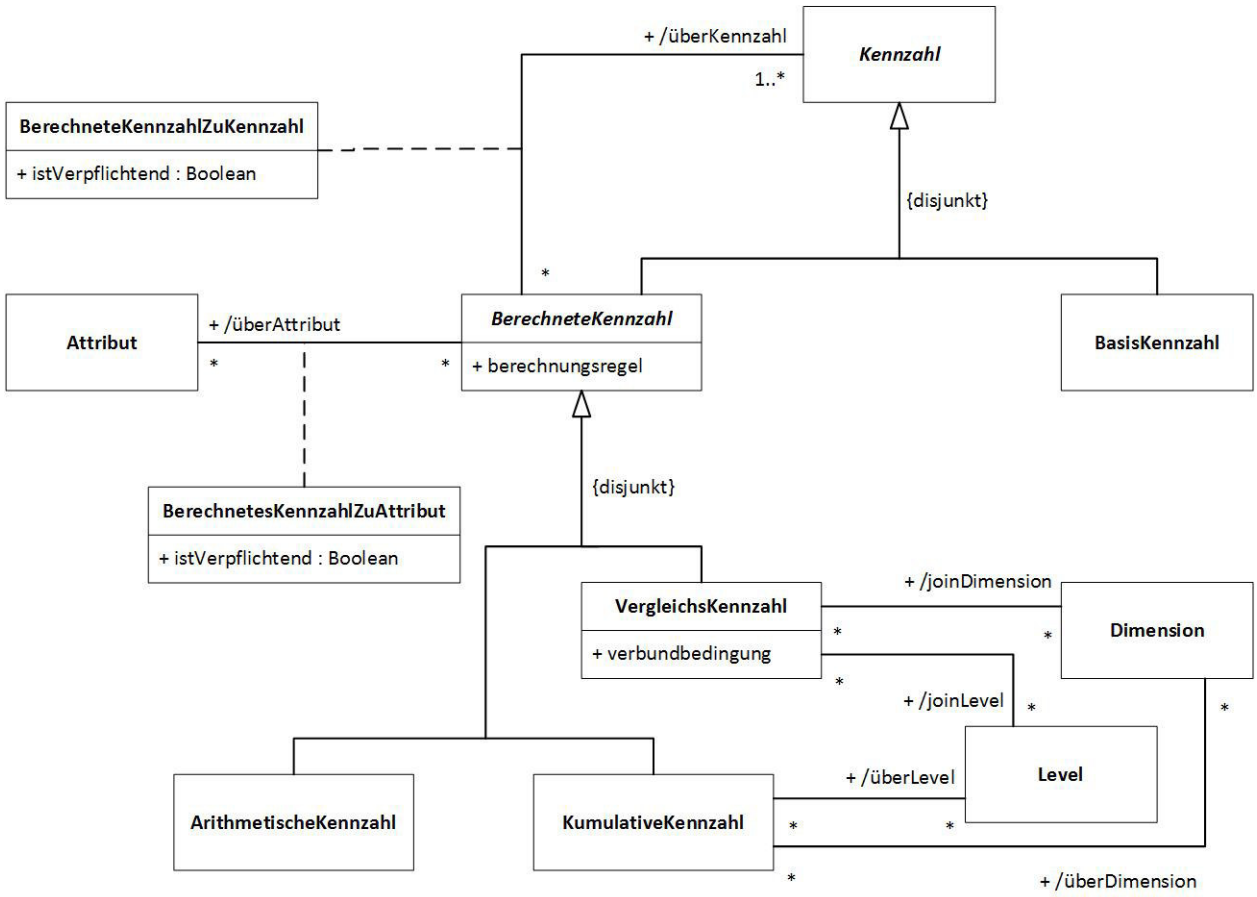

Abbildung 2: Kennzahltypen im Metamodell für Referenzmodelle (Schütz u. a. 2016)

Kumulative Kennzahlen teilen Fakten in Gruppen und können dafür neben Kennzahlen und Attributen auch auf Levels und Dimensionen verweisen. Auch hier kann jede Dimension und jedes Level von mehreren kumulativen Kennzahlen referenziert werden. Ähnliches gilt für Vergleichskennzahlen: Jede Vergleichskennzahl verfügt über eine Verbundbedingung, die auf Dimensionen und Levels der Faktklasse verweisen kann. Wie die Berechnungsregel ist die Verbundbedingung in SQL verfasst. Sie entspricht einem Fragment des Verbunds im `FROM`-Teil einer SQL-Abfrage (`SELECT Berechnungsregel AS Kennzahlname FROM Faktklassenname Fact ... JOIN`

`Faktklassenname ComparisonFact ... ON Verbundbedingung`). Für den Vergleich zweier Faktgruppen wird neben dem Bezeichner **`Fact`** analog der Bezeichner **`ComparisonFact`** verwendet.

Zum Filtern der Fakten in einem multidimensionalen Modell können jeder Faktklasse Prädikate zugeordnet werden. Diese Prädikate können in Dimensionsprädikate, multidimensionale Prädikate und Kennzahlprädikate unterschieden werden. Jedes Prädikat verfügt über einen Ausdruck, der einer Bedingung im **`WHERE`**-Teil einer SQL-Abfrage entspricht (**`SELECT ... FROM ... WHERE Ausdruck`**).

Abbildung 3 zeigt das Konzept der Dimensionsprädikate im Metamodell für multidimensionale Referenzmodelle als UML-Klassendiagramm. Der Ausdruck jedes Dimensionsprädikats verweist auf eine konkrete Dimension und auf ein oder mehrere Attribute. Bei der Zuordnung eines Attributs zu einem Dimensionsprädikat kann festgelegt werden, ob der Verweis verpflichtend ist. Dies bewirkt, dass das Attribut bei einer eventuellen Neudefinition des Ausdrucks im Zuge einer Anpassung des multidimensionalen Referenzmodells in der neuen Ausdrucksdefinition verwendet werden muss.

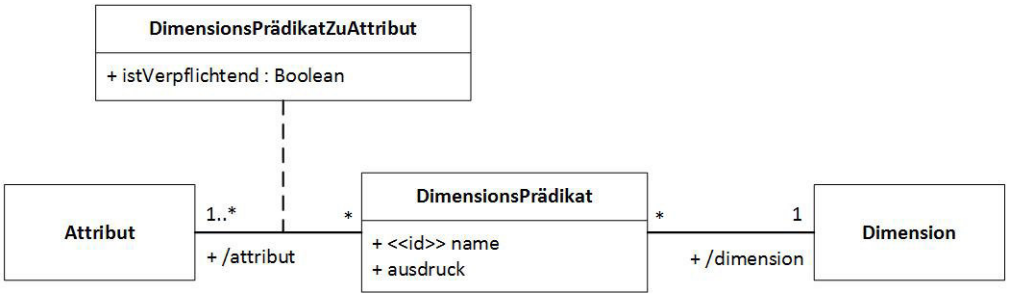

Abbildung 3: Dimensionsprädikate im Metamodell für Referenzmodelle (Schütz u. a. 2016)

Abbildung 4 zeigt das Konzept der multidimensionalen Prädikate im Metamodell für multidimensionale Referenzmodelle als UML-Klassendiagramm. Multidimensionale Prädikate unterscheiden sich von Dimensionsprädikaten insofern, als der Ausdruck eines multidimensionalen Prädikats auf mehrere Dimensionen verweisen kann.

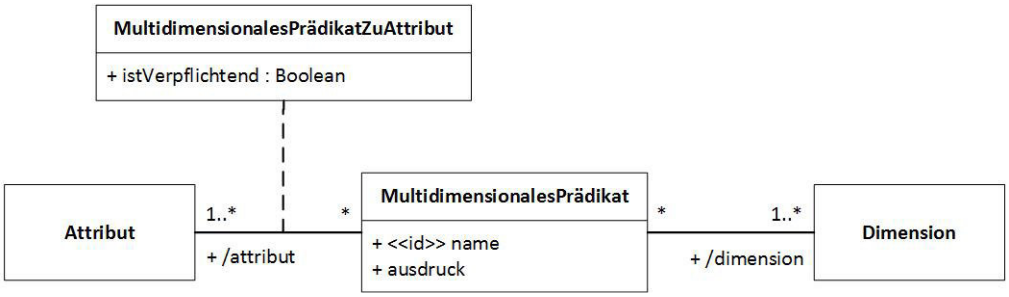

Abbildung 4: Multidimensionale Prädikate im Metamodell für Referenzmodelle (Schütz u. a. 2016)

Abbildung 5 zeigt das Konzept der Kennzahlprädikate im Metamodell für multidimensionale Referenzmodelle als UML-Klassendiagramm. Kennzahlprädikate erweitern das Konzept der multidimensionalen Prädikate: Nicht jedes Kennzahlprädikat muss im Prädikatausdruck auf eine Dimension oder ein Attribut verweisen, die Möglichkeit dazu besteht aber. Zusätzlich beziehungsweise anstelle von Attributen und Dimensionen verweist der Ausdruck jedes Kennzahlprädikats auf eine oder mehrere Kennzahlen. Wie bei der Zuordnung von Attributen kann auch bei der Zuordnung einer Kennzahl zu einem Kennzahlprädikat festgelegt werden, ob diese verpflichtend ist. Die Konsequenz einer solchen Kennzahl-Verpflichtung ist analog zu der oben erläuterten Attribut-Verpflichtung. Für alle Prädikate gilt, dass jede Dimension, jedes Attribut und jede Kennzahl im Ausdruck mehrerer Prädikate verwendet werden können.

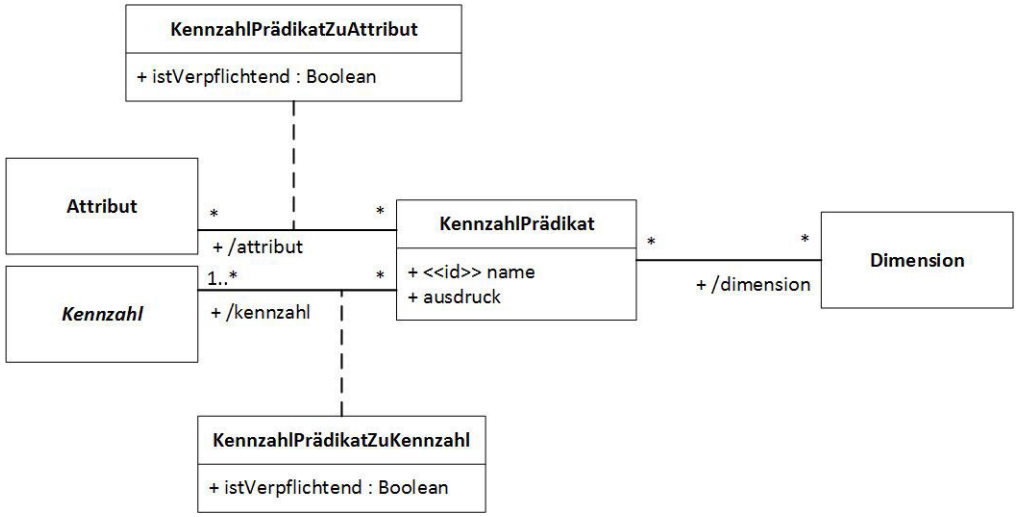

Abbildung 5: Kennzahlprädikate im Metamodell für Referenzmodelle (Schütz u. a. 2016)

Indem Elemente des multidimensionalen Referenzmodells als verpflichtend gekennzeichnet werden können, erlaubt BIRD, die Anpassungsmöglichkeiten des Referenzmodells zu beschränken. So kann beispielsweise die Befolgung abteilungs- oder unternehmensübergreifender Richtlinien für die Definition von Leistungskennzahlen oder Vorgehensweisen gewährleistet werden.

Mit Werkzeugen wie dem in der vorliegenden Arbeit verwendeten Indyco Builder der Firma Iconsulting kann aus dem multidimensionalen Referenzmodell ein relationales Datenmodell in Form von Star-Schema-Tabellen generiert werden. Dabei wird sowohl für die Faktklasse als auch für die damit verbundenen Dimensionen jeweils eine Tabelle generiert. Jeder Level und jedes Attribut der Dimension stellen dabei eine Spalte in der entsprechenden Dimensionstabelle dar. Die Fakttabelle enthält eine Spalte für jede Basiskennzahl und referenziert die Dimensionstabellen über Fremdschlüssel. Berechnete Kennzahlen können bei SQL-Abfragen aus den Basiskennzahlen abgeleitet oder in funktionsbasierten virtuellen Spalten in der Fakttabelle dargestellt werden. Wahlweise kann vor der Generierung der Tabellen eine Anpassung des multidimensionalen Referenzmodells erfolgen, wie im folgenden Abschnitt erläutert wird.

2.1.2 Anpassung von multidimensionalen Referenzmodellen

Die Anpassungsmöglichkeiten im Metamodell für multidimensionale Referenzmodelle sind in Abbildung 6 als UML-Klassendiagramm dargestellt. Für jede Faktklasse können mehrere Anpassungen (Customizations) erstellt werden, wobei nicht für jede Faktklasse eine Anpassung existieren muss. Über die Anpassung können Kennzahlen, Dimensionen und Hierarchien der Faktklasse abgewählt oder neue hinzugefügt werden. Zusätzlich können Attribute und Levels der Faktklasse abgewählt werden. Das Hinzufügen von Attributen und Levels ist nicht vorgesehen. Werden neue Attribute oder Levels gewünscht, sind neue Dimensionen respektive Hierarchien hinzuzufügen, die diese enthalten. Diese Vorgehensweise erleichtert die Prüfung einer Anpassung auf ihre Korrektheit und Konsistenz gegenüber der Faktklasse.

Wie in Abschnitt 2.1.1 bereits erläutert, können Elemente des multidimensionalen Referenzmodells als verpflichtend gekennzeichnet werden, was die Freiheiten bei der Abwahl von Kennzahlen, Dimensionen, Hierarchien, Attributen und Levels einschränkt. Weiters dürfen keine Elemente abgewählt werden, die für die Berechnung von nicht abgewählten oder neu hinzugefügten berechneten Kennzahlen benötigt werden.

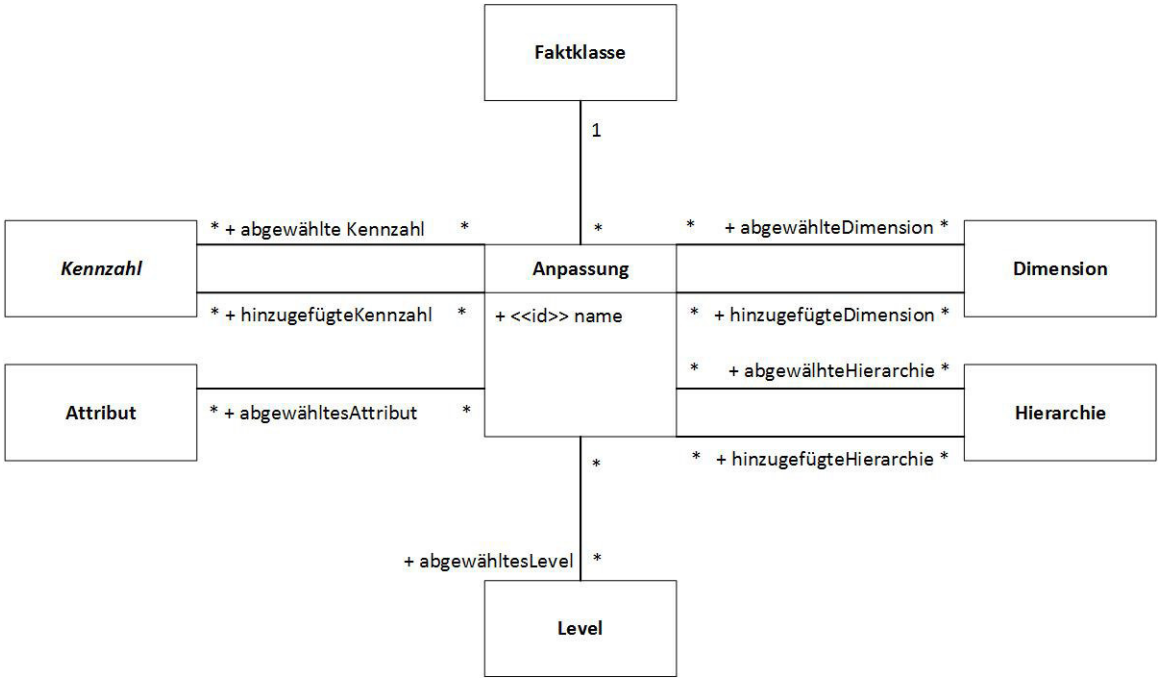

Abbildung 6: Anpassungen im Metamodell für Referenzmodelle (Schütz u. a. 2016)

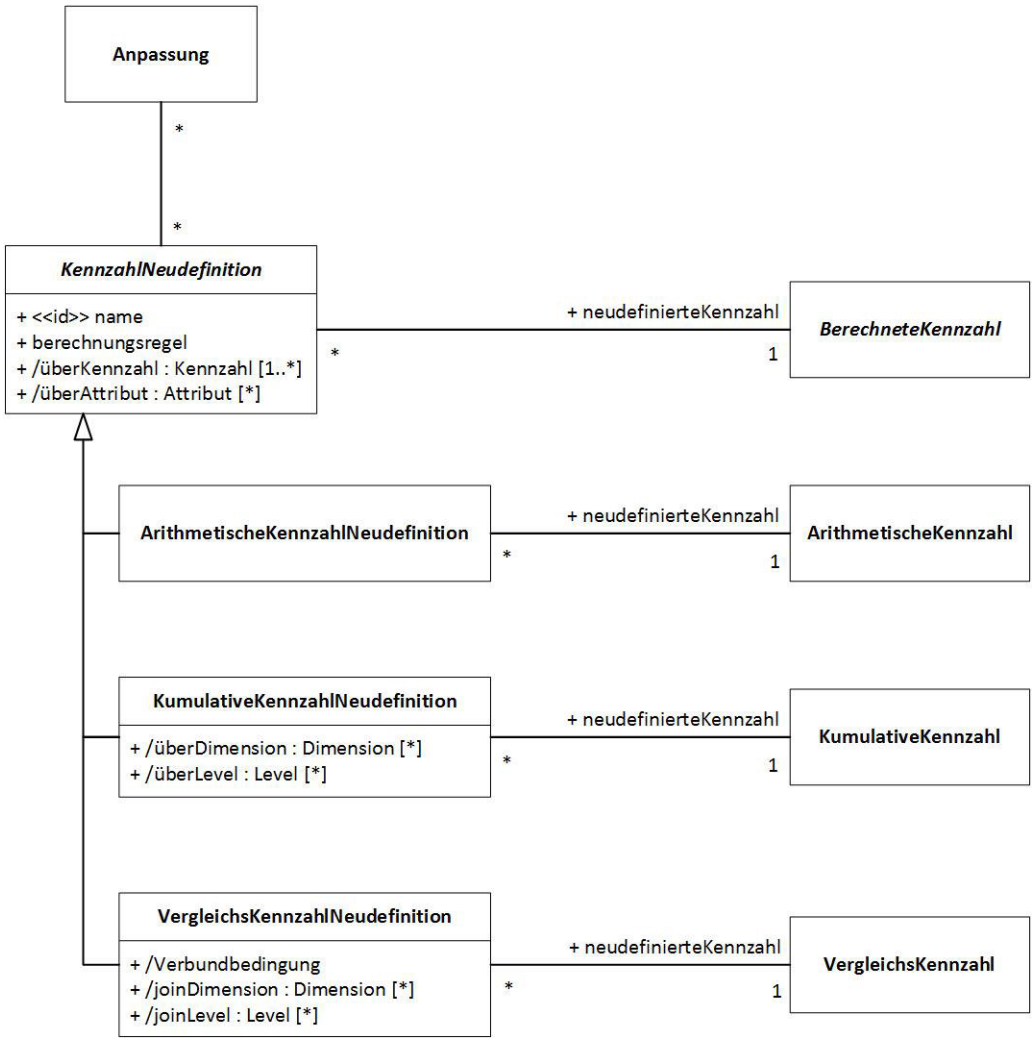

Abbildung 7: Kennzahl-Neudefinitionen im Metamodell für Referenzmodelle (Schütz u. a. 2016)

Zusätzlich zu abgewählten und hinzugefügten Elementen kann eine Anpassung Neudefinitionen der Berechnungsregeln bestehender berechneter Kennzahlen enthalten. Dieses optionale Vorgehen

wird in Abbildung 7 als UML-Klassendiagramm dargestellt. Jede Kennzahl-Neudefinition ist durch einen eindeutigen Namen gekennzeichnet und bezieht sich auf eine konkrete Kennzahl, dessen Berechnungsregel neu definiert wird. Daraus ergibt sich, dass sich für alle Arten berechneter Kennzahlen auch die Kennzahlen und Attribute, auf die verwiesen wird, durch die Anpassung des multidimensionalen Referenzmodells ändern können.

Bei kumulativen Kennzahlen können sich aufgrund der Neudefinition der Berechnungsregel analog auch die Dimensionen und Levels, auf die die kumulative Kennzahl verweist, verändern. Bei Vergleichskennzahlen kann zusätzlich zur Berechnungsregel auch die Verbundbedingung über eine Kennzahl-Neudefinition verändert werden. Dadurch ändern sich auch hier möglicherweise die Dimensionen und Levels, auf die verwiesen wird.

Wie in Abschnitt 2.1.1 erläutert, muss bei allen Neudefinitionen darauf geachtet werden, dass alle als verpflichtend gekennzeichneten Elemente der Berechnungsregel auch in der neuen Definition vorhanden sind. Außerdem muss gewährleistet werden, dass alle Kennzahlen, Attribute, Levels und Dimensionen, auf die in den Berechnungsregeln verwiesen wird, auch nach der Anpassung des multidimensionalen Referenzmodells noch vorhanden sind. Dies gilt allerdings nicht nur für neu definierte, sondern auch für bestehende, nicht veränderte berechnete Kennzahlen der Faktklasse.

Ähnlich den Kennzahlen können auch Prädikate optional im Zuge einer Anpassung des multidimensionalen Referenzmodells neu definiert werden. Abbildung 8 zeigt dies anhand des Beispiels der Dimensionsprädikate. Die Neudefinition von multidimensionalen Prädikaten und Kennzahlprädikaten läuft analog. Jede Prädikat-Neudefinition verfügt über einen eindeutigen Namen und nimmt Bezug auf ein konkretes Prädikat, dessen Ausdruck neu definiert wird. Wie bei den Kennzahl-Neudefinitionen können sich dabei auch die Modellelemente ändern, auf die das Prädikat verweist.

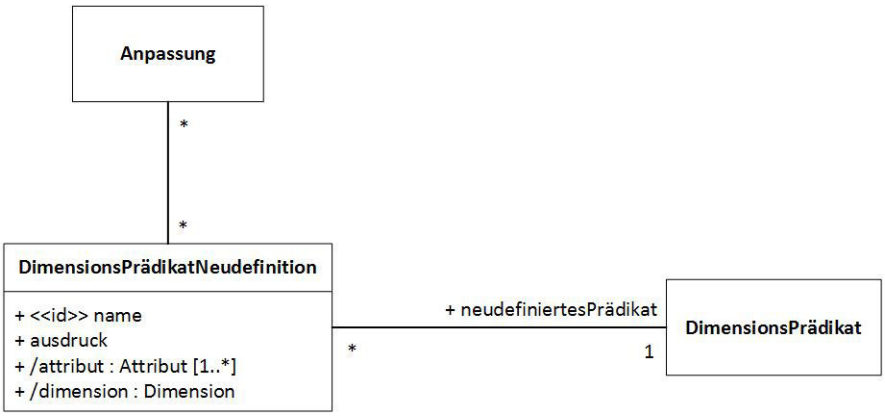

Abbildung 8: Prädikat-Neudefinitionen im Metamodell für Referenzmodelle (Schütz u. a. 2016)

Auch bei Prädikat-Neudefinitionen muss auf die als verpflichtend gekennzeichneten Elemente des Prädikatausdrucks Rücksicht genommen werden. Außerdem muss bei Prädikaten wie bei den Kennzahlen gewährleistet werden, dass alle Elemente des multidimensionalen Modells, auf die in bestehenden oder neu definierten Prädikatausdrücken verwiesen wird, auch nach der Anpassung des Referenzmodells noch vorhanden sind. Im Gegensatz zu den Kennzahlen werden Prädikate allerdings erst im Rahmen der Erstellung einer Analysesituation einem multidimensionalen Modell zugeordnet. Näheres dazu findet sich in den nächsten beiden Abschnitten.

2.1.3 Metamodell für Analysesituationen

Analysesituationen bezeichnen Sichten auf bestimmte Fakten eines multidimensionalen Modells. Da die Berechnungsregeln und Verbundbedingungen der berechneten Kennzahlen und die Ausdrücke der Prädikate im multidimensionalen Modell in BIRD als Fragmente von SQL-Abfragen formuliert

werden, können aus Analysesituationen direkt SQL-Abfragen generiert werden, die die gewünschte Sicht auf die Fakten liefern.

Abbildung 9 zeigt die Elemente einer Analysesituation als UML-Klassendiagramm. Jede Analysesituation nimmt Bezug auf eine konkrete Faktklasse und eine oder mehrere Kennzahlen dieser Faktklasse. Faktklassen und Kennzahlen können dabei in mehreren Analysesituationen verwendet werden. Zusätzlich kann für jede Kennzahl eine beliebige Anzahl an Aggregationsoperationen festgelegt werden.

Jede Analysesituation nimmt außerdem Bezug auf eine oder mehrere Dimensionen. Auch hier gilt, dass jede Dimension in mehreren Analysesituationen verwendet werden kann. Für jede Dimension wird die Feinheit der Granularität anhand des entsprechenden Levels angegeben.

Für jede Dimension der Analysesituation können Filterkriterien in Form von Dimensionsprädikaten, sogenannte Slice-Bedingungen, angegeben werden. Es muss aber nicht für jede Dimension eine Slice-Bedingung angegeben werden. Das Konzept der Dimensionsprädikate wurde bereits in Abschnitt 2.1.1 erläutert.

Weiters kann für jede Dimension ein Dice-Knoten angegeben werden, dessen Funktion derer einer Slice-Bedingung ähnelt: Der Dice-Knoten bestimmt den gewünschten Wert einer bestimmten Level-Spalte der entsprechenden Dimensionstabelle im Star-Schema. Wird ein Dice-Knoten angegeben, werden in der Analysesituation nur Fakten ausgewählt, die im angegebenen Dice-Level ebendiesen Wert enthalten. Daraus folgt, dass für jede Dimension, für die ein Dice-Knoten bestimmt wird, ein Dice-Level angegeben werden muss. Dieser bestimmt die Level-Spalte, die zum Filtern der Fakten anhand des Dice-Knotens herangezogen wird.

Neben der Filterung der Fakten nach Slice-Bedingungen und Dice-Knoten können die Fakten einer Analysesituation auch über multidimensionale Prädikate und Kennzahlprädikate gefiltert werden. Dazu können jeder Analysesituation ein oder mehrere solcher Prädikate zugeordnet werden. Auch hier gilt, dass nicht jede Analysesituation Prädikate verwenden muss und dass jedes Prädikat in mehreren Analysesituationen verwendet werden kann.

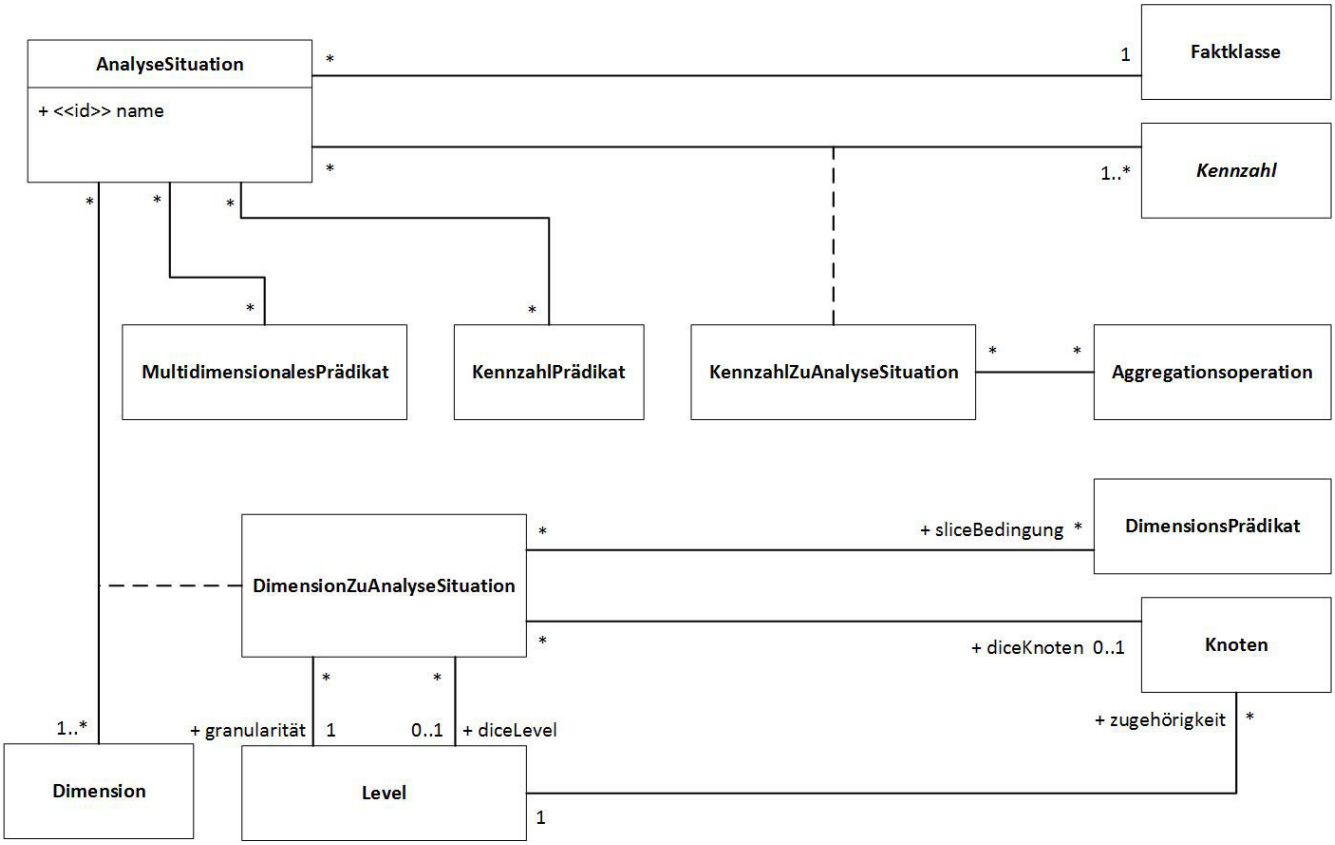

Abbildung 9: Metamodell für Analysesituationen (Schütz u. a. 2016)

Anstelle konkreter Kennzahlen, Aggregationsoperationen, Prädikate, Levels und Dice-Knoten können in einer Analysesituation auch Variablen gesetzt werden. Eine Analysesituation, die nicht nur konkrete Werte, sondern auch Variablen verwendet, wird in BIRD als Analysesituationsschema bezeichnet.

Ein Analysesituationsschema ermöglicht, dass die genaue Definition der Analysesituation durch Ersetzen der Variablen durch konkrete Werte wenn gewünscht auch erst bei der tatsächlichen Durchführung einer Analyse erfolgen kann. Außerdem können durch die Verwendung von Variablen und konkreten Werten die Möglichkeiten zur Anpassung einer Analysesituation erweitert beziehungsweise eingeschränkt werden. Details dazu finden sich im nächsten Abschnitt.

2.1.4 Anpassung von Analysesituationen

Wie bereits in den letzten Abschnitten erwähnt, können multidimensionale Referenzmodelle, die von Analysesituationen referenziert werden, an die konkreten Bedürfnisse angepasst werden. Zusätzlich den multidimensionalen Referenzmodellen können aber auch Analysesituationen selbst angepasst werden.

Die Anpassung einer Analysesituation ermöglicht das Hinzufügen und Entfernen von Kennzahlen und Prädikaten. Außerdem können die Granularitätslevel, Dice-Level und Dice-Knoten der Dimensionen der Analysesituation geändert werden. Dabei gilt aber, dass diese dimensionsspezifischen Änderungen nur insofern erfolgen dürfen, als Variablen durch konkrete Werte ersetzt werden. Die Änderung eines bereits in der ursprünglichen Analysesituation konkret angegebenen Werts eines Granularitätslevels, Dice-Levels oder Dice-Knotens ist nicht vorgesehen. So können die Möglichkeiten der Anpassung von Analysesituationen, wie in Abschnitt 2.1.3 bereits erwähnt, eingeschränkt werden. Ähnlich wie die Möglichkeit verpflichtender Elemente im multidimensionalen Referenzmodell kann diese Einschränkung dazu dienen, beispielsweise die Befolgung abteilungs- oder unternehmensübergreifender Richtlinien für die Definition von Vorgehensweisen zu gewährleisten.

2.1.5 Metamodell für Analysegraphen

Wie in Abbildung 10 als UML-Klassendiagramm dargestellt, beschreiben Analysegraphen einen möglichen Analyseprozess bestehend aus einer oder mehrerer Analysesituationen, die durch Navigationsschritte miteinander verbunden sind. Eine Analysesituation kann dabei in mehreren Analysegraphen verwendet werden.

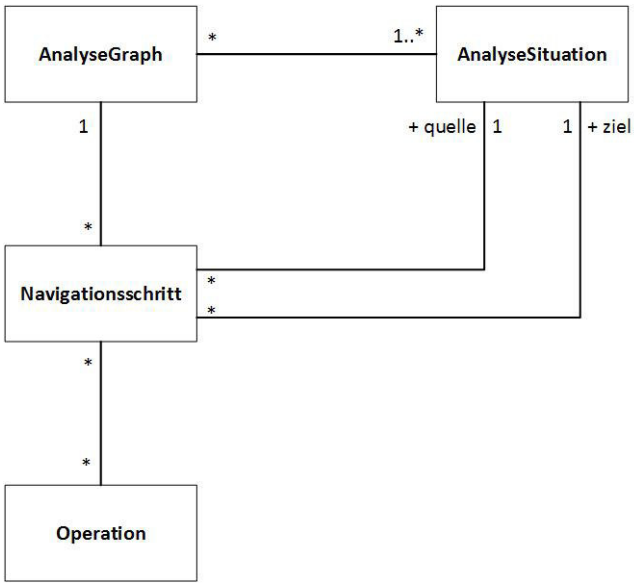

Abbildung 10: Metamodell für Analysegraphen (Schütz u. a. 2016)

Jeder Navigationsschritt eines Analysegraphen kann eine oder mehrere OLAP-Operationen enthalten und führt so von einer bestimmten Ausgangsanalysesituation zu einer bestimmten Zielanalysesituation. Dabei kann jede Analysesituation als Quelle oder Ziel mehrerer Navigationsschritte dienen. Die Analysesituationen verkörpern somit die Knoten, die Navigationsschritte die Kanten eines Analysegraphen.

Um eine Sicht auf die Fakten im multidimensionalen Modell in eine andere Sicht zu verändern, das heißt, um von einer Analysesituation über einen Navigationsschritt zu einer anderen zu gelangen, müssen bestimmte OLAP-Operationen durchgeführt werden. Im Konzept der BIRD-Analysegraphen kann jeder Navigationsschritt eine oder mehrere dieser Operationen enthalten. Enthält ein Navigationsschritt keine Operation, bedeutet dies, dass die Sicht auf die Fakten im multidimensionalen Modell sich nicht verändert. Das bedeutet, dass in diesem Fall die Ausgangsanalysesituation und die Zielanalysesituation die selbe sind.

BIRD definiert eine Liste von Navigationsoperationen, die in Tabelle 1 dargestellt ist. Diese Liste enthält häufig benötigte OLAP-Operationen, wobei Erweiterungen möglich sind. Die Operation `moveToNode` ersetzt den Dice-Knoten einer gegebenen Dimension, während die Operationen `addSliceCondition`, `removeSliceCondition` und `refocusSliceCondition` die Slice-Bedingungen dieser verändern. Die Operationen `addPredicate` und `removePredicate` fügen multidimensionale oder Kennzahlprädikate zur Analysesituation hinzu beziehungsweise entfernen diese. Die Operation `changeGranularity` ändert den Granularitätslevel einer gegebenen Dimension und die Operationen `addMeasure` und `removeMeasure` ergänzen beziehungsweise entfernen Kennzahlen der Analysesituation. Die Operation `drillAcrossToFactClass` ermöglicht es, die Faktklasse der Analysesituation durch eine andere zu ersetzen.

Dice	moveToNode(D,L,K)	Ersetzt den Dice-Knoten in Dimension D mit Knoten K in Level L.
Slice	addSliceCondition(D,P)	Ergänzt Dimensionsprädikat P in den Slice-Bedingungen von Dimension D.
	removeSliceCondition(D,P)	Entfernt Dimensionsprädikat P aus den Slice-Bedingungen von Dimension D.
	refocusSliceCondition(D,P_1,P_2)	Ersetzt Dimensionsprädikat P_1 in den Slice-Bedingungen von Dimension D durch Dimensionsprädikat P_2.
	addPredicate(P)	Ergänzt Prädikat P in der Analysesituation.
	removePredicate(P)	Entfernt Prädikat P aus der Analysesituation.
Granularität	changeGranularity(D,L)	Ersetzt den Granularitätslevel von Dimension D durch Level L.
Projektion	addMeasure(K,A)	Ergänzt Kennzahl K mit Aggregationsoperation A in der Analysesituation.
	removeMeasure(K)	Entfernt Kennzahl K aus der Analysesituation.
	drillAcrossToFactClass(F)	Ersetzt die Faktklasse der Analysesituation durch Faktklasse F.

Tabelle 1: Navigationsoperationen in Analysegraphen (Schütz u. a. 2016)

2.1.6 Anpassung von Analysegraphen

BIRD sieht nicht nur die Anpassung von multidimensionalen Referenzmodellen und Analysesituationen, sondern auch die von Analysegraphen vor. Die Möglichkeiten dieser Anpassung sind in Abbildung 11 als UML-Klassendiagramm dargestellt. Während sich jede Anpassung auf genau einen Analysegraphen bezieht, können für jeden Analysegraphen beliebig viele Anpassungen erstellt werden. Dabei muss nicht für jeden Analysegraphen eine Anpassung existieren.

Die Anpassung von Analysegraphen ermöglicht das Hinzufügen und Entfernen von Analysesituationen und Navigationsschritten. Dabei ist darauf zu achten, dass die Abfolge von

Analysesituationen und Navigationsschritten auch nach der Anpassung des Analysegraphen noch konsistent ist.

Neben dem Hinzufügen und Entfernen ermöglicht die Anpassung eines Analysegraphen außerdem die Neudefinition von Navigationsschritten, die es erlaubt, ein neues Set von OLAP-Operationen für den gewählten Navigationsschritt zu definieren. Auch hier muss die Konsistenz des angepassten Analysegraphen beachtet werden.

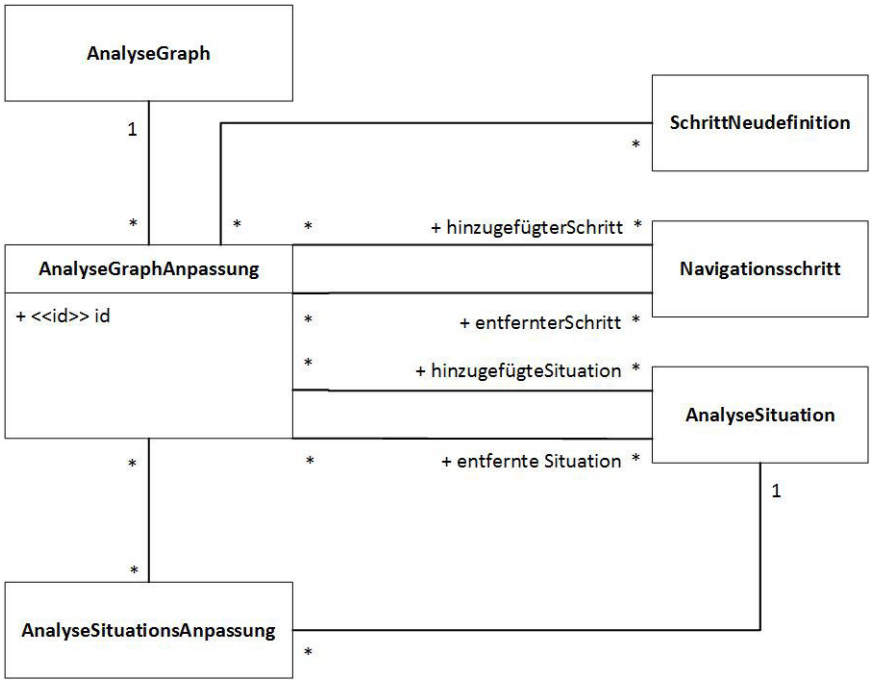

Abbildung 11: Anpassungen im Metamodell für Analysegraphen (Schütz u. a. 2016)

Durch die Verwendung von SQL-Fragmenten zur Definition von berechneten Kennzahlen und Prädikaten ermöglicht BIRD die Erstellung von ausführbaren Analysegraphen, beispielsweise mithilfe des von W3C (2015) vorgestellten Konzepts des State Chart XML (SCXML). Dabei stellt jede Analysesituation einen Zustand (State) dar, der über eine SQL-Abfrage für die Erstellung der gewünschten Sicht auf die Fakten im multidimensionalen Modell verfügt. Außerdem kann jeder Zustand Transitionen (Transitions) in Form von Navigationsschritten enthalten, die wiederum spezifische Aktionselemente (Custom Action Elements) in Form von OLAP-Operationen enthalten.

2.2 Ähnliche Arbeiten

Wie bereits in Abschnitt 1 erwähnt, beschäftigen sich Scholz u. a. (2010) und Olszak und Ziemba (2012) eingehend mit den Vorteilen und Herausforderungen der Einführung und Verwendung von Business Intelligence Werkzeugen in kleinen und mittleren Unternehmen. Scholz u. a. (2010) führten dazu eine Studie mit 214 KMUs in Sachsen in Deutschland mit dem Ziel durch, die bedeutendsten Vorteile und Herausforderungen und deren Auswirkungen zu identifizieren. Olszak und Ziemba (2012) führten Interviews in 20 KMUs in Oberschlesien in Polen durch, um kritische Erfolgsfaktoren in Zusammenhang mit der Einführung von Business Intelligence Systemen zu ermitteln.

Fettke und Vom Brocke (2013) definieren den Begriff Referenzmodell wie in Abschnitt 1 erläutert. Auch Thomas (2006) beschäftigt sich mit diesem Begriff und hält unter anderem wie Fettke und Vom Brocke (2013) fest, dass ein Referenzmodell ein Modell ist, dass die Konstruktion weiterer Modelle unterstützt.

Rosemann und Van Der Aalst (2007) stellen eine Modellierungssprache für die Referenzmodellierung vor, Thomas und Scheer (2006) ein Werkzeug für die kollaborative Verwaltung

dieser Modelle. Knackstedt u. a. (2006) entwickelten ein Werkzeug für die „konfigurative Referenzmodellierung“ namens H2-Toolset mit dem Ziel, explizites Expertenwissen in Referenzmodellen zu vereinen, zu bewahren und zur Verfügung zu stellen. Becker u. a. (2012) empfehlen schließlich, das H2-Toolset anzuwenden, um rechtlichen Vorschriften und Berichtspflichten gerecht zu werden.

Für Scheer und Nüttgens (2000) bildet die Referenzmodellierung einen wichtigen Bestandteil des Geschäftsprozessmanagements, wobei die Verwaltung von Prozessdaten in Data Warehouses durchgeführt werden kann. Recker u. a. (2006) sehen in der Referenzmodellierung großes Potenzial in Bezug auf die Anpassung von ERP-Software (Enterprise-Ressource-Planning-Software) an die Bedürfnisse konkreter Kunden. Vom Brocke (2009) präsentiert ein Konstruktionsprinzip für den Entwurf von Referenzmodellen, Becker u. a. (2007) einen allgemeinen Ansatz zur Konfigurierung dieser.

Becker und Knackstedt (2004) analysieren die Vor- und Nachteile sowie die Einsatzmöglichkeiten von Referenzmodellierung für Data Warehouses, während Knackstedt und Klose (2005) besonderen Wert auf die Konfigurierbarkeit von Referenzmodellen für Data Warehouses legen. Goeken und Knackstedt (2007) behandeln den Einsatz von Referenzmodellen bei der Planung und Konstruktion von Data Warehouses; Diamantini u. a. (2014) konzentrieren sich auf Referenzmodellierung von Leistungskennzahlen, wofür sie eine Ontologie erstellten. Auch diese Arbeiten werden bereits in Abschnitt 1 angesprochen.

Chan und Chan (2004) evaluierten Leistungskennzahlen anhand von Beispielprojekten aus der Bauindustrie und stellen einen Katalog geeigneter Leistungskennzahlen zur Verfügung. Auch Maté u. a. (2012) behandeln die Bedeutung von Leistungskennzahlen: Sie stellen einen Ansatz vor, der die aus Data-Warehouse-Abfragen gewonnen Leistungskennzahlen mit strategischen Geschäftsmodellen verknüpft, um die strategischen Zielsetzungen mit den Daten zu verbinden. Auch Strecker u. a. (2011) beschäftigen sich damit, wiederverwendbare und aussagekräftige Leistungskennzahlen zu ermitteln, ohne dabei die strategischen Hintergründe und Anforderungen außer Acht zu lassen.

Rumpe (2011) stellt mit UML/P eine Verknüpfung von UML-Modellen mit Java-Code vor. Ein ähnliches Konzept verfolgt BIRD mit der Verknüpfung von UML-Modellen mit SQL-Abfragen.

Golfarelli u. a. (1998) legten mit der Entwicklung des „Dimensional Fact Model“ (DFM – dimensionales Faktmodell) einen wichtigen Grundstein für die Modellierung von Data Warehouses und damit auch die Basis für die Entwicklung von Referenzmodellen in diesem Bereich. Dies wird durch die Arbeit von Malinowski und Zimányi (2006) ergänzt, die das Konzept von Hierarchien durch die Einführung verschiedener Kategorien erweitern.

Golfarelli und Rizzi (1998) erweiterten das DFM-Konzept von Golfarelli u. a. (1998) um ein Rahmenwerk zur Erstellung von logischem und physischem Entwurf, das die Entwicklung von Modellierungswerkzeugen und eine teilweise Automatisierung des Entwurfsprozesses vereinfacht beziehungsweise überhaupt erst ermöglicht. Beispiele für solche Werkzeuge sind der in der vorliegenden Arbeit verwendete Indyco Builder der Firma Iconsulting sowie das von Battaglia u. a. (2011) vorgestellte QBX. Indyco Builder automatisiert unter anderem die Generierung der von Pokorný und Sokolowsky (1999) beschriebenen Star- beziehungsweise Snowflake-Schema-Tabellen und übernimmt somit die logische Modellierung.

Heer und Shneiderman (2012) klassifizieren und bewerten interaktive Visualisierungen für Analysedaten. Indyco Builder unterstützt in seiner Funktionalität interaktive Visualisierung und auch Bearbeitung von multidimensionalen Datenmodellen gemäß dem DFM-Konzept von Golfarelli u. a. (1998).

Die bei BIRD verwendeten Analysesituationen und Analysegraphen wurden bereits von Neuböck u. a. (2012) vorgestellt. Nach Schütz u. a. (2016) dienten dabei WebML nach Ceri u. a. (2009) beziehungsweise IFML nach Acerbis u. a. (2015) als Basis für die Entwicklung und Verwendung von Analysegraphen. Die Anwendung dieser beiden Konzepte im industriellen Bereich und im Speziellen in Verbindung mit Industrie 4.0 behandeln Neuböck und Schrefl (2015). Trujillo u. a. (2000) verwenden

in ihrem Konzept zur objektorientierten Modellierung Zustandsdiagramme vergleichbar mit den in BIRD verwendeten Analysegraphen. Auch Trujillo u. a. (2001) verwenden in BIRD aufgegriffene Konzepte objektorientierter Modellierung für Data Warehouses. Die in BIRD verwendeten Prädikate ähneln den von Neumayr u. a. (2013) vorgestellten dimensionalen und multidimensionalen Konzepten.

Romero u. a. (2011) präsentieren einen Ansatz zur Strukturierung und Normalisierung von OLAP-Sitzungen, Aligon u. a. (2015) ein Konzept zur Wiederverwendung der Sitzungsdaten, um Empfehlungen für zukünftige Sitzungen erstellen zu können. Beide Entwürfe unterstützen die Wiederverwendbarkeit von Abfragen beziehungsweise OLAP-Sitzungen, ähnlich der Referenzmodellierung bei BIRD.

Der von Sapia (1999) vorgestellte Ansatz behandelt die Analyse von OLAP-Sitzungen, konkreter das Abfrageverhalten der Benutzer. Im Gegensatz zu Aligon u. a. (2015) ist dabei jedoch nicht die Erstellung von Empfehlungen für zukünftige Sitzungen die Zielsetzung, sondern die optimale Gestaltung des zugrundeliegenden Data Warehouse.

Thalhammer u. a. (2001) beschäftigen sich mit aktiven Data Warehouses („Active Data Warehouses“), die durch die Anwendung festgelegter Analyse-Regeln („Analysis Rules“), Entscheidungsprozesse unterstützen sollen. Auch Steiner u. a. (2015) verwenden Analyse-Regeln, allerdings mit dem Ziel, die Gestaltung und Bewertung vergleichender Datenabfragen zu erleichtern und zu verbessern.

3 Verwendung der BIRD-Implementierung

Nachdem in Abschnitt 2.1 der BIRD-Ansatz von Schütz u. a. (2016) diskutiert wurde, befasst sich dieser Abschnitt mit der Verwendung der erstellten BIRD-Implementierung aus Benutzersicht. Dazu wird in Abschnitt 3.1 die Erstellung multidimensionaler Modelle beziehungsweise multidimensionaler Referenzmodelle mit dem Indyco Builder der Firma Iconsulting erläutert. Abschnitt 3.2 befasst sich anschließend mit der Verwendung der erstellten Implementierung über BaseX-Datenbanken und XQuery-Funktionen. Details zur Implementierung selbst finden sich in Abschnitt 4.

Zur Erläuterung der Funktionalität des Indyco Builder und der erstellten BIRD-Implementierung wird in diesem Abschnitt mit einem durchgehenden Beispiel gearbeitet. Dieses entspricht einem vereinfachten Auszug aus einem beispielhaften Indyco-Builder-Projekt der Firma Iconsulting. Das vollständige Beispielprojekt ist jedem Indyco-Nutzer zugänglich.

3.1 Indyco Builder

Wie bereits erwähnt wird in der vorliegenden Arbeit für die Erstellung und Bearbeitung multidimensionaler Referenzmodelle die Anwendung Indyco Builder der Firma Iconsulting verwendet. Die verwendete Version ist 2.5.0.0. Auf den nächsten Seiten werden in Abschnitt 3.1.1 die Erstellung und Bearbeitung multidimensionaler Modelle mit Indyco Builder und in Abschnitt 3.1.2 die beim Speichern von Projekten erstellten Dateien behandelt.

3.1.1 Erstellung und Bearbeitung von multidimensionalen Modellen

Abbildung 12 zeigt die Projekt-Ansicht des oben erwähnten Beispielprojekts in Indyco Builder. Das Projekt umfasst lediglich ein multidimensionales Modell für Rechnungen namens `Invoice` (Rechnung). Dieses enthält eine Faktklasse, die ebenfalls den Namen `Invoice` trägt. Dieser Faktklasse sind fünf Hierarchien zugeordnet: `Agent`, `Currency` (Währung), `Customer` (Kunde), `Product` (Produkt) und `Time` (Zeit). Indyco Builder sieht keine Gruppierung von Hierarchien zu Dimensionen vor. Die in Abschnitt 2.1.1 erläuterten Konzepte Dimension und Hierarchie sind in Indyco demnach gleichbedeutend.

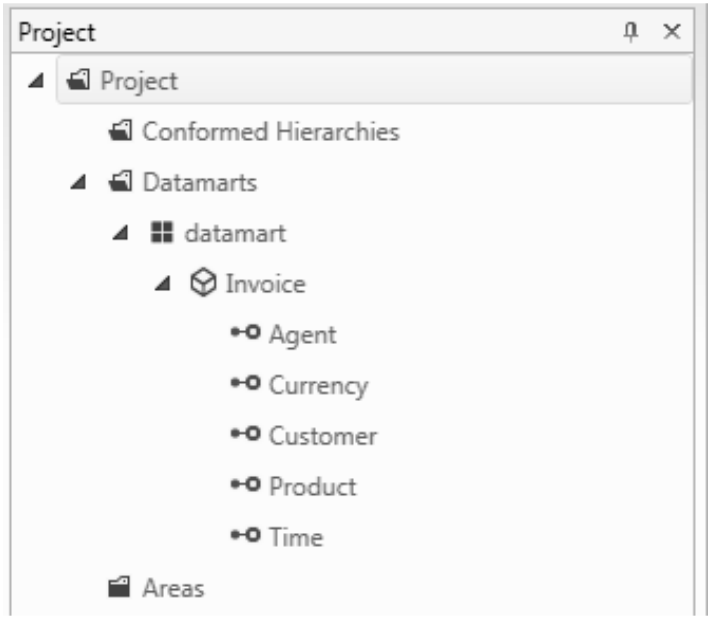

Abbildung 12: Indyco-Projekt-Ansicht für multidimensionale Referenzmodelle

Indyco Builder ermöglicht das Speichern mehrerer multidimensionaler Modelle und damit auch mehrerer Faktklassen in einem Projekt. Die multidimensionalen Modelle werden dabei in Datamarts

gruppiert. Ein Datamart repräsentiert somit einen Teilbestand der Daten in einem Data Warehouse. Da das verwendete Beispielprojekt nur ein multidimensionales Modell enthält, gibt es in diesem Fall auch nur einen Datamart. Indyco Builder ermöglicht es, sowohl multidimensionale Modelle als auch Datamarts individuell zu benennen.

Neben der Verwendung von Datamarts ermöglicht Indyco Builder auch die Speicherung von Referenzhierarchien (Conformed Hierarchies). Diese können auf einfache Weise verschiedenen Faktklassen zugeordnet werden. Damit ermöglicht Indyco Builder selbst bereits eine Art der Referenzmodellierung für Data Warehouses. Um die volle Funktionalität der erstellten Implementierung nutzen zu können, ist jede in Indyco Builder verwendete Hierarchie in Form einer solchen Referenzhierarchie anzulegen.

Für bessere Übersichtlichkeit bietet Indyco Builder auch die Möglichkeit, multidimensionale Modelle aus einem oder mehreren Datamarts in Bereiche (Areas) zu gruppieren. So ist eine individuelle, von Datamarts unabhängige Sicht auf Teilbereiche des Data Warehouse möglich. Da wie erwähnt das verwendete Beispielprojekt nur ein multidimensionales Modell umfasst, wird diese Funktionalität in der vorliegenden Arbeit nicht genutzt.

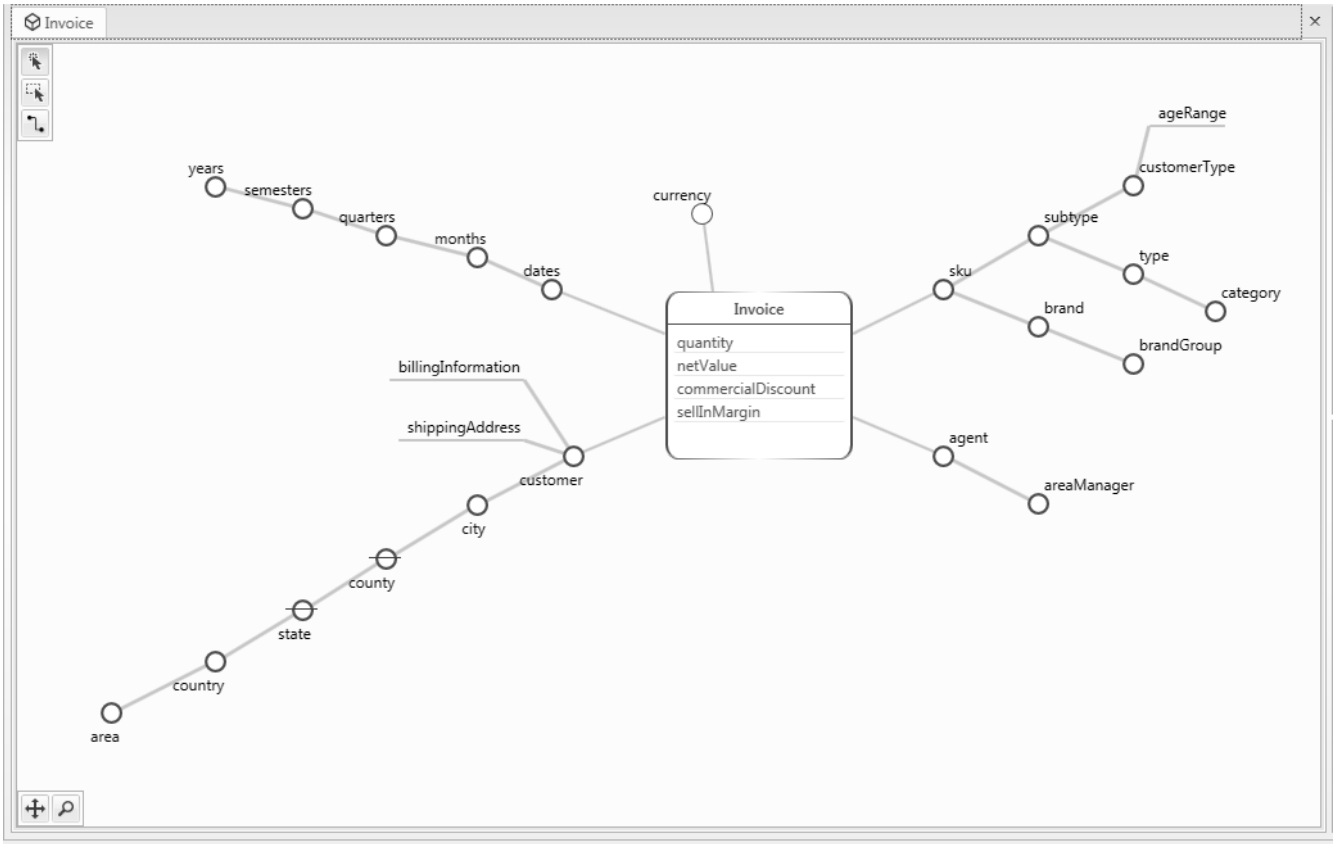

Abbildung 13: Indyco-Fakt-Ansicht für multidimensionale Referenzmodelle

Wird in der Projekt-Ansicht ein multidimensionales Modell ausgewählt, wird dieses in der Fakt-Ansicht von Indyco Builder dargestellt. Abbildung 13 zeigt die Fakt-Ansicht für das in der vorliegenden Arbeit als Beispiel verwendete multidimensionale Referenzmodell. Die Faktklasse des Modells trägt wie erwähnt den Namen **`Invoice`** und enthält vier Basiskennzahlen: **`quantity`** (Menge), **`netValue`** (Nettowert), **`commercialDiscount`** (Rabatt) und **`sellInMargin`** (Handelsspanne). Indyco Builder bietet die Möglichkeit, für Kennzahlen einfache Formeln zu hinterlegen. Da in der vorliegenden Arbeit jedoch komplexe SQL-Fragmente zur Berechnung der Kennzahlen notwendig sind, die in Indyco Builder nicht verarbeitet werden können, wird auf die Erfassung der berechneten Kennzahlen in Indyco Builder verzichtet. Daraus folgt, dass nur Basiskennzahlen Faktklassen verpflichtend zugeordnet werden können wie in Abschnitt 2.1.1 erläutert.

Die oben genannten fünf Hierarchien respektive Dimensionen sind mit ihren Levels und deren Attributen dargestellt. Die Hierarchie **`Time`** weist fünf Levels auf, die Hierarchie **`Agent`** zwei und die Hierarchie **`Currency`** nur einen. Die Hierarchie **`Customer`** umfasst sechs Levels, wobei dem Basislevel **`customer`** die beiden Attribute **`billingInformation`** (Rechnungsinformationen) und **`shippingAdress`** (Lieferadresse) zugeordnet sind. Zwei Levels der Hierarchie **`Customer`** sind als optional gekennzeichnet, nämlich die Levels **`county`** (Bezirk) und **`state`** (Bundesland). Die Hierarchie **`Product`** enthält ebenfalls einen Level, dem ein Attribut zugeordnet ist, nämlich den Level **`customerType`** (Kundentyp) mit dem Attribut **`ageRange`** (Altersgruppe). Außerdem handelt es sich hierbei um eine verzweigte Hierarchie. Dies entspricht in BIRD einer Dimension mit mehreren Hierarchien wie in Abschnitt 2.1.1 erläutert.

Wird in der Fakt- oder in der Projekt-Ansicht ein Modellelement ausgewählt, werden die Eigenschaften dieses Modellelements in der Eigenschaften-Ansicht von Indyco Builder angezeigt. Abbildung 14 zeigt einen Auszug dieser Ansicht am Beispiel eines Levels, konkret des Basislevels der Hierarchie **`Product`**, **`sku`** (Stock Keeping Unit – Artikelposition). Die verfügbaren Felder der Eigenschaften-Ansicht variieren je nach Typ des ausgewählten Modellelements.

Im Fall von Levels wie in Abbildung 14 können als allgemeine Eigenschaften Name, Beschreibung (Description), Beispielwerte (Sample Values) und Notizen (Notes) zum gewählten Level angegeben werden. Die Angaben zur logischen Modellierung enthalten den logischen Namen (Logical Name) und den SQL-Datentyp (SQL Type) sowie dessen Länge (Length). Bei Levels bestimmt Indyco Builder diese Eigenschaften, der Benutzer kann diese also nicht verändern. Wird beispielsweise eine Kennzahl ausgewählt, sind auch diese Eigenschaften veränderbar.

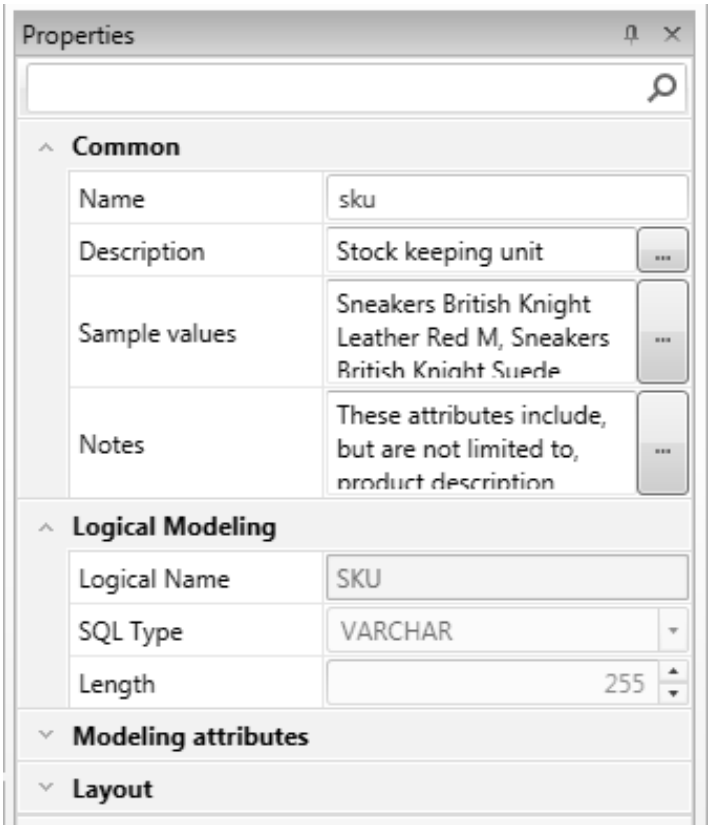

Abbildung 14: Indyco-Eigenschaften-Ansicht für Levels in multidimensionalen Referenzmodellen A

Die Möglichkeit, über die Eigenschaften-Ansicht von Indyco Builder Notizen zu hinterlegen, besteht sowohl für Kennzahlen und Hierarchien als auch für Levels und Attribute. Aus diesem Grund wird dieses Feld für die Kennzeichnung verpflichtender Modellelemente wie in Abschnitt 2.1.1 erläutert verwendet. Sobald einem Modellelement eine Notiz hinzugefügt wird, ist dieser als verpflichtend gekennzeichnet. Der Inhalt der Notiz ist dabei nicht von Bedeutung. Für zukünftige Implementierungen

besteht die Möglichkeit, die Notizen der Modellelemente über eine definierte Syntax nicht nur für die Kennzeichnung als verpflichtend, sondern auch für weitere Eigenschaften zu verwenden, die von Indyco Builder nicht unterstützt, jedoch für die Referenzmodellierung benötigt werden.

Da Indyco Builder wie bereits erwähnt nicht zwischen Hierarchien und Dimensionen differenziert, werden die in BIRD genannten Möglichkeiten zur Kennzeichnung verpflichtender Modellelemente dementsprechend eingeschränkt: Die verpflichtende Zuordnung von Dimensionen zu Faktklassen und von Levels zu Dimensionen entfällt. Sehr ähnliche Konzepte sind jedoch mit der Möglichkeit der verpflichtenden Zuordnung von Hierarchien zu Faktklassen und Levels zu Hierarchien, wie ebenfalls in BIRD definiert und in Abschnitt 2.1.1 erläutert, vorhanden.

Abbildung 15 zeigt einen weiteren Ausschnitt der Eigenschaften-Ansicht von Indyco Builder. Weiterhin wird als Beispiel der Level **`sku`** verwendet. Für diesen können diverse Modellierungsattribute angegeben werden, wobei an dieser Stelle nur auf die Attribute optional und beschreibend (descriptive) eingegangen wird, da diese in der vorliegenden Arbeit verwendet werden.

Wird ein Level als optional markiert, muss ihm nicht für jeden Fakt ein Wert zugewiesen werden. Dabei ist zu betonen, dass es sich bei optionalen und nicht als verpflichtend markierten Levels um verschiedene Konzepte handelt: Optionalen Levels muss nicht bei jedem Fakt ein Wert zugewiesen werden. Levels, die nicht als verpflichtend gekennzeichnet wurden, können über eine eventuelle Anpassung eines multidimensionalen Modells aus der Hierarchie entfernt werden. Dies bedeutet, dass ein Level möglicherweise gleichzeitig als optional und verpflichtend gekennzeichnet werden kann. Die Darstellung optionaler Levels in der Fakt-Ansicht zeigt Abbildung 13 am Beispiel der Levels **`county`** und **`state`** der Hierarchie **`Customer`**. Wird ein Level als beschreibend markiert, bedeutet dies gemäß der Definition in BIRD in Abschnitt 2.1.1, dass es sich nicht um einen Level, sondern um ein Attribut handelt. Auch hierfür gibt es in Abbildung 13 Beispiele für die Darstellung in der Fakt-Ansicht, nämlich beispielsweise das Attribut **`ageRange`** des Levels **`customerType`** in der Hierarchie **`Product`**.

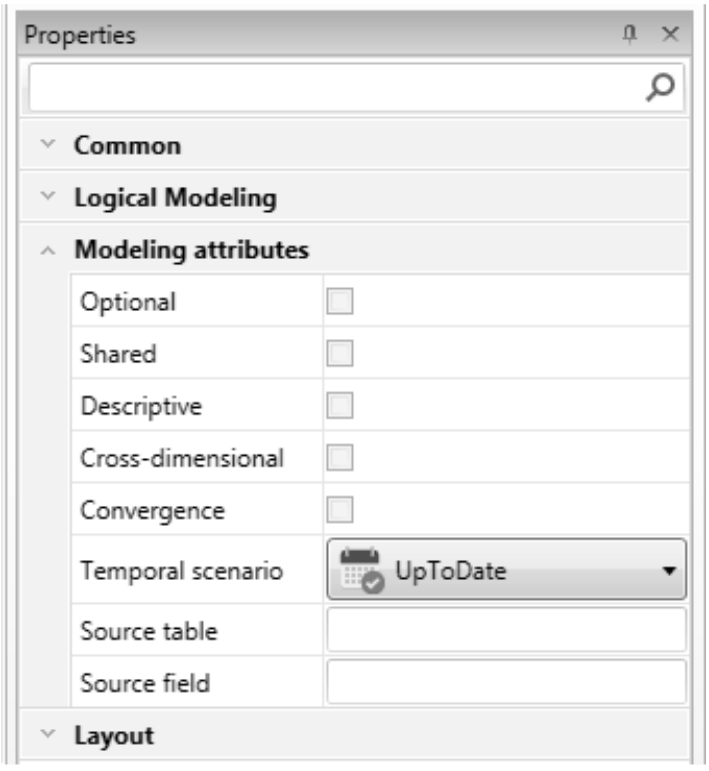

Abbildung 15: Indyco-Eigenschaften-Ansicht für Levels in multidimensionalen Referenzmodellen B

3.1.2 Dateistruktur von Indyco-Projekten

Indyco Builder speichert Projekte als Dateien vom Typ DFMPROJ (Dimensional Fact Model Project). Die Struktur dieser Dateien entspricht der von XML-Dateien. Die eindeutige Identifikation von Elementen erfolgt dabei über UUIDs (Universally Unique Identifiers). Liste 1 zeigt die Projektdatei des verwendeten Beispielprojektes. Das Wurzelelement **`project`** gibt den Namen des Projekts, seine UUID und die Indyco-Builder-Versionsnummer, mit der das Projekt erstellt wurde, an.

Jedes Projekt enthält ein Data Warehouse, in dem die Datamarts gespeichert werden. Für jeden Datamart werden wiederum Name und UUID gespeichert. In jedem Datamart gibt es eine Liste von Faktklassen namens **`factschemas`**, denen ebenfalls eine UUID zugeordnet wird. Über diese Faktklassen und ihre UUIDs werden Faktklassendateien referenziert, die im weiteren Verlauf dieses Abschnitts behandelt werden.

Neben dem Data Warehouse enthält jedes Projekt außerdem eine Liste der Sichten namens **`views`** und eine Liste der Bereiche des Projekts namens **`areas`**. Da diese Funktionen von Indyco Builder im vorliegenden Beispiel wie in Abschnitt 3.1.1 erwähnt nicht verwendet werden, sind diese Elemente in Liste 1 leer. Auch auf das Anlegen von spezifischen Eigenschaften wird in der vorliegenden Arbeit verzichtet. Daher ist die Liste der **`customProperties`** in Liste 1 ebenfalls leer.

Zuletzt enthält jedes Projekt noch die verwendeten Referenzdimensionen namens **`conformedHierarchies`** und projektspezifische Optionen, genannt **`projectOptions`**. Letztere enthalten Standardwerte für die logische Modellierung. Da diese Elemente in der vorliegenden Arbeit jedoch nicht verändert oder abgerufen werden und vergleichsweise umfangreich sind, werden sie aus Gründen der Übersichtlichkeit in Liste 1 nicht dargestellt.

```
01 <project name="Project" id="273adea5-da06-43fb-9a95-65596b23f036" toolversion="2.5.0.0">
02   <datawarehouse>
03     <datamarts>
04       <datamart name="datamart" id="a7347976-c3b4-4e13-a82d-4f950e0c75ba">
05         <factschemas>
06           <factschema id="7b7d4c05-d2fd-4957-8ad9-ed89bfa6a707"/>
07         </factschemas>
08       </datamart>
09     </datamarts>
10   </datawarehouse>
11   <conformedHierarchies>
       ...
17   </conformedHierarchies>
18   <views/>
19   <areas/>
20   <customProperties>
21     <attributes/>
22     <measures/>
23   </customProperties>
24   <projectOptions>
       ...
35   </projectOptions>
36 </project>
```

Liste 1: Indyco-Projektdatei für multidimensionale Referenzmodelle

Wie bereits erwähnt, wird beim Speichern eines Indyco-Projektes neben der Projektdatei für jede Faktklasse eine Faktklassendatei angelegt. Diese Dateien werden im Verzeichnis Datamarts des Projektverzeichnisses gespeichert und sind vom Typ DFM (Dimensional Fact Model). Wie bei den Projektdateien entspricht auch die Struktur der Faktklassendateien der von XML-Dateien. Eine weitere Parallele zu den Projektdateien ist die Verwendung von UUIDs für die eindeutige Identifikation von Elementen.

Das verwendete Beispielprojekt enthält lediglich ein multidimensionales Modell. Dessen Faktklassendatei ist in Liste 2 dargestellt und beginnt mit Angaben zur XML-Version und Kodierung. Diesen folgt das Wurzelelement **`graphml`**, welches die verwendeten Namensräume definiert. Dieses

Wurzelelement enthält `key`-Elemente (Schlüssel), die im folgenden Element, welches das multidimensionale Modell als Graphen beschreibt, zur Kategorisierung von Subelementen dienen.

Jeder Graph enthält Subelemente des Typs `node` (Knoten), `edge` (Kante) und `hierarchy` (Hierarchie), wobei die Knoten in `fact`-Knoten für Faktklassen und `attribute`-Knoten für Levels und Attribute der Hierarchien unterschieden werden. Jedem Graphen werden eine ID, die Version des Indyco Builder, mit der der Graph erstellt wurde, die UUID der Faktklasse als `factSchemaId` und ein Name zugeordnet. Der Name des Graphen entspricht dabei standardmäßig dem der Faktklasse. Weiters enthält jeder Graph eine Angabe zum standardmäßig verwendeten Kantentyp als `edgedefault`.

Jeder Knoten des Graphen ist durch eine UUID eindeutig identifizierbar. Durch den oben erwähnten Schlüssel wird der Knoten als Faktknoten oder Level- beziehungsweise Attributknoten identifiziert. Zwischen Level- und Attributknoten wird nicht über einen Schlüssel unterschieden, die Differenzierung erfolgt darüber, ob der Knoten als beschreibend gekennzeichnet ist, wie bereits im vorhergehenden Abschnitt erläutert wurde.

Der Faktknoten enthält eine UUID, die Eigenschaften der Faktklasse, wie zum Beispiel Name und Beschreibung, sowie verschiedene Angaben zur grafischen Darstellung in der Fakt-Ansicht von Indyco Builder. Innerhalb des Faktknotens ist eine Liste von Kennzahl-Elementen gespeichert, wobei auch für jede Kennzahl eine UUID, die in Indyco Builder festgelegten Eigenschaften der Kennzahl und Angaben für die grafische Darstellung gespeichert sind. Der Übersichtlichkeit wegen wird in Liste 2 nur die Kennzahl `quantity` als Beispiel dargestellt.

Wie bereits in Abschnitt 3.1.1 erwähnt wird in der vorliegenden Arbeit bei den in Indyco Builder erfassten Kennzahlen auf die Hinterlegung von Formeln, spezifischen Eigenschaften und spezifischen Einstellungen für die logische Modellierung verzichtet. Deshalb sind die entsprechenden Subelemente der in Liste 2 dargestellten Kennzahl leer.

Auch Level- respektive Attributknoten sind durch eine UUID eindeutig identifizierbar und durch einen Schlüssel als solche gekennzeichnet. Für jeden Knoten sind eine UUID, die Eigenschaften des Levels respektive Attributs und Angaben für die grafische Darstellung in der Fakt-Ansicht von Indyco Bilder gespeichert. Diese Eigenschaften enthalten dabei wie erwähnt auch die Angabe, ob es sich um einen beschreibenden Knoten, also ein Attribut oder um ein Level handelt. Außerdem enthält jeder Level- respektive Attributknoten einen Verweis auf die UUID der Hierarchie, der er angehört.

Wie bei den Kennzahlen sind in Liste 2 auch bei den Level- und Attributknoten die Subelemente für spezifische Eigenschaften und spezifische Einstellungen für die logische Modellierung aus den erwähnten Gründen leer. Es wird in Liste 2 nur der Level `subtype` der Hierarchie `Product` als Beispiel dargestellt.

Neben den Fakt-, Level- und Attributknoten enthält jeder Graph Elemente vom Typ `edge` (Kante). Neben den bereits bei den Knoten erwähnten Angaben für die grafische Darstellung in der Fakt-Ansicht des Indyco Builder werden für jede Kante eine UUID, der Ausgangsknoten als `source`, der Zielknoten als `target` und die UUID der Hierarchie, der die Kante angehört, gespeichert.

Der letzte Elementtyp innerhalb des Graphen ist der der `hierarchy`(Hierarchie). Für jede Hierarchie sind dabei eine UUID, die ID der Kante vom Fakt- zum Basisknoten der Hierarchie und die in Indyco Builder angegebenen Eigenschaften der Hierarchie gespeichert. In Liste 2 ist nur eine Hierarchie, nämlich die Hierarchie `Customer` als Beispiel dargestellt.

Neben dem Graphen enthält das Wurzelelement der Faktklassendatei noch ein Element vom Typ `additivityMatrix` (Additivitätsmatrix). In diesem ist für jede Kombination aus Kennzahl und Hierarchie ein Subelement `item` angelegt, dessen Operator die Aggregationsoperation angibt. Standardmäßig wird hier für jedes Element die Operation `SUM` (Summe) angegeben. Kennzahl und Hierarchie werden dabei über ihre UUIDs referenziert. Im Rahmen der vorliegenden Implementierung werden erst im späteren Verlauf Aggregationsoperationen definiert. Aus diesem Grund ist es nicht notwendig, die von Indyco Builder erstellten Standardwerte innerhalb der Additivitätsmatrix zu verändern.

```
001 <?xml version="1.0" encoding="UTF-8"?>
002 <graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi=
   "http://www.w3.org/2001/XMLSchema-instance" xmlns:ico="http://iconsulting.biz/dfm"
   xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
   http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
003   <key id="d0" for="node" attr.type="fact"/>
004   <key id="d1" for="node" attr.type="attribute"/>
005   <key id="d2" for="edge" attr.type="arc"/>
006   <graph id="G" edgedefault="directed" toolversion="2.5.0.0" factSchemaId=
   "7b7d4c05-d2fd-4957-8ad9-ed89bfa6a707" name="Invoice">
007     <node id="ba84fd4e-7199-4c7c-a18c-0e8f06acb5a2">
008       <data key="d0">
009         <ico:Fact Name="Invoice" Description="..." HistoricalDepth="0" Id=
   "ba84fd4e-7199-4c7c-a18c-0e8f06acb5a2" X="580" Y="340" BackgroundColor="#FFFFFF"
   BorderColor="#00539C" IsBoldText="false" IsItalicText="false" LoadingFrequency="">
010           <ico:Measures>
011             <ico:Measure Name="quantity" Description="..." Note="" IsBoldText="false"
   IsItalicText="false" SourceTable="" SourceField="" Id=
   "7ddc948b-498e-425a-9834-80a325b17fb5">
012               <ico:Formula/>
013               <ico:customProperties/>
014               <ico:logicalModeling/>
015             </ico:Measure>
                  ...
031           </ico:Measures>
032         </ico:Fact>
033       </data>
034     </node>
035     <node id="ca37dcd0-f98c-4d98-ad97-905175c45150">
036       <data key="d1">
037         <ico:Attribute Name="subtype" Description="..." Id=
   "ca37dcd0-f98c-4d98-ad97-905175c45150" X="862" Y="288" BackgroundColor="#FFFFFF"
   BorderColor="#00539C" IsBoldText="false" IsItalicText="false" IsDescriptive="false"
   IsOptional="false" IsShared="false" SampleValues="..." Note="" MasterId=
   "34237ec7-733d-4be0-ae0f-5bbaa649f31b" HierarchyId="87d656f1-1326-4b96-8bda-4f2c7bfb07a8"
   MasterHierarchyId="93c1ee5f-f0d2-4b14-b05a-805f69a22f01" IsCrossDimensional="false"
   TemporalScenario="UpToDate" SourceTable="" SourceField="">
038           <ico:customProperties/>
039           <ico:logicalModeling/>
040         </ico:Attribute>
041       </data>
042     </node>
        ...
227     <edge id="63b33cee-ee6e-4e0e-91b4-eba81fda10e5" source=
   "ba84fd4e-7199-4c7c-a18c-0e8f06acb5a2" target="f9da22b0-0606-47ef-9cba-20be27a03ea8"
   IsBoldText="false" IsItalicText="false" HierarchyId=
   "0ebae6cf-1afd-4e60-b03a-3d3372358edc" MasterHierarchyId="000-0000-0000-0000-00"
   MasterId="000-0000-0000-0000-00">
228       <data key="d2">
229         <ico:Arc Role="" IsConvergence="false" IsOptional="false" IsMultiple="false"
   IsRecursive="false" IndexOfRecursion="0" SourceConnectorPosition="Auto"
   IndexOfSameNodesLink="0"/>
230       </data>
231     </edge>
        ...
347     <hierarchy id="0ebae6cf-1afd-4e60-b03a-3d3372358edc" rootArcId=
   "63b33cee-ee6e-4e0e-91b4-eba81fda10e5" name="Customer" description="" note=""/>
        ...
352   </graph>
353   <additivityMatrix xmlns="">
354     <item measureId="7ddc948b-498e-425a-9834-80a325b17fb5" hierarchyId=
   "0ebae6cf-1afd-4e60-b03a-3d3372358edc">
355       <operator id="Sum"/>
356     </item>
        ...
414   </additivityMatrix>
415 </graphml>
```

Liste 2: Indyco-Faktklassendatei für multidimensionale Referenzmodelle

3.2 BaseX und XQuery

Wie bereits zu Beginn dieser Arbeit erwähnt, wurde für die vorliegende BIRD-Implementierung neben Indyco Builder von Iconsulting BaseX verwendet. Die genutzte Version ist 8.2.3. Für die Erstellung und Verwaltung von Referenzmodellen, Referenzkatalogen und Anpassungen sowie für die Generierung ausführbarer Analysegraphen wurden XQuery-Funktionen erstellt, die in diesem Abschnitt aufgelistet und erläutert werden. Für die Verwendung aller Funktionen ist anzumerken, dass XQuery im Gegensatz zu SQL zwischen Groß- und Kleinbuchstaben unterscheidet.

Tabelle 2 gibt eine Übersicht über die XQuery-Funktionen, die der Erstellung und Verwaltung von BaseX-Datenbanken dienen. Die Funktion `createDB` dient zum Erstellen, die Funktion `exportDB` zum Exportieren und die Funktion `dropDB` zum Löschen einer Datenbank, wobei die Datenbank bei allen dieser und den folgenden Funktionen über ihren Namen referenziert wird. Für den Export einer Datenbank mit `exportDB` muss zusätzlich zum Namen der Datenbank auch der Pfad des Verzeichnisses angegeben werden, in das die Datenbank exportiert werden soll. Generell gilt bei allen der in diesem und den folgenden Abschnitten angeführten XQuery-Funktionen, dass alle Elemente über ihren Namen referenziert werden, sofern nichts Gegenteiliges angegeben wird.

Neben dem Konzept der multidimensionalen Referenzmodelle setzt die vorliegende BIRD-Implementierung mit Referenzkatalogen für Hierarchien, Kennzahlen, Prädikate, Analysesituationen und Analysegraphen zusätzliche Möglichkeiten für die Referenzmodellierung für Data Warehouses um. Derartige Referenzkataloge können über die Funktion `createCatalogues` einer angegebenen BaseX-Datenbank hinzugefügt werden.

Funktion	Parameter	Beschreibung
createDB	$database xs:string	Erzeugt eine leere BaseX-Datenbank mit dem angegebenen Namen ($database).
exportDB	$database xs:string $path xs:string	Exportiert die angegebene BaseX-Datenbank ($database) in den angegebenen Pfad ($path).
dropDB	$database	Löscht die angegebene BaseX-Datenbank ($database).
createCatalogues	$database xs:string	Erzeugt in der angegebenen BaseX-Datenbank ($database) leere Referenzkataloge für Hierarchien, Kennzahlen, Prädikate, Analysesituation und Analysegraphen.

Tabelle 2: Funktionen zur Verwaltung von BaseX-Datenbanken

Bevor Anpassungen auf multidimensionale Referenzmodelle, Analysesituationen und Analysegraphen angewendet und darauf basierend ein ausführbares Ablaufmodell erstellt werden kann, müssen der BaseX-Datenbank neben den Referenzkatalogen auch noch andere Dateien vom Typ XML mit vorgegebenen Elementstrukturen hinzugefügt und diese befüllt werden. Die für die Bewältigung dieser Aufgaben verfügbaren XQuery-Funktionen werden auf den folgenden Seiten erläutert.

Zuerst werden in Abschnitt 3.2.1 die Funktionen zur Verwaltung multidimensionaler Referenzmodelle diskutiert. Nachdem Abschnitt 3.2.2 bis Abschnitt 3.2.6 die Funktionen zur Verwaltung der bereits erwähnten Referenzkataloge behandeln, widmen sich Abschnitt 3.2.7 bis Abschnitt 3.2.9 den Funktionen zur Verwaltung von Anpassungen für multidimensionale Referenzmodelle, Analysesituationen und Analysegraphen. Abschnitt 3.2.10 befasst sich mit den Funktionen für die Anwendung der Anpassungen und die Erstellung des ausführbaren Analysegraphen.

3.2.1 Multidimensionale Referenzmodelle

Die für die Verwaltung der mit Indyco Builder erstellten multidimensionalen Referenzmodelle verfügbaren XQuery-Funktionen sind in Tabelle 3 dargestellt. Die Funktion `insertModel` importiert das angegebene multidimensionale Referenzmodell in die angegebene BaseX-Datenbank. Das multidimensionale Referenzmodell entspricht dabei einer wie in Abschnitt 3.1 beschrieben mit Indyco Builder erstellten Faktklassendatei vom Dateityp DFM und wird über seinen Dateipfad referenziert. Da für die Bearbeitung der Elemente in multidimensionalen Referenzmodellen die Verwendung von Indyco Builder vorgesehen ist, sind dafür keine XQuery-Funktionen vorgesehen.

Die Funktion `deleteModel` löscht das angegebene multidimensionale Referenzmodell aus der angegebenen BaseX-Datenbank. Das multidimensionale Referenzmodell wird dabei über den Namen der Faktklasse referenziert. Daraus folgt, dass die Namen der Faktklassen in den multidimensionalen Referenzmodellen eindeutig sein müssen. In jede BaseX-Datenbank kann eine beliebige Anzahl von multidimensionalen Referenzmodellen importiert werden.

Funktion	Parameter	Beschreibung
insertModel	$database xs:string $model xs:string	Fügt der angegebenen BaseX-Datenbank ($database) die angegebene Faktklassendatei ($model) im Dateiformat XML hinzu.
deleteModel	$database xs:string $model xs:string	Löscht die Faktklassendatei mit dem angegebenen Namen ($model) aus der angegebenen BaseX-Datenbank ($database).

Tabelle 3: Funktionen zur Verwaltung von multidimensionalen Referenzmodellen

3.2.2 Referenzkataloge für Hierarchien

Referenzkataloge für Hierarchien enthalten Hierarchien, um die die multidimensionalen Referenzmodelle über eine Anpassung ergänzt werden können. Die Liste der Hierarchien im Referenzkatalog ist jedoch nicht vollständig. Das heißt, es ist möglich, einem multidimensionalen Referenzmodell über eine Anpassung eine neue, direkt erstellte Hierarchie zuzuordnen, die nicht im Referenzkatalog für Hierarchien enthalten ist. Näheres zur Anpassung multidimensionaler Referenzmodelle und zur direkten Erstellung von Hierarchien findet sich in Abschnitt 3.2.7.

Tabelle 4 zeigt die XQuery-Funktionen, die für die Verwaltung von Referenzkatalogen für Hierarchien zur Verfügung stehen. Einem über die bereits beschriebene Funktion `createCatalogues` erstellten Referenzkatalog für Hierarchien können über die Funktion `insertHierarchyToHierarchyCat` leere Hierarchien ohne Levels hinzugefügt werden. Diese können über die Funktion `deleteHierarchyFromHierarchyCatalogue` auch wieder entfernt werden.

Die Funktion `insertLevelToHierarchyCatalogue` ermöglicht es, den einzelnen Hierarchien im Referenzkatalog Levels zuzuordnen. In BIRD ist definiert, dass jeder Hierarchie mindestens ein Level zugeordnet werden muss. Über die Angabe einer Notiz zu einem Level kann wie in Abschnitt 3.1.1 erläutert festgelegt werden, dass der Level der angegebenen Hierarchie verpflichtend zugeordnet ist. Hinzugefügte Levels können über die Funktion `deleteLevelFromHierarchyCatalogue` wieder entfernt werden.

Mit der Funktion `insertSuperLevelToHierarchyCatalogue` kann einem Level innerhalb des Referenzkatalogs für Hierarchien ein Superlevel zugeordnet werden. Auf diese Weise wird die Granularität der einzelnen Levels bestimmt. Der Sublevel ist bei dieser Funktion eindeutig identifizierbar unter Angabe des Hierarchienamens anzugeben (`Hierarchiename.Levelname`). Für den Superlevel ist lediglich der Name des Levels anzugeben. Wie in BIRD definiert, muss jede Hierarchie einen eindeutigen Basislevel besitzen. Das bedeutet, dass es für jede Hierarchie genau einen Level geben muss, der kein Superlevel eines anderen Levels ist. Die Namen der Basislevels müssen dabei

nicht nur innerhalb der Hierarchie, sondern auch hierarchieübergreifend eindeutig sein, um die volle Funktionalität der erstellten Implementierung zu gewährleisten.

Die Funktion **`insertAttributeToHierarchyCatalogue`** fügt analog zur letztgenannten Funktion dem angegebenen Level ein Attribut hinzu. Wieder ist der Level eindeutig identifizierbar anzugeben (**`Hierarchiename.Levelname`**). Über die Erfassung einer Notiz zum eingefügten Attribut kann festgelegt werden, dass dieses dem Level verpflichtend zugeordnet wird.

Für das Entfernen von Superlevels und Attributen aus dem Referenzkatalog für Hierarchien ist keine XQuery-Funktion vorgesehen. Um die Prüfung auf Korrektheit zu erleichtern, muss der gesamte Level entfernt und neu angelegt werden, wenn seine Superlevels oder Attribute im Referenzkatalog verändert werden sollen. Wird ein Level über die Funktion **`deleteLevelFromHierarchyCatalogue`** entfernt, wird er gleichzeitig auch als Superlevel aller anderen Levels innerhalb seiner Hierarchie gelöscht.

Um Anpassungen von multidimensionalen Referenzmodellen über die vorliegende BIRD-Implementierung vornehmen zu können, ist es notwendig, in jeder BaseX-Datenbank genau einen Referenzkatalog für Hierarchien anzulegen. Aus diesem Grund wurde keine XQuery-Funktion für das Löschen eines Referenzkatalogs für Hierarchien umgesetzt.

Funktion	Parameter	Beschreibung
insertHierarchy ToHierarchyCat	$database xs:string $name xs:string $description xs:string	Fügt dem Hierarchiekatalog der angegebenen BaseX-Datenbank ($database) eine Hierarchie mit dem angegebenen Namen ($name) und der angegebenen Beschreibung ($description) hinzu.
deleteHierarchy FromHierarchyCat	$database xs:string $hierarchy xs:string	Löscht die angegebene Hierarchie ($hierarchy) aus dem Hierarchiekatalog der angegebenen BaseX-Datenbank ($database).
insertLevel ToHierarchyCat	$database xs:string $name xs:string $description xs:string $sampleValues xs:string $note xs:string	Fügt der Hierarchie des angegebenen Levels ($name) im Hierarchiekatalog der angegebenen BaseX-Datenbank ($database) einen Level mit dem angegebenen Namen ($name), der angegebenen Beschreibung ($description) und Notiz ($note) und den angegebenen Beispielwerten ($sampleValues) hinzu.
deleteLevel FromHierarchyCat	$database xs:string $level xs:string	Löscht den angegebenen Level ($level) aus der entsprechenden Hierarchie im Hierarchiekatalog der angegebenen Datenbank ($database).
insertSuperLevel ToHierarchyCat	$database xs:string $level xs:string $superLevel xs:string	Fügt dem angegebenen Level ($level) in der entsprechenden Hierarchie im Hierarchiekatalog der angegebenen BaseX-Datenbank ($database) den angegebenen Superlevel ($superLevel) hinzu.
insertAttribute ToHierarchyCat	$database xs:string $level xs:string $name xs:string $description xs:string $sampleValues xs:string $note xs:string	Fügt dem angegebenen Level ($level) in der entsprechenden Hierarchie im Hierarchiekatalog der angegebenen BaseX-Datenbank ($database) ein Attribut mit dem angegebenen Namen ($name), der angegebenen Beschreibung ($description) und Notiz ($note) und den angegebenen Beispielwerten ($sampleValues) hinzu.

Tabelle 4: Funktionen zur Verwaltung von Referenzkatalogen für Hierarchien

Liste 3 zeigt die Struktur eines Referenzkatalogs für Hierarchien mit Hierarchien, die das in Abschnitt 3.1 vorgestellte beispielhafte multidimensionale Referenzmodell erweitern könnten. Der Referenzkatalog enthält zwei Hierarchien, nämlich **`ReturnJust`** (Return Justification – Rücksendegrund) und **`DocumentType`** (Dokumententyp) wobei die erstere drei Levels, die letztere nur einen enthält. Dem Basislevel der Hierarchie **`ReturnJust`** sind zwei Superlevels zugeordnet, von denen keines einen weiteren Superlevel aufweist. Der Level der Hierarchie **`DocumentType`** ist als verpflichtend

gekennzeichnet. Diesem Level ist außerdem das Attribut **comment** (Kommentar) zugeordnet, welches allerdings nicht verpflichtend ist.

```
01 <hierarchyCatalogue>
02   <hierarchy Name="ReturnJust" Description="ReturnJustification">
03     <levels>
04       <level Name="returnJust" Description="ReturnJustification" SampleValues="" Note="">
05         <superLevels>
06           <level Name="category"/>
07           <level Name="accountability"/>
08         </superLevels>
09         <attributes/>
10       </level>
11         <level Name="category" Description="" SampleValues="" Note="">
12           <superLevels/>
13           <attributes/>
14       </level>
15       <level Name="accountability" Description="" SampleValues="" Note="">
16         <superLevels/>
17         <attributes/>
18       </level>
19     </levels>
20   </hierarchy>
21   <hierarchy Name="DocumentType" Description="">
22     <levels>
23       <level Name="documentType" Description="" SampleValues="" Note="mandatory">
24         <superLevels/>
25         <attributes>
26           <attribute Name="comment" Description="" SampleValues="" Note=""/>
27         </attributes>
28       </level>
29     </levels>
30   </hierarchy>
31 </hierarchyCatalogue>
```

Liste 3: Referenzkatalog für Hierarchien in multidimensionalen Referenzmodellen

Vergleicht man den Aufbau des Referenzkatalogs für Hierarchien in Liste 3 mit dem der von Indyco Builder erstellten Faktklassendatei in Liste 2, zeigt sich, dass die auf Basis von BIRD erstellte Struktur der Hierarchien einfacher visuell lesbar ist. Es wäre möglich, auch die Faktklassendateien auf eine ähnliche Art und Weise darzustellen. Im Gegenzug wäre es auch möglich, die über Indyco Builder erstellten Referenzhierarchien anstelle eines Hierarchiekatalogs zu verwenden. Die erstellte Implementierung setzt dies nicht um, sieht aber eine einfache Erweiterung um diese Funktionalität vor.

3.2.3 Referenzkataloge für Kennzahlen

Wie für Hierarchien gibt es auch für Kennzahlen Referenzkataloge. Diese enthalten sowohl Basiskennzahlen als auch berechnete Kennzahlen, die den Faktklassen der multidimensionalen Referenzmodelle zugeordnet werden können. Wie in Abschnitt 3.1.1 erläutert, werden in der vorliegenden Implementierung über Indyco Builder den Faktklassen nur Basiskennzahlen zugeordnet. Die Zuordnung berechneter Kennzahlen aus dem Referenzkatalog für Kennzahlen erfolgt über eine Anpassung des entsprechenden multidimensionalen Referenzmodells.

Die Liste der berechneten Kennzahlen im Referenzkatalog ist vollständig. Dies bedeutet, dass im Zuge einer Anpassung eines multidimensionalen Referenzmodells nur berechnete Kennzahlen aus dem Referenzkatalog hinzugefügt werden können. Es ist jedoch möglich, die Berechnungsregeln der berechneten Kennzahlen über die Anpassung eines multidimensionalen Referenzmodells neu zu definieren. Die Liste der Basiskennzahlen im Referenzkatalog ist dagegen nicht vollständig. Wie bei den Hierarchien ist es möglich, Basiskennzahlen direkt in einer Anpassung für ein multidimensionales Referenzmodell zu erstellen. Dazu findet sich mehr in Abschnitt 3.2.7.

In Tabelle 5 sind die XQuery-Funktionen für die Erstellung und Verwaltung von Referenzkatalogen für Kennzahlen aufgelistet. Die Funktionen **`insertArithmeticMeasureToMeasureCat`**, **`insertArithmeticMeasureToMeasureCat`**, **`insertArithmeticMeasureToMeasureCat`** und **`insertArithmeticMeasureToMeasureCat`** dienen dem Einfügen von arithmetischen, kumulativen, Vergleichs- beziehungsweise Basiskennzahlen in einen über die Funktion **`createCatalogues`** erstellten Referenzkatalog für Kennzahlen. Besonders bei Berechnungsregeln und Verbundbedingungen ist darauf zu achten, XML-spezifische Sonderzeichen mit entsprechenden Unterbrechungszeichen zu erfassen.

Mit der Funktion **`deleteMeasureFromMeasureCat`** können Kennzahlen wieder aus dem Referenzkatalog entfernt werden. Dabei ist es nicht notwendig, den Typ der Kennzahl anzugeben. Es werden alle Kennzahlen gelöscht, die den angegebenen Namen tragen. Daraus ergibt sich, dass die Namen der Kennzahlen im Referenzkatalog sowohl innerhalb eines Kennzahltyps als auch typübergreifend eindeutig sein müssen.

Funktion	Parameter	Beschreibung
insertArithmeticMeasure ToMeasureCat	$database xs:string $name xs:string $description xs:string $calculationRule xs:string	Fügt dem Kennzahlkatalog der angegebenen BaseX-Datenbank ($database) eine arithmetische Kennzahl mit dem angegebenen Namen ($name) und der angegebenen Beschreibung ($description) und Berechnungsregel ($calculationRule) hinzu.
insertCumulativeMeasure ToMeasureCat	$database xs:string $name xs:string $description xs:string $calculationRule xs:string	Fügt dem Kennzahlkatalog der angegebenen BaseX-Datenbank ($database) eine kumulative Kennzahl mit dem angegebenen Namen ($name) und der angegebenen Beschreibung ($description) und Berechnungsregel ($calculationRule) hinzu.
insertComparisonMeasure ToMeasureCat	$database xs:string $name xs:string $description xs:string $calculationRule xs:string $joinCondition xs:string	Fügt dem Kennzahlkatalog der angegebenen BaseX-Datenbank ($database) eine Vergleichskennzahl mit dem angegebenen Namen ($name) und der angegebenen Beschreibung ($description), Berechnungsregel ($calculationRule) und Verbundbedingung ($joinCondition) hinzu.
insertBaseMeasure ToMeasureCat	$database xs:string $name xs:string $description xs:string	Fügt dem Kennzahlkatalog der angegebenen BaseX-Datenbank ($database) eine Basiskennzahl mit dem angegebenen Namen ($name) und der angegebenen Beschreibung ($description) hinzu.
deleteMeasure FromMeasureCat	$database xs:string $measure xs:string	Löscht die angegebene Kennzahl ($measure) aus dem Kennzahlkatalog der angegebenen BaseX-Datenbank ($database)
markMandatoryOverMeasure InMeasureCat	$database xs:string $measure xs:string $overMeasure xs:string	Kennzeichnet die letztere angegebene Kennzahl ($overMeasure) für die Berechnungsregel der ersteren angegebenen Kennzahl ($measure) im Kennzahlkatalog der angegebenen BaseX-Datenbank als verpflichtend.
markMandatoryOverAttribute InMeasureCat	$database xs:string $measure xs:string $overAttribute xs:string	Kennzeichnet das angegebene Attribut ($overAttribute) für die Berechnungsregel der angegebenen Kennzahl ($measure) im Kennzahlkatalog der angegebenen BaseX-Datenbank als verpflichtend.

Tabelle 5: Funktionen zur Verwaltung von Referenzkatalogen für Kennzahlen

Im Fall von berechneten Kennzahlen ist es nicht notwendig, beim Einfügen der Kennzahl in den Referenzkatalog die Modellelemente, die für die Berechnung der Kennzahl benötigt werden, manuell anzugeben. Diese werden über die in Abschnitt 3.2.10 erläuterte Funktion **`parseCatalogues`** automatisch generiert.

Nach Anwendung der Funktion **`parseCatalogues`** kann über die Funktion **`markMandatoryOverMeasureInMeasureCat`** für die einzelnen Kennzahlen im Referenzkatalog festgelegt werden, welche Kennzahlen verpflichtend in der Berechnungsregel vorkommen müssen, das heißt, welche Kennzahlen bei einer eventuellen Neudefinition der berechneten Kennzahl im Zuge einer Anpassung eines multidimensionalen Referenzmodells unbedingt vorhanden sein müssen. Die Funktion **`markMandatoryOverAttributeInMeasureCat`** erfüllt die selbe Aufgabe analog für die Attribute, auf die die Berechnungsregel der jeweiligen berechneten Kennzahl verweist.

```
01 <measureCatalogue>
02   <calculatedMeasures>
03     <arithmeticMeasures>
04       <measure Name="grossValue" Description="" CalculationRule="Fact.netValue +
  Fact.vat">
05         <overMeasures>
06           <measure Name="netValue" IsMandatory=""/>
07           <measure Name="vat" IsMandatory=""/>
08         </overMeasures>
09         <overAttributes/>
10       </measure>
11     </arithmeticMeasures>
12     <cumulativeMeasures>
13       <measure Name="averageValue" Description="" CalculationRule="AVG(Fact.netValue)
  OVER (PARTITION BY Fact.Product, Fact.Customer, Fact.Currency, Fact.Agent, Time.dates
  ORDER BY Time.dates)">
14         <overMeasures>
15           <measure Name="netValue" IsMandatory=""/>
16         </overMeasures>
17         <overAttributes/>
18         <overHierarchies>
19           <hierarchy Name="Product"/>
20           <hierarchy Name="Customer"/>
21           <hierarchy Name="Currency"/>
22           <hierarchy Name="Agent"/>
23         </overHierarchies>
24         <overLevels>
25           <level Name="dates" Hierarchy="Time"/>
26         </overLevels>
27       </measure>
28     </cumulativeMeasures>
29     <comparisonMeasures>
30       <measure Name="annualGrowth" Description="" CalculationRule="Fact.quantity -
  ComparisonFact.quantity" JoinCondition="Fact.Product = ComparisonFact.Product AND
  Time.years = (ComparisonTime.years + 1)">
31         <overMeasures>
32           <measure Name="quantity" IsMandatory=""/>
33         </overMeasures>
34         <ovrAttributes/>
35         <joinHierarchies>
36           <hierarchy Name="Product"/>
37         </joinHierarchies>
38         <joinLevels>
39           <level Name="years" Hierarchy="Time"/>
40         </joinLevels>
41       </measure>
42     </comparisonMeasures>
43   </calculatedMeasures>
44   <baseMeasures>
45     <measure Id="f10be446-2aed-4c7e-847a-00a25e4976fa" Name="vat" Description=""/>
46   </baseMeasures>
47 </measureCatalogue>
```

Liste 4: Referenzkatalog für Kennzahlen in multidimensionalen Referenzmodellen

Damit die volle Funktionalität der vorliegenden BIRD-Implementierung genutzt werden kann, ist für jede BaseX-Datenbank genau ein Referenzkatalog für Kennzahlen anzulegen. Aus diesem Grund ist keine XQuery-Funktion für das Löschen dieser Referenzkataloge vorgesehen.

Einen beispielhaften Referenzkatalog für Kennzahlen, dessen Kennzahlen dem Einsatzgebiet des verwendeten Beispielmodells angepasst sind, zeigt Liste 4. Der Katalog enthält eine Kennzahl jedes der in BIRD definierten Kennzahltypen, nämlich die arithmetische Kennzahl `grossValue` (Bruttowert), die kumulative Kennzahl `averageValue` (Durchschnittswert), die Vergleichskennzahl `annualGrowth` (jährliches Wachstum) und die Basiskennzahl `vat` (Value-Added Tax – Mehrwertsteuer).

Es ist anzumerken, dass Liste 4 die Struktur des Katalogs nach Anwendung der Funktion `parseCatalogues` darstellt. Durch Anwendung dieser Funktion werden die Subelemente der berechneten Kennzahlen den Berechnungsgrundlagen und Verbundbedingungen entsprechend befüllt. Details zur Funktion `parseCatalogues` finden sich in Abschnitt 3.2.10.

3.2.4 Referenzkataloge für Prädikate

Referenzkataloge für Prädikate ähneln den Referenzkatalogen für Kennzahlen in ihrer Struktur und auch in den darauf anwendbaren XQuery-Funktionen. Sie enthalten Prädikate, die die Fakten in einem Data Warehouse anhand der Werte von Kennzahlen, Levels und Attributen filtern können. Mithilfe von Prädikaten ist es also möglich, nur Fakten aus einem bestimmten Wertebereich abzurufen. Die Filterung der Fakten findet dabei nicht schon im multidimensionalen Modell, sondern erst über die Erstellung von Analysesituationen statt. Näheres zum Aufbau von Analysesituationen und deren Anpassung findet sich in Abschnitt 3.2.5 respektive Abschnitt 3.2.8.

Funktion	Parameter	Beschreibung
insertDimensionPredicate ToPredicateCat	$database xs:string $name xs:string $description xs:string $expression xs:string	Fügt dem Prädikatkatalog der angegebenen BaseX-Datenbank ($database) ein Dimensionsprädikat mit dem angegebenen Namen ($name) und Ausdruck ($expression) und der angegebenen Beschreibung ($description) hinzu.
insertMultidimensionalPredicate ToPredicateCat	$database xs:string $name xs:string $description xs:string $expression xs:string	Fügt dem Prädikatkatalog der angegebenen BaseX-Datenbank ($database) ein multidimensionales Prädikat mit dem angegebenen Namen ($name) und Ausdruck ($expression) und der angegebenen Beschreibung ($description) hinzu.
insertMeasurePredicate ToPredicateCat	$database xs:string $name xs:string $description xs:string $expression xs:string	Fügt dem Prädikatkatalog der angegebenen BaseX-Datenbank ($database) ein Kennzahlprädikat mit dem angegebenen Namen ($name) und Ausdruck ($expression) und der angegebenen Beschreibung ($description) hinzu.
deletePredicate FromPredicateCat	$database xs:string $predicate xs:string	Löscht das angegebene Prädikat aus dem Prädikatkatalog der angegebenen BaseX-Datenbank ($database).
markMandatoryMeasure InPredicateCat	$database xs:string $predicate xs:string $measure xs:string	Kennzeichnet die angegebene Kennzahl ($measure) für den Ausdruck des angegebenen Prädikats ($predicate) im Prädikatkatalog der angegebenen BaseX-Datenbank als verpflichtend.
markMandatoryAttribute InPredicateCat	$database xs:string $predicate xs:string $attribute xs:string	Kennzeichnet das angegebene Attribut ($attribute) für den Ausdruck des angegebenen Prädikats ($predicate) im Prädikatkatalog der angegebenen BaseX-Datenbank als verpflichtend.

Tabelle 6: Funktionen zur Verwaltung von Referenzkatalogen für Prädikate

Wie bei den berechneten Kennzahlen im Referenzkatalog für Kennzahlen sind die Prädikate im Referenzkatalog für Prädikate vollständig aufgezählt. Das bedeutet, dass bei der Erstellung und Anpassung von Analysesituationen nur Prädikate verwendet werden können, die im Referenzkatalog gespeichert sind. Eine weitere Parallele zu den berechneten Kennzahlen ist, dass die Prädikatausdrücke im Zuge einer Anpassung einer Analysesituation neu definiert werden können. So kann eine bedarfsgerechte Anpassung der Prädikate ermöglicht werden, obwohl die Zahl der anwendbaren Prädikate begrenzt ist.

Tabelle 6 zeigt die für die Erstellung und Verwaltung von Referenzkatalogen für Prädikate verfügbaren XQuery-Funktionen. Nachdem über die Funktion **`createCatalogues`** ein Referenzkatalog erstellt wurde, können diesem mit der Funktion **`insertDimensionPredicateToPredicateCat`** Dimensionsprädikate, mit der Funktion **`insertMultidimensionalPredicateToPredicateCat`** multidimensionale Prädikate, und mit der Funktion **`insertMeasurePredicateToPredicateCat`** Kennzahlprädikate hinzugefügt werden. Wie bei den Berechnungsregeln und Verbundbedingungen der Kennzahlen müssen auch bei den Ausdrücken der Prädikate XML-spezifische Sonderzeichen mit entsprechenden Unterbrechungszeichen erfasst werden.

Die Funktion **`deletePredicateFromPredicateCat`** ermöglicht das Löschen von Prädikaten aus dem Referenzkatalog. Der Typ des Prädikats muss dabei nicht angegeben werden. Daraus folgt, dass bei Anwendung der Funktion **`deletePredicateFromPredicateCat`** alle Prädikate aus dem Referenzkatalog entfernt werden, die den angegebenen Namen tragen. Die Namen der Prädikate müssen im Referenzkatalog demnach auch typübergreifend eindeutig sein.

```
01 <predicateCatalogue>
02   <dimensionPredicates>
03     <predicate Name="groceries" Description="" Expression="Product.category = 'grocery'">
04       <attributes>
05         <attribute Name="category" Hierarchy="Product" IsMandatory=""/>
06       </attributes>
07       <hierarchy Name="Product"/>
08     </predicate>
09   </dimensionPredicates>
10   <multidimensionalPredicates>
11     <predicate Name="beveragesNovember2015" Description="" Expression="Product.type =
  'beverage' AND Time.months = 'november' AND Time.years = '2015'">
12       <hierarchies>
13         <hierarchy Name="Product"/>
14         <hierarchy Name="Time"/>
15       </hierarchies>
16       <attributes>
17         <attribute Name="type" Hierarchy="Product" IsMandatory=""/>
18         <attribute Name="months" Hierarchy="Time" IsMandatory=""/>
19         <attribute Name="years" Hierarchy="Time" IsMandatory=""/>
20       </attributes>
21     </predicate>
22   </multidimensionalPredicates>
23   <measurePredicates>
24     <predicate Name="premiumBeverages2015" Description="" Expression="Product.type =
  'beverage' AND Time.years = '2015' AND Fact.netValue = 50">
25       <hierarchies>
26         <hierarchy Name="Product"/>
27         <hierarchy Name="Time"/>
28       </hierarchies>
29       <attributes>
30         <attribute Name="type" Hierarchy="Product" IsMandatory=""/>
31         <attribute Name="years" Hierarchy="Time" IsMandatory=""/>
32       </attributes>
33       <measures>
34         <measure Name="netValue" IsMandatory=""/>
35       </measures>
36     </predicate>
37   </measurePredicates>
38 </predicateCatalogue>
```

Liste 5: Referenzkatalog für Prädikate in multidimensionalen Referenzmodellen

Die bereits in den beiden vorhergehenden Abschnitten erwähnte Funktion **parseCatalogues** greift auch auf die Prädikate im Referenzkatalog zu. Wie bei den Kennzahlen im Referenzkatalog für Kennzahlen werden mit dieser Funktion die für die Verwendung der Prädikate notwendigen Elemente des multidimensionalen Modells aus den Prädikatausdrücken ausgelesen und so den jeweiligen Prädikaten im Referenzkatalog zugewiesen. Es ist also nicht notwendig, diese benötigten Modellelemente beim Einfügen eines Prädikats in den Referenzkatalog manuell anzugeben.

Die Funktionen **markMandatoryMeasureInPredicateCat** und **markMandatoryAttributeInPredicateCat** können erst nach der Anwendung der Funktion **parseCatalogues** verwendet werden. Sie ermöglichen es, festzulegen, welche Kennzahlen beziehungsweise Attribute in den Ausdrücken bestimmter Prädikate verpflichtend, also auch bei einer Neudefinition des Prädikatausdrucks im Rahmen der Anpassung eines multidimensionalen Referenzmodells, verwendet werden müssen.

Wie für Referenzkataloge für Hierarchien und Kennzahlen gilt auch für Referenzkataloge für Prädikate, dass jeder BaseX-Datenbank genau ein solcher Referenzkatalog zugewiesen werden muss, um die volle Funktionalität der vorliegenden Implementierung nutzen zu können. Deshalb wird auch für Referenzkataloge für Prädikate keine XQuery-Funktion zum Löschen ganzer Kataloge zur Verfügung gestellt.

Liste 5 zeigt einen beispielhaften Referenzkatalog für Prädikate abgestimmt auf das Einsatzgebiet des verwendeten Beispielreferenzmodells. Die Struktur des Referenzkatalogs ähnelt der in Liste 4 gezeigten eines Referenzkatalogs für Kennzahlen. Für jeden Prädikattyp ist ein Prädikat erfasst, nämlich das Dimensionsprädikat **groceries** (Lebensmittel), das multidimensionale Prädikat **beveragesNovember2015** (Getränke im Monat November 2015) und die Kennzahlprädikat **premiumBeverages2015** (Premiumgetränke im Jahr 2015).

Die in Liste 5 abgebildete Struktur ist die eines Referenzkatalogs nach Anwendung der Funktion **parseCatalogues**, die wie bereits erwähnt in Abschnitt 3.2.10 definiert wird. Durch Anwendung dieser Funktion werden die Subelemente der Prädikate entsprechend der Prädikatausdrücke befüllt.

3.2.5 Referenzkataloge für Analysesituationen

Analysesituationen stellen Sichten auf bestimmte Teilbereiche der Fakten in einem Data Warehouse dar. Durch die Verbindung von multidimensionalen Referenzmodellen mit Prädikaten und die Auswahl geeigneter Kennzahlen und Granularitäten können diese Sichten gestaltet werden.

Um die Verwendung von Analysesituationen und Analysegraphen zu erleichtern, sieht die vorliegende Implementierung Referenzkataloge für Analysesituationen vor. Jede der in einem Analysegraphen verwendeten Analysesituationen muss im entsprechenden Referenzkatalog für Analysesituationen enthalten sein. Es besteht jedoch die Möglichkeit, die Analysesituationen im Referenzkatalog bedarfsgerecht anzupassen. Abschnitt 3.2.8 enthält Details zu den Anpassungsmöglichkeiten für Analysesituationen.

Tabelle 7 listet die XQuery-Funktionen für die Erstellung und Verwaltung von Referenzkatalogen für Analysesituationen. Einem über die Funktion **createCatalogues** erstellten Referenzkatalog für Analysesituationen können über die Funktion **insertSituationToSituationCat** Analysesituationen hinzugefügt werden, wobei anzugeben ist, auf welche Faktklasse sich die Analysesituation bezieht. Das Löschen von Analysesituationen wird über die Funktion **deleteSituationFromSituationCat** ermöglicht.

Mit der Funktion **insertPredicateToSituationCat** können einer Analysesituation im Referenzkatalog multidimensionale Prädikate und Kennzahlprädikate zugewiesen werden. Analog erfolgt die Zuweisung von Kennzahlen über die Funktion **insertMeasureToSituationCat** und die Zuweisung von Dimensionen über die Funktion **insertDimensionToSituationCat**. Bei letzterer sind neben dem Namen der Dimension auch Granularitätslevel, Dice-Level und Dice-Knoten anzugeben.

Auch für das Löschen von Elementen aus einer gegebenen Analysesituation im Referenzkatalog sind XQuery-Funktionen vorgesehen. Mit **deletePredicateFromSituationCat** können Prädikate, mit

`deleteDimensionFromSituationCat` Dimensionen und mit `deleteMeasureFromSituationCat` Kennzahlen entfernt werden.

Funktion	Parameter	Beschreibung
insertSituation ToSituationCat	$database xs:string $name xs:string $factClass xs:string	Fügt dem Analysesituationskatalog der angegebenen BaseX-Datenbank ($database) eine Analysesituation mit dem angegebenen Namen ($name), hinzu, die sich auf die angegebene Faktklasse ($factClass) bezieht.
deleteSituation FromSituationCat	$database xs:string $situation xs:string	Löscht die angegebene Analysesituation aus dem Analysesituationskatalog der angegebenen BaseX-Datenbank ($database).
insertPredicate ToSituationCat	$database xs:string $situation xs:string $predicate xs:string	Fügt der angegebenen Analysesituation ($situation) im Analysesituationskatalog der angegebenen BaseX-Datenbank ($database) das angegebene Prädikat ($predicate) hinzu.
deletePredicate FromSituationCat	$database xs:string $predicate xs:string	Löscht das angegebene Prädikat ($predicate) aus der angegebenen Analysesituation ($situation) im Analysesituationskatalog der angegebenen BaseX-Datenbank ($database).
insertMeasure ToSituationCat	$database xs:string $situation xs:string $measure xs:string	Fügt der angegebenen Analysesituation ($situation) im Analysesituationskatalog der angegebenen BaseX-Datenbank ($database) die angegebene Kennzahl ($measure) hinzu.
deleteMeasure FromSituationCat	$database xs:string $situation xs:string $measure xs:string	Löscht die angegebene Kennzahl ($measure) aus der angegebenen Analysesituation ($situation) im Analysesituationskatalog der angegebenen BaseX-Datenbank ($database).
insertAggregation ToSituationCat	$database xs:string $situation xs:string $measure xs:string $aggregationOperation xs:string	Fügt der angegebenen Kennzahl ($measure) in der angegebenen Analysesituation ($situation) im Analysesituationskatalog der angegebenen BaseX-Datenbank ($database) die angegebene Aggregationsoperation ($aggregationOperation) hinzu.
insertDimension ToSituationCat	$database xs:string $situation xs:string $name xs:string $granularityLevel xs:string $diceLevel xs:string $diceNode xs:string	Fügt der angegebenen Analysesituation ($situation) im Analysesituationskatalog der angegebenen BaseX-Datenbank ($database) eine Dimension mit dem angegebenen Namen ($name), Granularitätslevel ($granularityLevel), Dice-Level ($diceLevel) und Dice-Knoten ($diceNode) hinzu.
deleteDimension FromSituationCat	$database xs:string $situation xs:string $dimension xs:string	Löscht die angegebene Dimension ($dimension) aus der angegebenen Analysesituation ($situation) im Analysesituationskatalog der angegebenen BaseX-Datenbank ($database).
insertSliceCondition ToSituationCat	$database xs:string $situation xs:string $dimension xs:string $sliceCondition xs:string	Fügt der angegebenen Dimension ($dimension) in der angegebenen Analysesituation ($situation) im Analysesituationskatalog der angegebenen BaseX-Datenbank ($database) die angegebene Slice-Bedingung ($sliceCondition) hinzu.

Tabelle 7: Funktionen zur Verwaltung von Referenzkatalogen für Analysesituationen

Nachdem einer Analysesituation eine Kennzahl zugewiesen wurde, können für diese über die Funktion `insertAggregationToSituationCat` Aggregationsoperationen festgelegt werden. Ein Löschen dieser Aggregationsoperationen ist nicht über eine XQuery-Funktion vorgesehen. Sollen die Aggregationsoperationen einer Kennzahl verändert werden, ist die ganze Kennzahl über die Funktion `deleteMeasureFromSituationCat` zu entfernen und über die Funktion `insertMeasureToSituationCat` neu anzulegen.

Ähnlich wie das Hinzufügen von Aggregationsoperationen zu Kennzahlen über die Funktion `insertAggregationToSituationCat` erfolgt das Hinzufügen von Slice-Bedingungen zu Dimensionen einer Analysesituation über die Funktion `insertSliceConditionToSituationCat`. Für das Löschen dieser Slice-Bedingungen ist ebenfalls keine XQuery-Funktion vorgesehen. Sollen die Slice-Bedingungen einer Dimension verändert werden, muss die Dimension aus der Analysesituation entfernt und anschließend neu hinzugefügt werden.

```
001 <analysisSituationCatalogue>
002   <analysisSituation Name="Situation1" FactClass="Invoice">
003     <measures>
004       <measure Name="netValue">
005         <aggregationOperations>
006           <aggregationOperation Name="AVG"/>
007         </aggregationOperations>
008       </measure>
009       <measure Name="annualGrowth">
010         <aggregationOperations/>
011       </measure>
012       <measure Name="grossValue">
013         <aggregationOperations>
014           <aggregationOperation Name="AVG"/>
015         </aggregationOperations>
016       </measure>
017       <measure Name="?m1">
018         <aggregationOperations>
019           <aggregationOperation Name="SUM"/>
020         </aggregationOperations>
021       </measure>
022     </measures>
023     <predicates>
024       <predicate Name="premiumBeverages2015"/>
025       <predicate Name="?p1"/>
026     </predicates>
027     <dimensions>
028       <dimension Name="Product" GranularityLevel="?l1" DiceLevel="?l2" DiceNode="?n1">
029         <sliceConditions>
030           <sliceCondition Name="groceries"/>
031         </sliceConditions>
032       </dimension>
033     </dimensions>
034   </analysisSituation>
035   <analysisSituation Name="Situation2" FactClass="Invoice">
        ...
073   </analysisSituation>
074   <analysisSituation Name="Situation3" FactClass="Invoice">
        ...
113   </analysisSituation>
114 </analysisSituationCatalogue>
```

Liste 6: Referenzkatalog für Analysesituationen

Wie in Abschnitt 2.1.3 erläutert, ermöglicht BIRD bei der Zuweisung von Elementen zu Analysesituationen die Verwendung von Variablen, die erst bei der Anpassung der Analysesituation an den konkreten Informationsbedarf oder auch erst bei der tatsächlichen Durchführung der Analyse durch konkrete Werte ersetzt werden müssen. Dabei ist festzuhalten, dass für Faktklassen und Dimensionen keine Variablen angegeben werden dürfen.

Variablen werden in der vorliegenden Implementierung insofern umgesetzt, als alle Elemente einer Analysesituation, deren Name mit einem Fragezeichen (`?`) beginnt, als Variable angesehen werden. Daraus folgt, dass bei der Anpassung einer Analysesituation die Werte von Granularitätslevels, Dice-Levels und Dice-Knoten nur dann verändert werden dürfen, wenn der Name des entsprechenden Elements im Referenzkatalog mit einem Fragezeichen beginnt. Außerdem werden alle Elemente, deren Name im Referenzkatalog mit einem Fragezeichen beginnt, also auch Prädikate, Kennzahlen und Aggregationsoperationen, bei der Prüfung auf Konsistenz von Faktklasse und Analysesituation außer Acht gelassen. Anzumerken ist, dass die Namen aller Variablen eindeutig sein müssen.

In Liste 6 ist ein beispielhafter Referenzkatalog für Analysesituationen abgebildet, der die verwendeten Beispieldaten aufgreift. Der Referenzkatalog enthält drei Analysesituationen, wobei aus Gründen der Übersichtlichkeit in Liste 6 nur die erste im Detail dargestellt wird. Für jede Analysesituation wird angegeben, auf welche Faktklasse, das heißt, auf welches multidimensionale Referenzmodell sie sich bezieht. Die dargestellte Analysesituation bezieht sich auf die Faktklasse **`Invoice`** aus dem bereits verwendeten Beispiel.

Die abgebildete Analysesituation verwendet die Kennzahlen **`netValue`**, **`annualGrowth`**, **`grossValue`** und die als Variable angegebene Kennzahl **`?m1`**. Für jede dieser Kennzahlen außer der Kennzahl **`annualGrowth`** ist eine Aggregationsoperation erfasst. Die Analysesituation verwendet außerdem das Kennzahlprädikat **`premiumBeverages2015`** und das als Variable angegebene Prädikat **`?p1`** sowie die Dimension **`Product`**. Für die Dimension **`Product`** ist das Dimensionsprädikat **`groceries`** als Slice-Bedingung angegeben. Granularitätslevel, Dice-Level und Dice-Knoten sind als Variablen angegeben, die wie auch die Variablen **`?m1`** und **`?m2`** entweder im Rahmen einer Anpassung oder bei der tatsächlichen Durchführung einer Analyse durch einen konkreten Wert ersetzt werden können.

3.2.6 Referenzkataloge für Analysegraphen

Wie Analysesituationen werden auch Analysegraphen in der vorliegenden Implementierung über Referenzkataloge verwaltet. Analysegraphen enthalten eine oder mehrere Analysesituationen verbunden über Navigationsschritte und können über Anpassungen verändert werden. Näheres zu Anpassungen für Analysegraphen findet sich in Abschnitt 3.2.9 Nach einer eventuellen Anpassung kann aus einem Analysegraphen ein ausführbarer Graph generiert werden. Dies wird in Abschnitt 3.2.10 erläutert. Um aus einem Analysegraphen einen ausführbaren Graphen zu erstellen, ist es notwendig, dass der Analysegraph im Referenzkatalog für Analysegraphen in der entsprechenden BaseX-Datenbank enthalten ist.

Tabelle 8 zeigt die XQuery-Funktionen, die für die Erstellung und Verwaltung von Referenzkatalogen für Analysegraphen zur Verfügung stehen. Nachdem mit der Funktion **`createCatalogues`** ein Referenzkatalog erstellt wurde, können diesem mit der Funktion **`insertGraphToGraphCat`** Analysegraphen hinzugefügt werden. Mit der Funktion **`deleteGraph-FromGraphCat`** werden hinzugefügte Graphen wieder entfernt.

Die Funktionen **`insertSituationToGraphCat`** und **`insertStepToGraphCat`** ermöglichen das Hinzufügen von Analysesituationen und Navigationsschritten zu Analysegraphen im Referenzkatalog. Bei Navigationsschritten sind dabei neben dem Namen des Navigationsschrittes auch die Ausgangs- und Zielanalysesituation anzugeben. Während die Analysesituationen, die den Analysegraphen im Referenzkatalog zugewiesen werden, dem Referenzkatalog für Analysesituationen entstammen müssen, sind die Navigationsschritte für jeden Analysegraphen frei zu definieren. Das heißt, es ist kein Referenzkatalog mit einer begrenzten Menge an Navigationsschritten vorgesehen.

Die Funktion **`deleteSituationFromGraphCat`** ermöglicht das Entfernen von Analysesituationen aus einem bestimmten Analysegraphen im Referenzkatalog. Die Funktion **`deleteStepFromGraphCat`** entfernt analog Navigationsschritte. Beim Entfernen von Analysesituationen und Navigationsschritten aus Analysegraphen ist wie auch beim Hinzufügen auf die Konsistenz der Analysesituationen und Navigationsschritte innerhalb des Analysegraphen zu achten: Jede Analysesituation muss über

Navigationsschritte direkt oder indirekt mit allen anderen Analysesituationen verbunden sein. Außerdem darf nur je ein Navigationsschritt von einer Analysesituation zu einer anderen führen.

Nachdem einem Analysegraphen im Referenzkatalog Analysesituationen und Navigationsschritte hinzugefügt wurden, können den einzelnen Navigationsschritten über die Funktion `insertOperationToGraphCat` Navigationsoperationen zugeordnet werden. Die möglichen Navigationsoperationen wurden in Abschnitt 2.1.5 in Tabelle 1 aufgelistet. Auch beim Einfügen von Navigationsoperationen ist darauf zu achten, dass die Analysesituationen und die Navigationsschritte mit den Navigationsoperationen zueinander konsistent sind. Alle Unterschiede zwischen der Ausgangs- und Zielanalysesituation eines Navigationsschrittes müssen in dessen Navigationsoperationen abgebildet werden. Die Parameter der Navigationsoperationen sind wie in Tabelle 1 dargestellt durch Kommas getrennt und ohne Leerzeichen zu erfassen. In der vorliegenden Implementierung erfolgt keine Prüfung auf Konsistenz der Navigationsoperationen, es wird auf Horschitz (2016) verwiesen.

Das Löschen von Navigationsoperationen aus Navigationsschritten ist nicht vorgesehen. Um die Navigationsoperationen eines Navigationsschrittes zu verändern, ist der gesamte Navigationsschritt über die Funktionen `deleteStepFromGraphCat` und `insertStepToGraphCat` aus dem Analysegraph zu entfernen beziehungsweise wieder hinzuzufügen.

Wie bei allen anderen Referenzkatalogen ist auch für Referenzkataloge für Analysegraphen keine XQuery-Funktion für das Löschen ganzer Referenzkataloge vorgesehen. Auch hier ist der Grund dafür, dass jeder BaseX-Datenbank genau ein Referenzkatalog für Analysegraphen zugeordnet sein muss. Das Löschen eines Referenzkatalogs aus einer bestehenden BaseX-Datenbank würde die Funktionalität der Implementierung beeinträchtigen.

Funktion	Parameter	Beschreibung
insertGraph ToGraphCat	$database xs:string $graph xs:string	Fügt dem Analysegraphkatalog der angegebenen BaseX-Datenbank ($database) den angegebenen Analysegraph ($graph) hinzu.
deleteGraph FromGraphCat	$database xs:string $graph xs:string	Löscht den angegebenen Analysegraph aus dem Analysegraphkatalog der angegebenen BaseX-Datenbank ($database).
insertSituation ToGraphCat	$database xs:string $graph xs:string $situation xs:string	Fügt dem angegebenen Analysegraph ($graph) im Analysegraphkatalog der angegebenen BaseX-Datenbank ($database) die angegebene Analysesituation ($situation) hinzu.
deleteSituation FromGraphCat	$database xs:string $graph xs:string $situation xs:string	Löscht die angegebene Analysesituation ($situation) aus dem angegebenen Analysegraph ($graph) im Analysegraphkatalog der angegebenen BaseX-Datenbank ($database).
insertStep ToGraphCat	$database xs:string $graph xs:string $name xs:string $source xs:string $target xs:string	Fügt dem angegebenen Analysegraph ($graph) im Analysegraphkatalog der angegebenen BaseX-Datenbank ($database) einen Navigationsschritt mit dem angegebenen Namen ($name) und der angegebenen Ausgangsanalysesituation ($source) und Zielanalysesituation ($target) hinzu.
deleteStep FromGraphCat	$database xs:string $graph xs:string $navigationStep xs:string	Löscht den angegebenen Navigationsschritt aus dem angegebenen Analysegraph ($graph) im Analysegraphkatalog der angegebenen BaseX-Datenbank ($database).
insertOperation ToGraphCat	$database xs:string $graph xs:string $navigationStep xs:string $navigationOperation xs:string	Fügt dem angegebenen Navigationsschritt ($navigationStep) im angegebenen Analysegraph ($graph) im Analysegraphkatalog der angegebenen BaseX-Datenbank ($database) die angegebene Navigationsoperation ($navigationOperation) hinzu.

Tabelle 8: Funktionen zur Verwaltung von Referenzkatalogen für Analysegraphen

Liste 7 zeigt einen beispielhaften Referenzkatalog für Analysegraphen, der Analysesituationen aus dem in Liste 6 dargestellten Referenzkatalog für Analysesituationen verwendet. Der Katalog enthält lediglich einen Graphen, bestehend aus zwei Analysesituationen. Die beiden Analysesituationen sind über einen Navigationsschritt miteinander verbunden. Dieser Navigationsschritt enthält drei Navigationsoperationen, welche die Unterschiede zwischen `Situation1` und `Situation2` widerspiegeln.

```
01 <analysisGraphCatalogue>
02   <analysisGraph Name="Graph1">
03     <analysisSituations>
04       <analysisSituation Name="Situation1"/>
05       <analysisSituation Name="Situation2"/>
06     </analysisSituations>
07     <navigationSteps>
08       <navigationStep Name="Step1" Source="Situation1" Target="Situation2">
09         <operations>
10           <operation Name="addMeasure(?m2,SUM)"/>
11           <operation Name="addPredicate(?p2)"/>
12           <operation Name="refocusSliceCondition(Product,groceries,?p3)"/>
13         </operations>
14       </navigationStep>
15     </navigationSteps>
16   </analysisGraph>
17 </analysisGraphCatalogue>
```

Liste 7: Referenzkatalog für Analysegraphen

3.2.7 Anpassungen für multidimensionale Referenzmodelle

Wie für die in den vorhergehenden Abschnitten diskutierten Referenzkataloge stellt die vorliegende BIRD-Implementierung XQuery-Funktionen zum Erstellen und Verwalten von Anpassungen für multidimensionale Referenzmodelle zur Verfügung. Im Gegensatz zu den Referenzkatalogen kann in jeder BaseX-Datenbank eine beliebige Anzahl solcher Modellanpassungen erfasst werden. Aus diesem Grund wurde auch eine Funktion zum Löschen einer Modellanpassung umgesetzt.

Tabelle 9 beschreibt die Funktionen `createModelCustomization` und `deleteModelCustomization` zum Erstellen und Löschen von Modellanpassungen. Beim Erstellen ist zusätzlich zum Namen der Modellanpassung auch der Name der Faktklasse, die dadurch angepasst wird, anzugeben.

Funktion	Parameter	Beschreibung
createModelCustomization	$database xs:string $name xs:string $fact xs:string	Erzeugt in der angegebenen BaseX-Datenbank ($database) eine leere Anpassung für multidimensionale Referenzmodelle mit dem angegebenen Namen ($name), die sich auf das multidimensionale Modell der angegebene Faktklasse ($fact) bezieht.
deleteModelCustomization	$database xs:string $modelCustomization xs:string	Löscht die angegebene Anpassung für multidimensionale Referenzmodelle ($modelCustomization) aus der angegebenen BaseX-Datenbank ($database).

Tabelle 9: Funktionen zur Verwaltung von Anpassungen für multidimensionale Referenzmodelle

Jede Modellanpassung enthält Listen von Kennzahlen, Hierarchien, Levels und Attributen, die bei Anwendung der Modellanpassung aus dem entsprechenden multidimensionalen Referenzmodell abgewählt werden. Die Funktionen zur Verwaltung dieser abgewählten Elemente sind in Tabelle 10 dargestellt.

Die Funktionen `insertDeselectedMeasureToModelCust` und `deleteDeselectedMeasureFromModelCust` ermöglichen das Hinzufügen respektive Entfernen von Kennzahlen zur Liste der abgewählten Kennzahlen der angegebenen Modellanpassung. Wie in Abschnitt 3.1.1 erläutert, enthalten die von

den Modellanpassungen referenzierten Faktklassen nur Basiskennzahlen, da nur diese über Indyco Builder verwaltet werden. Aus diesem Grund können in der Liste der abgewählten Kennzahlen in der Modellanpassung nur Basiskennzahlen angegeben werden. Kennzahlen, die in der Faktklasse als verpflichtend angegeben wurden, können nicht abgewählt werden.

Die Funktionen `insertDeselectedHierarchyToModelCust` und `deleteDeselectedHierarchyFromModelCust` erfüllen die selben Aufgaben wie die letztgenannten Funktionen in Bezug auf abgewählte Hierarchien. Wird eine Hierarchie abgewählt, werden damit auch alle Levels und Attribute der Hierarchie aus der Faktklasse abgewählt. Ist die Hierarchie der Faktklasse verpflichtend zugeordnet, kann sie nicht über die Modellanpassung abgewählt werden.

Funktion	Parameter	Beschreibung
insertDeselectedMeasure ToModelCust	$database xs:string $modelCustomization xs:string $measure xs:string	Fügt der Liste der abgewählten Kennzahlen der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Kennzahl ($measure) hinzu.
deleteDeselectedMeasure FromModelCust	$database xs:string $modelCustomization xs:string $measure xs:string	Löscht die angegebene Kennzahl ($measure) aus der Liste der abgewählten Kennzahlen der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).
insertDeselectedHierarchy ToModelCust	$database xs:string $modelCustomization xs:string $hierarchy xs:string	Fügt der Liste der abgewählten Hierarchien der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Hierarchie ($hierarchy) hinzu.
deleteDeselectedHierarchy FromModelCust	$database xs:string $modelCustomization xs:string $hierarchy xs:string	Löscht die angegebene Hierarchie ($hierarchy) aus der Liste der abgewählten Hierarchien der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).
insertDeselectedLevel ToModelCust	$database xs:string $modelCustomization xs:string $level xs:string	Fügt der Liste der abgewählten Levels der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) den angegebene Level ($level) hinzu.
deleteDeselectedLevel FromModelCust	$database xs:string $modelCustomization xs:string $level xs:string	Löscht den angegebenen Level ($level) aus der Liste der abgewählten Levels der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).
insertDeselectedAttribute ToModelCust	$database xs:string $modelCustomization xs:string $attribute xs:string	Fügt der Liste der abgewählten Attribute der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) das angegebene Attribute ($attribute) hinzu.
deleteDeselectedAttribute FromModelCust	$database xs:string $modelCustomization xs:string $attribute xs:string	Löscht das angegebene Attribut ($attribute) aus der Liste der abgewählten Attribute der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).

Tabelle 10: Funktionen zur Verwaltung von abgewählten Elementen in Modellanpassungen

Die Funktionen `insertDeselectedLevelToModelCust` ermöglicht es, einzelne Levels aus Hierarchien der Faktklasse abzuwählen, wenn diese in der entsprechenden Hierarchie nicht als verpflichtend gekennzeichnet wurden. Mit der Funktion `deleteDeselectedLevelFromModelCust` können Levels wieder

aus der Liste der abgewählten Levels der Modellanpassung entfernt werden. Bei beiden Funktionen ist der Level eindeutig identifizierbar unter Angabe des Hierarchienamens anzugeben (`Hierarchiename.Levelname`). Ist einem abgewählten Level ein Attribut zugeordnet, wird dieses bei Anwendung der Modellanpassung automatisch ebenfalls abgewählt.

Über die Funktionen `insertDeselectedAttributeToModelCust` und `deleteDeselectedAttribute-FromModelCust` können Attribute der Liste der abgewählten Attribute der Modellanpassung hinzugefügt respektive aus dieser entfernt werden. Attribute, die für einen Level als verpflichtend markiert wurden, können nicht abgewählt werden, sofern nicht der gesamte Level abgewählt wurde. Ist dies der Fall, muss das Attribut jedoch wie oben erwähnt nicht manuell in der Liste der abgewählten Attribute erfasst werden, sondern wird automatisch mit dem Level abgewählt. Sowohl bei der Funktion `insertDeselectedAttributeToModelCust` als auch bei der Funktion `deleteDeselectedLevel-FromModelCust` ist das Attribut eindeutig identifizierbar unter Angabe des Hierarchienamens anzugeben (`Hierarchiename.Attributname`). Die Funktionen `insertDeselectedLevelToModelCust` beziehungsweise `deleteDeselectedLevel-FromModelCust` können nicht nur Levels aus Hierarchien der Faktklasse, sondern auch Levels aus Hierarchien beinhalten, die dieser über die Modellanpassung hinzugefügt werden. Analoges gilt für die Funktionen `insertDeselectedAttributeToModelCust` beziehungsweise `deleteDeselectedAttribute-FromModelCust.`

Kennzahlen können über eine Modellanpassung nicht nur aus einem multidimensionalen Modell abgewählt, sondern diesem auch hinzugefügt werden. Die XQuery-Funktionen für die Verwaltung hinzugefügter Kennzahlen in Modellanpassungen sind in Tabelle 11 aufgelistet. Die Funktion `insertAddedNewMeasureToModelCust` ermöglicht es, der von der Modellanpassung referenzierten Faktklasse eine Basiskennzahl hinzuzufügen, die nicht im entsprechenden Referenzkatalog für Kennzahlen enthalten ist. Die Kennzahl wird dabei direkt in der Modellanpassung über einen Namen und eine Beschreibung definiert. Mit der Funktion `deleteAddedNewMeasureToModelCust` kann eine so definierte Basiskennzahl wieder aus einer Modellanpassung entfernt werden.

Funktion	Parameter	Beschreibung
insertAddedNewMeasure ToModelCust	$database xs:string $modelCustomization xs:string $name xs:string $description xs:string	Fügt der Liste der direkt erstellten Basiskennzahlen der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) eine Kennzahl mit dem angegebenen Namen ($name) und der angegebenen Beschreibung ($description) hinzu.
deleteAddedNewMeasure FromModelCust	$database xs:string $modelCustomization xs:string $measure xs:string	Löscht die angegebene Kennzahl ($measure) aus der Liste der direkt erstellten Basiskennzahlen der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).
insertAddedCatalogueMeasure ToModelCust	$database xs:string $modelCustomization xs:string $measure xs:string	Fügt der Liste der hinzugefügten Kennzahlen aus dem Kennzahlkatalog der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Kennzahl ($measure) hinzu.
deleteAddedCatalogueMeasure FromModelCust	$database xs:string $modelCustomization xs:string $measure xs:string	Löscht die angegebene Kennzahl ($measure) aus der Liste der hinzugefügten Kennzahlen aus dem Kennzahlkatalog der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).

Tabelle 11 Funktionen zur Verwaltung von hinzugefügten Kennzahlen in Modellanpassungen

Funktion	Parameter	Beschreibung
insertAddedNewHierarchy ToModelCust	$database xs:string $modelCustomization xs:string $name xs:string $description xs:string	Fügt der Liste der direkt erstellten Hierarchien der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) eine Hierarchie mit dem angegebenen Namen ($name) und der angegebenen Beschreibung ($description) hinzu.
deleteAddedNewHierarchy FromModelCust	$database xs:string $modelCustomization xs:string $hierarchy xs:string	Löscht die angegebene Hierarchie ($hierarchy) aus der Liste der direkt erstellten Hierarchien der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).
insertAddedNewLevel ToModelCust	$database xs:string $modelCustomization xs:string $name xs:string $description xs:string $sampleValues xs:string	Fügt der Hierarchie des angegebenen Levels ($name) in der Liste der direkt erstellten Hierarchien der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) einen Level mit dem angegebenen Namen ($name), der angegebenen Beschreibung ($description) und den angegebenen Beispielwerten ($sampleValues) hinzu.
insertAddedNewSuperLevel ToModelCust	$database xs:string $modelCustomization xs:string $level xs:string $superLevel xs:string	Fügt dem angegebenen Level ($level) in der entsprechenden Hierarchie in der Liste der direkt erstellten Hierarchien der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) den angegebenen Superlevel ($superLevel) hinzu.
insertAddedNewAttribute ToModelCust	$database xs:string $modelCustomization xs:string $level xs:string $name xs:string $description xs:string $sampleValues xs:string	Fügt dem angegebenen Level ($level) in der entsprechenden Hierarchie in der Liste der direkt erstellten Hierarchien der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) ein Attribut mit dem angegebenen Namen ($name), der angegebenen Beschreibung ($description) und den angegebenen Beispielwerten ($sampleValues) hinzu.
insertAddedCatalogueHierarchy ToModelCust	$database xs:string $modelCustomization xs:string $hierarchy xs:string	Fügt der Liste der hinzugefügten Hierarchien aus dem Hierarchie-Katalog der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Hierarchie ($hierarchy) hinzu.
deleteAddedCatalogueHierarchy FromModelCust	$database xs:string $modelCustomization xs:string $hierarchy xs:string	Löscht die angegebene Hierarchie ($hierarchy) aus der Liste der hinzugefügten Hierarchien aus dem Hierarchie-Katalog der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).

Tabelle 12: Funktionen zur Verwaltung von hinzugefügten Hierarchien in Modellanpassungen

Berechnete Kennzahlen können nicht direkt in einer Modellanpassung definiert werden. Um einer Faktklasse eine berechnete Kennzahl hinzuzufügen, ist die Funktion **`insertCatalogueMeasure-`**

ToModelCust zu verwenden. Über die Funktion **deleteCatalogueMeasureFromModelCust** kann die Kennzahl wiederum aus der Liste der hinzugefügten Kennzahlen der Modellanpassung entfernt werden. Diese beiden Funktionen können auch für Basiskennzahlen aus dem Referenzkatalog für Kennzahlen verwendet werden.

Auch Hierarchien können einem multidimensionalen Modell über eine Modellanpassung hinzugefügt werden. Die für die Verwaltung hinzugefügter Hierarchien verfügbaren XQuery-Funktionen sind in Tabelle 12 angeführt. Das Hinzufügen einzelner Levels und Attribute zu bestehenden Hierarchien im multidimensionalen Modell oder im Referenzkatalog für Hierarchien ist aus Gründen der Konsistenzsicherung nicht vorgesehen.

Wie Basiskennzahlen können auch Hierarchien, die nicht im Referenzkatalog für Hierarchien enthalten sind, direkt in einer Modellanpassung definiert und so der entsprechenden Faktklasse zugeordnet werden. Mit den Funktionen **insertAddedNewHierarchyToModelCust** kann eine neue Hierarchie hinzugefügt, mit der Funktion **deleteAddedNewHierarchyFromModelCust** wieder aus der Liste der hinzugefügten Hierarchien der Modellanpassung entfernt werden.

Einer über die Funktion **insertAddedNewHierarchyToModelCust** definierten Hierarchie können über die Funktion **insertAddedNewLevelToModelCust** Levels zugeteilt werden. Diesen können über die Funktionen **insertAddedNewSuperLevelToModelCust** und **insertAddedNewAttributeToModelCust** Superlevels und Attribute zugeordnet werden. Das Löschen einzelner Levels, Superlevels oder Attribute aus direkt in der Modellanpassung definierten Hierarchien ist nicht vorgesehen. Um die Elemente einer solchen Hierarchie zu verändern, ist diese über die Funktion **deleteAddedNewHierarchyFromModelCust** zu entfernen und anschließend neu anzulegen.

Funktion	Parameter	Beschreibung
insertMeasureRedefinition ToModelCust	$database xs:string $modelCustomization xs:string $name xs:string $calculationRule xs:string $measure xs:string	Fügt der Liste der Kennzahl-Neudefinitionen der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) eine Neudefinition mit dem angegebenen Namen ($name) und der angegebenen Berechnungsregel ($calculationRule) hinzu, die die angegebene Kennzahl ($measure) neu definiert.
deleteMeasureRedefinition FromModelCust	$database xs:string $modelCustomization xs:string $measure xs:string	Löscht die Neudefinition der angegebenen Kennzahl ($measure) aus der Liste der Kennzahl-Neudefinitionen der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).
insertPredicateRedefinition ToModelCust	$database xs:string $modelCustomization xs:string $name xs:string $expression xs:string $predicate xs:string	Fügt der Liste der Prädikat-Neudefinitionen der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database) eine Neudefinition mit dem angegebenen Namen ($name) und Ausdruck ($expression) hinzu, die das angegebene Prädikat ($predicate) neu definiert.
deletePredicateRedefinition FromModelCust	$database xs:string $modelCustomization xs:string $predicate xs:string	Löscht die Neudefinition des angegebenen Prädikats ($predicate) aus der Liste der Prädikat-Neudefinitionen der angegebenen Modellanpassung ($modelCustomization) in der angegebenen BaseX-Datenbank ($database).

Tabelle 13: Funktionen zur Verwaltung von Neudefinitionen in Modellanpassungen

Über die Funktionen **insertAddedCatalogueHierarchyToModelCust** und **deleteAddedCatalogue-HierarchyFromModelCust** können Hierarchien aus dem Referenzkatalog für Hierarchien der Liste der hinzugefügten Hierarchien einer Modellanpassung und damit der entsprechenden Faktklasse zugeordnet beziehungsweise aus dieser entfernt werden.

```
01 <modelCustomization Name="ModelCustomization1">
02   <factClass Name="Invoice"/>
03   <deselectedMeasures>
04     <measure Name="commercialDiscount"/>
05   </deselectedMeasures>
06   <deselectedHierarchies>
07     <hierarchy Name="Agent"/>
08   </deselectedHierarchies>
09   <deselectedLevels>
10     <level Name="county" Hierarchy="Customer"/>
11     <level Name="state" Hierarchy="Customer"/>
12   </deselectedLevels>
13   <deselectedAttributes>
14     <attribute Name="ageRange" Hierarchy="Product"/>
15   </deselectedAttributes>
16   <addedMeasures>
17     <newMeasures>
18       <measure Id="c3105e74-6c37-4987-96d0-acbf97553deb" Name="netAveragePrice"
  Description=""/>
19     </newMeasures>
20     <catalogueMeasures>
21       <measure Name="vat"/>
22       <measure Name="grossValue"/>
23       <measure Name="annualGrowth"/>
24     </catalogueMeasures>
25   </addedMeasures>
26   <addedHierarchies>
27     <newHierarchies>
28       <hierarchy Id="68258a6f-2b80-4179-879f-83f7638aaa01" RootArcId=
  "99fca426-b81c-4238-b211-4461971fa160" Name="ProfitCenter" Description="">
29         <levels>
30           <level Id="86deda95-2f86-4b6d-ad0e-9e7402b3ba35" Name="profitCenter"
  Description="" SampleValues="">
31             <superLevels>
32               <level Name="location"/>
33             </superLevels>
34             <attributes/>
35           </level>
             ...
48         </levels>
49       </hierarchy>
50     </newHierarchies>
51     <catalogueHierarchies>
52       <hierarchy Name="ReturnJust"/>
53     </catalogueHierarchies>
54   </addedHierarchies>
55   <measureRedefinitions>
56     <redefinition Name="growthInPercent" Measure="annualGrowth" CalculationRule=
  "Fact.quantity / ComparisonFact.quantity * 100"/>
57   </measureRedefinitions>
58   <predicateRedefinitions>
59     <redefinition Name="exclusiveBeverages2015" Predicate="premiumBeverages2015"
  Expression="Product.type = 'beverage' AND Time.years = '2015' AND Fact.netValue = 100"/>
60   </predicateRedefinitions>
61 </modelCustomization>
```

Liste 8: Anpassung eines multidimensionalen Referenzmodells

Neben dem Hinzufügen und Abwählen von Elementen des multidimensionalen Modells ermöglichen Modellanpassungen auch die Neudefinition von berechneten Kennzahlen und Prädikaten. Die XQuery-Funktionen für die Verwaltung solcher Neudefinitionen werden in Tabelle 13 erläutert. Über die Funktion **insertMeasureRedefinitionToModelCust** kann einer Modellanpassung eine Kennzahl-Neudefinition hinzugefügt werden. Dabei sind neben dem Namen der Neudefinition auch

der Name der neudefinierten Kennzahl und die neue Berechnungsregel anzugeben. Bestehende Neudefinitionen können über die Funktion `deleteMeasureRedefinitionFromModelCust` entfernt werden, wobei die Neudefinition über den Namen der neu definierten Kennzahl referenziert wird. Jede Modellanpassung kann demnach nur eine Neudefinition für jede Kennzahl enthalten.

Neudefinitionen für Prädikate können über die Funktionen `insertPredicateRedefinitionToModelCust` und `deletePredicateRedefinitionFromModelCust` einer Modellanpassung hinzugefügt beziehungsweise aus dieser entfernt werden. Beim Hinzufügen sind dabei neben dem Namen der Neudefinition auch der Name des Prädikats und der neu definierte Prädikatausdruck anzugeben. Beim Entfernen wird die Neudefinition über den Namen des neu definierten Prädikats referenziert.

Bei neudefinierten, aber auch bei bestehenden Kennzahlen und Prädikaten ist darauf zu achten, dass alle benötigten Elemente des multidimensionalen Modells vorhanden sind. Dabei sind auch die über die Modellanpassung entfernten und hinzugefügten Elemente zu berücksichtigen.

Liste 8 zeigt eine beispielhafte Modellanpassung namens `ModelCustomization1`, die sich auf die Faktklasse `Invoice` aus dem verwendeten Beispielprojekt bezieht. Die Modellanpassung wählt die Kennzahl `commercialDiscount`, die Hierarchie `Agent`, die Levels `county` und `state` der Hierarchie `Customer` und das Attribut `ageRange` der Hierarchie `Product` ab.

In der Modellanpassung wird die Basiskennzahl `netAveragePrice` (durchschnittlicher Nettopreis) neu definiert und somit der Faktklasse `Invoice` zugeordnet. Zusätzlich dazu werden die Basiskennzahl `vat`, die kumulative Kennzahl `averageValue` und die Vergleichskennzahl `annualGrowth` aus dem Referenzkatalog für Kennzahlen hinzugefügt. Für die Kennzahl `annualGrowth` ist außerdem, wie auch für das Prädikat `premiumBeverages2015`, eine Neudefinition in der Modellanpassung erfasst.

Weiters definiert die Modellanpassung eine neue Hierarchie namens `ProfitCenter`, deren Elemente aus Gründen der Übersichtlichkeit in Liste 8 jedoch nicht zur Gänze abgebildet sind. Lediglich der Basislevel `profitCenter`, welcher auf den Superlevel `location` verweist, ist dargestellt. Zusätzlich zur Hierarchie `ProfitCenter`, fügt die gezeigte Modellanpassung der Faktklasse die Hierarchie `ReturnJust` aus dem Referenzkatalog für Hierarchien hinzu.

3.2.8 Anpassungen für Analysesituationen

Wie bereits erwähnt, sieht BIRD nicht nur die Anpassung von multidimensionalen Referenzmodellen, sondern auch die Anpassung von Analysesituationen und Analysegraphen vor. In der vorliegenden Implementierung können für jede BaseX-Datenbank beliebig viele Anpassungen für Analysesituationen erstellt werden. Die XQuery-Funktionen zum Erstellen und Löschen solcher Situationsanpassungen zeigt Tabelle 14, die Funktionen für die Verwaltung der Elemente der Situationsanpassungen sind in Tabelle 15 bis Tabelle 17 zu finden.

Funktion	Parameter	Beschreibung
createSituationCustomization	$database xs:string $name xs:string $situation xs:string	Erzeugt in der angegebenen BaseX-Datenbank ($database) eine leere Anpassung für Analysesituationen mit dem angegebenen Namen ($name), die sich auf die angegebene Analysesituation ($situation) bezieht.
deleteSituationCustomization	$database xs:string $situationCustomization xs:string	Löscht die angegebene Anpassung für Analysesituationen ($situationCustomization) aus der angegebenen BaseX-Datenbank ($database).

Tabelle 14: Funktionen zur Verwaltung von Anpassungen für Analysesituationen

Die Funktion `createSituationCustomization` ermöglicht das Erstellen einer leeren Situationsanpassung in der angegebenen BaseX-Datenbank, wobei neben dem Namen der

Situationsanpassung auch der Name der Analysesituation im Referenzkatalog für Analysesituationen anzugeben ist. Bestehende Situationsanpassungen können mit der Funktion `deleteSituationCustomization` aus der BaseX-Datenbank gelöscht werden.

Über eine Situationsanpassung können Prädikate und Kennzahlen aus einer Analysesituation im Referenzkatalog abgewählt werden. Die dafür umgesetzten XQuery-Funktionen zeigt Tabelle 15. Die Funktion `insertDeselectedPredicateToSituationCust` fügt der Liste der abgewählten Prädikate in der Situationsanpassung ein multidimensionales Prädikat oder Kennzahlprädikat hinzu. Über die Funktion `deleteDeselectedPredicateFromSituationCust` können die Prädikate wieder aus der Liste der abgewählten Prädikate entfernt werden. Die Funktionen `insertDeselectedMeasureToSituationCust` und `deleteDeselectedMeasureFromSituationCust` erfüllen die selben Aufgaben analog für Kennzahlen.

Funktion	Parameter	Beschreibung
insertDeselectedPredicatToSituationCust	$database xs:string $situationCustomization xs:string $predicate xs:string	Fügt der Liste der abgewählten Prädikate der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database) das angegebene Prädikat ($predicate) hinzu.
deleteDeselectedPredicateFromSituationCust	$database xs:string $situationCustomization xs:string $predicate xs:string	Löscht das angegebene Prädikat ($predicate) aus der Liste der abgewählten Prädikate der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database).
insertDeselectedMeasureToSituationCust	$database xs:string $situationCustomization xs:string $measure xs:string	Fügt der Liste der abgewählten Kennzahlen der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Kennzahl ($measure) hinzu.
deleteDeselectedMeasureFromSituationCust	$database xs:string $situationCustomization xs:string $measure xs:string	Löscht die angegebene Kennzahl ($measure) aus der Liste der abgewählten Kennzahlen der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database).

Tabelle 15: Funktionen zur Verwaltung von abgewählten Elementen in Situationsanpassungen

Prädikate und Kennzahlen können über eine Situationsanpassung nicht nur aus einer Analysesituation entfernt, sondern dieser auch hinzugefügt werden. Dabei ist jedoch zu beachten, dass der Analysesituation nur solche Kennzahlen hinzugefügt werden können, die in der Faktklasse, auf die sich die Situation bezieht, vorhanden sind. Die XQuery-Funktionen für das Hinzufügen von Prädikaten und Kennzahlen werden in Tabelle 16 beschrieben.

Die Funktion `insertAddedPredicateToSituationCust` fügt ein multidimensionales Prädikat oder Kennzahlprädikat in die Liste der hinzugefügten Prädikate der Situationsanpassung ein. Dimensionsprädikate können der Analysesituation nicht direkt, sondern nur als Slice-Bedingung einer Dimension zugeordnet werden. Der Aufbau von Analysesituationen wurde bereits in Abschnitt 2.1.3 diskutiert. Hinzugefügte Situationen können über die Funktion `deleteAddedPredicate-FromSituationCust` wieder entfernt werden.

Über die Funktion `insertAddedMeasureToSituationCust` können der Liste der hinzugefügten Kennzahlen der Situationsanpassung wie oben erwähnt Kennzahlen der Faktklasse hinzugefügt werden, die von der Situationsanpassung referenziert wird. Die Funktion `deleteAddedMeasureFromSituationCust` entfernt auf diese Art hinzugefügte Kennzahlen.

Funktion	Parameter	Beschreibung
insertAddedPredicate ToSituationCust	$database xs:string $situationCustomization xs:string $predicate xs:string	Fügt der Liste der hinzugefügten Prädikate der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database) das angegebene Prädikat ($predicate) hinzu.
deleteAddedPredicate FromSituationCust	$database xs:string $situationCustomization xs:string $predicate xs:string	Löscht das angegebene Prädikat ($predicate) aus der Liste der hinzugefügten Prädikate der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database).
insertAddedMeasure ToSituationCust	$database xs:string $situationCustomization xs:string $measure xs:string	Fügt der Liste der hinzugefügten Kennzahlen der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Kennzahl ($measure) hinzu.
deleteAddedMeasure FromSituationCust	$database xs:string $situationCustomization xs:string $measure xs:string	Löscht die angegebene Kennzahl ($measure) aus der Liste der hinzugefügten Kennzahlen der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database).
insertAggregation ToSituationCust	$database xs:string $situationCustomization xs:string $measure xs:string $aggregationOperation xs:string	Fügt der Liste der Aggregationsoperationen der angegebenen hinzugefügten Kennzahl ($measure) in der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Aggregationsoperation ($aggregationOperation) hinzu.

Tabelle 16 Funktionen zur Verwaltung von hinzugefügten Elementen in Situationsanpassungen

Funktion	Parameter	Beschreibung
insertDimensionRedefinition ToSituationCust	$database xs:string $situationCustomization xs:string $dimension xs:string $granularityLevel xs:string $diceLevel $diceNode	Fügt der Liste der Dimensions-Neudefinitionen der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database) eine Neudefinition mit dem angegebenen Granularitätslevel ($granularityLevel), dem angegebenen Dice-Level ($diceLevel) und dem angegebenen Dice-Knoten ($diceNode) hinzu, die die angegebene Dimension ($dimension) neu definiert.
deleteDimensionRedefinition FromSituationCust	$database xs:string $situationCustomization xs:string $dimension xs:string	Löscht die Neudefinition der angegebenen Dimension ($dimension) aus der Liste der Dimensions-Neudefinitionen der angegebenen Situationsanpassung ($situationCustomization) in der angegebenen BaseX-Datenbank ($database).

Tabelle 17: Funktionen zur Verwaltung von Neudefinitionen in Situationsanpassungen

Wurde einer Situationsanpassung eine Kennzahl über die Funktion **`insertAddedMeasure-ToSitutuationCust`** hinzugefügt, können dieser über die Funktion **`insertAggregationToSituationCust`** optional eine oder mehrere Aggregationsoperationen hinzugefügt werden. Das Entfernen einzelner Aggregationsoperationen ist dabei nicht vorgesehen. Sollen die Aggregationsoperationen einer

Kennzahl verändert werden, ist die Kennzahl über die Funktion `deleteAddedMeasureToSituationCust` zu entfernen und über die Funktion `insertAddedMeasureTo-SituatuionCust` neu hinzuzufügen.

Wurde für eine Dimension der Granularitätslevel, der Dice-Level oder der Dice-Knoten im Referenzkatalog für Analysesituationen in Form einer Variable angegeben, kann dieser über eine Neudefinition in der Situationsanpassung verändert werden. Die für die Verwaltung solcher Neudefinitionen verfügbaren XQuery-Funktionen sind in Tabelle 17 abgebildet.

Die Funktion `insertDimensionRedefinitionToSituationCust` ermöglicht das Erstellen einer Neudefinition. Dabei sind neben dem Namen der Neudefinition und dem Namen der Dimension der Granularitätslevel, der Dice-Level und der Dice-Knoten anzugeben. Über die Funktion `deleteDimensionRedefinitionFromSituationCust` kann eine so erstellte Neudefinition wieder entfernt werden, wobei die Neudefinition über den Namen der neu definierten Dimension referenziert wird. Je Dimension kann eine Situationsanpassung maximal eine Neudefinition enthalten.

Liste 9 zeigt die Struktur einer beispielhaften Anpassung der Analysesituation `Situation2` aus dem Referenzkatalog für Analysesituationen namens `SituationCustomization1`. Die Situationsanpassung wählt die Kennzahl `netValue` der Analysesituation ab und ergänzt die Kennzahl `quantity` aus der Faktklasse `Invoice`, auf die sich die referenzierte Analysesituation bezieht. Dieser Kennzahl wird die Aggregationsoperation `SUM` (Summe) zugewiesen. Weiters definiert die Situationsanpassung die Dimension `Product` neu, indem sie die im Referenzkatalog für Analysesituationen hinterlegten Variablen durch konkrete Werte ersetzt. Prädikate werden weder entfernt noch hinzugefügt.

```
01 <analysisSituationCustomization Name="SituationCustomization1">
02   <analysisSituation Name="Situation1"/>
03   <deselectedMeasures>
04     <measure Name="netValue"/>
05   </deselectedMeasures>
06   <deselectedPredicates/>
07   <addedMeasures>
08     <measure Name="quantity">
09       <aggregationOperations>
10         <aggregationOperation Name="AVG"/>
11       </aggregationOperations>
12     </measure>
13   </addedMeasures>
14   <addedPredicates/>
15   <dimensionRedefinitions>
16     <redefinition Name="typeGranularity" Dimension="Product" GranularityLevel="type"
  DiceLevel="brandGroup" DiceNode="austrianBrands"/>
17   </dimensionRedefinitions>
18 </analysisSituationCustomization>
```

Liste 9: Anpassung einer Analysesituation

3.2.9 Anpassungen für Analysegraphen

Die letzte in BIRD vorgesehene Art der Anpassung ist die Anpassung von Analysegraphen. In der vorliegenden Implementierung kann in jeder BaseX-Datenbank eine beliebige Anzahl solcher Graphanpassungen angelegt werden. Die für das Erstellen und Löschen der Graphanpassungen vorgesehenen XQuery-Funktionen werden in Tabelle 18, die für die Verwaltung der Elemente der Graphanpassung vorgesehenen in den darauffolgenden Tabellen, beschrieben.

Mit der Funktion `createGraphCustomization` kann eine Graphanpassung für einen bestimmten Analysegraphen in einer angegebenen BaseX-Datenbank erstellt werden. Die Funktion `deleteGraphCustomization` dient dem Entfernen bestehender Graphanpassungen. Der beim Erstellen der Graphanpassung angegebene Analysegraph muss dabei im Referenzkatalog für Analysegraphen der BaseX-Datenbank enthalten sein.

Funktion	Parameter	Beschreibung
createGraphCustomization	$database xs:string $name xs:string $graph xs:string	Erzeugt in der angegebenen BaseX-Datenbank ($database) eine leere Anpassung für Analysegraphen mit dem angegebenen Namen ($name), die sich auf den angegebene Analysegraphen ($graph) bezieht.
deleteGraphCustomization	$database xs:string $graphCustomization xs:string	Löscht die angegebene Anpassung für Analysegraphen ($graphCustomization) aus der angegebenen BaseX-Datenbank ($database).

Tabelle 18: Funktionen zur Verwaltung von Anpassungen für Analysegraphen

Tabelle 19 zeigt die XQuery-Funktionen für die Verwaltung der über die Graphanpassung abgewählten Analysesituationen und Navigationsschritte des referenzierten Analysegraphen. Mit der Funktion `insertRemovedSituationToGraphCust` kann eine Analysesituation der Liste der abgewählten Analysesituationen in der Graphanpassung hinzugefügt werden. Die Funktion `deleteRemoved-SituationFromGraphCust` ermöglicht das Entfernen auf diese Weise hinzugefügter Analysesituationen aus der Liste der abgewählten Analysesituationen. Die Funktionen `insertRemovedStepToGraphCust` und `deleteRemovedStepFromGraphCust` bieten die selbe Funktionalität wie die beiden letztgenannten in Bezug auf das Hinzufügen und Entfernen abgewählter Navigationsschritte.

Funktion	Parameter	Beschreibung
insertRemovedSituation ToGraphCust	$database xs:string $graphCustomization xs:string $situation xs:string	Fügt der Liste der entfernten Analysesituationen der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Analysesituation ($situation) hinzu.
deleteRemovedSituation FromGraphCust	$database xs:string $graphCustomization xs:string $situation xs:string	Löscht die angegebene Analysesituation ($situation) aus der Liste der entfernten Analysesituationen der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database).
insertRemovedStep ToGraphCust	$database xs:string $graphCustomization xs:string $navigationStep xs:string	Fügt der Liste der entfernten Navigationsschritte der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database) den angegebenen Navigationsschritt ($navigationStep) hinzu.
deleteRemovedStep FromGraphCust	$database xs:string $graphCustomization xs:string $navigationStep xs:string	Löscht den angegebenen Navigationsschritt ($navigationStep) aus der Liste der entfernten Navigationsschritte der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database).

Tabelle 19: Funktionen zur Verwaltung von abgewählten Elementen in Graphanpassungen

Neben dem Abwählen von Analysesituationen und Navigationsschritten aus Analysegraphen ermöglichen Graphanpassungen auch das Hinzufügen dieser. Die dafür umgesetzten XQuery-Funktionen zeigt Tabelle 20.

Während über die Funktion `insertAddedSituationToGraphCust` der Liste der über die Graphanpassung hinzugefügten Analysesituationen eine Analysesituation hinzugefügt werden kann, ermöglicht die Funktion `deleteAddedSituationFromGraphCust` das Entfernen dieser hinzugefügten Analysesituationen. Dabei ist zu beachten, dass über eine Graphanpassung nur Analysesituationen

hinzugefügt werden können, die im Referenzkatalog für Analysesituationen der entsprechenden BaseX-Datenbank enthalten sind.

Wird über die Funktion `insertAddedStepToGraphCust` der Liste der hinzugefügten Navigationsschritte der Graphanpassung ein Navigationsschritt hinzugefügt, sind neben dem Namen des Navigationsschrittes auch die Ausgangs- und Zielanalysesituation anzugeben. Die der Graphanpassung über die Funktion `insertAddedStepToGraphCust` hinzugefügten Navigationsschritte können über die Funktion `deletAddedStepFromGraphCust` wieder entfernt werden.

Wurde ein Navigationsschritt über die Funktion `insertAddedStepToGraphCust` der Liste der hinzugefügten Navigationsschritte einer Graphanpassung hinzugefügt, können diesem über die Funktion `insertAddedOperationToGraphCust` Navigationsoperationen hinzugefügt werden. Tabelle 1 in Abschnitt 2.1.5 listet die in der vorliegenden Implementierung umgesetzten möglichen Operationen auf.

Das Löschen einzelner Navigationsoperationen eines über die Graphanpassung hinzugefügten Navigationsschrittes ist nicht vorgesehen. Sollen die Operationen eines Navigationsschrittes verändert werden, muss der gesamte Navigationsschritt aus der Graphanpassung entfernt und neu hinzugefügt werden.

Funktion	Parameter	Beschreibung
insertAddedSituation ToGraphCust	$database xs:string $graphCustomization xs:string $situation xs:string	Fügt der Liste der hinzugefügten Analysesituationen der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Analysesituation ($situation) hinzu.
deleteAddedSituation FromGraphCust	$database xs:string $graphCustomization xs:string $situation xs:string	Löscht die angegebene Analysesituation ($situation) aus der Liste der hinzugefügten Analysesituationen der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database).
insertAddedStep ToGraphCust	$database xs:string $graphCustomization xs:string $navigationStep xs:string $source xs:string $target xs:string	Fügt der Liste der hinzugefügten Navigationsschritte der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database) den angegebenen Navigationsschritt ($navigationStep) mit der angegebenen Ausgangsanalysesituation ($source) und Zielanalysesituation ($target) hinzu.
deleteAddedStep FromGraphCust	$database xs:string $graphCustomization xs:string $navigationStep xs:string	Löscht den angegebenen Navigationsschritt ($navigationStep) aus der Liste der hinzugefügten Navigationsschritte der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database).
insertAddedOperation ToGraphCust	$database xs:string $graphCustomization xs:string $navigationStep xs:string $navigationOperation xs:string	Fügt der Liste der Navigationsoperationen des angegebenen hinzugefügten Navigationsschritts ($navigationStep) in der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Navigationsoperation ($navigationOperation) hinzu.

Tabelle 20 Funktionen zur Verwaltung von hinzugefügten Elementen in Graphanpassungen

Neben dem Hinzufügen neuer Navigationsschritte ermöglichen Graphanpassungen auch die Neudefinition bestehender Navigationsschritte. Tabelle 21 zeigt die für die Verwaltung solcher Neudefinitionen in Graphanpassungen vorgesehenen XQuery-Funktionen.

Mit der Funktion `insertStepRedefinitionToGraphCust` kann eine Neudefinition in einer Graphanpassung angelegt werden. Dabei ist neben dem Namen der Neudefinition auch der des neu definierten Navigationsschrittes anzugeben. Die Funktion `deleteStepRedefinitionFromGraphCust` dient

dem Entfernen bestehender Neudefinitionen. Die zu entfernende Neudefinition ist dabei über den Namen des neu definierten Navigationsschrittes zu referenzieren.

Funktion	Parameter	Beschreibung
insertStepRedefinition ToGraphCust	$database xs:string $graphCustomization xs:string $navigationStep xs:string	Fügt der Liste der Navigationsschritt-Neudefinitionen der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database) eine Neudefinition des angegebenen Navigationsschritts ($navigationStep) hinzu.
deleteStepRedefinition FromGraphCust	$database xs:string $graphCustomization xs:string $navigationStep xs:string	Löscht die Neudefinition des angegebenen Navigationsschritts ($navigationStep) aus der Liste der Navigationsschritt-Neudefinitionen der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database).
insertRedefinedOperation ToGraphCust	$database xs:string $graphCustomization xs:string $navigationStep xs:string $navigationOperation xs:string	Fügt der Liste der Navigationsoperationen des angegebenen neudefinierten Navigationsschritts ($navigationStep) in der angegebenen Graphanpassung ($graphCustomization) in der angegebenen BaseX-Datenbank ($database) die angegebene Navigationsoperation ($navigationOperation) hinzu.

Tabelle 21: Funktionen zur Verwaltung von Neudefinitionen in Graphanpassungen

Nachdem über die Funktion `insertStepRedefinitionToGraphCust` eine Neudefinition angelegt wurde, können dieser über die Funktion `insertRedefinedOperationToGraphCust` die neu definierten Navigationsoperationen hinzugefügt werden. Diese ergänzen dabei nicht die bestehenden Navigationsoperationen des neu definierten Navigationsschrittes, sondern ersetzen diese.

```
01 <analysisGraphCustomization Name="GraphCustomization1">
02   <analysisGraph Name="Graph1"/>
03   <addedSteps>
04     <navigationStep Name="AddedStep1" Source="Situation2" Target="Situation3">
05       <operations>
06         <operation Name="addPredicate(beveragesNovember2015)"/>
07       </operations>
08     </navigationStep>
09   </addedSteps>
10   <removedSteps/>
11   <addedSituations>
12     <analysisSituation Name="Situation3"/>
13   </addedSituations>
14   <removedSituations/>
15   <stepRedefinitions>
16     <redefinition Name="redefinition1" NavigationStep="Step1">
17       <operations>
18         <operation Name="removeMeasure(quantity)"/>
19         <operation Name="addMeasure(netValue,AVG)"/>
20         <operation Name="addMeasure(?m2,SUM)"/>
21         <operation Name="addPredicate(?p2)"/>
22         <operation Name="changeGranularity(Product,?l1)"/>
23         <operation Name="moveToNode(Product,?l2,?n1)"/>
24         <operation Name="refocusSliceCondition(Product,groceries,?p3)"/>
25       </operations>
26     </redefinition>
27   </stepRedefinitions>
28 </analysisGraphCustomization>
```

Liste 10: Anpassung eines Analysegraphen

Liste 10 zeigt die Struktur einer beispielhaften Graphanpassung, die sich auf den Analysegraph `Graph1` aus dem in Liste 7 gezeigten Referenzkatalog für Analysegraphen bezieht. Die Graphanpassung ergänzt einen Navigationsschritt von Analysesituation `Situation2` zu Analysesituation `Situation3`, dessen Navigationsoperation `addPredicate(beveragesNovember2015)` das multidimensionale Prädikat `beveragesNovember2015` aus dem in Liste 5 dargestellten Referenzkatalog für Prädikate hinzufügt. Da die Analysesituation `Situation3` nicht im Analysegraph `Graph1` im Referenzkatalog für Analysegraphen enthalten ist, wird diese ebenfalls über die Graphanpassung ergänzt, um die Konsistenz von Analysesituationen und Navigationsschritten zu erhalten.

Daraus, dass die Navigationsoperationen des Navigationsschritts `Step1`, der von Analysesituation `Situation1` zu Analysesituation `Situation2` führt, neu definiert werden, kann man schließen, dass eine dieser beiden über eine Situationsanpassung verändert wurde, da sie ansonsten nicht mehr mit dem Navigationsschritt übereinstimmen würden. Im konkreten Fall bezieht sich die Graphanpassung auf die Analysesituation `Situation1` unter der Anwendung der in Liste 9 gezeigten Situationsanpassung.

3.2.10 Anpassungsanwendung und ausführbare Analysegraphen

Wurden multidimensionale Referenzmodelle, Referenzkataloge und Anpassungen wie in den letzten Abschnitten erläutert angelegt, können über die in diesem Abschnitt beschriebenen XQuery-Funktionen die Anpassungen für multidimensionale Referenzmodelle, Analysesituationen und Analysegraphen angewendet und darauf basierend ausführbare Analysegraphen generiert werden. Die ersten Funktionen für die Anwendung der Anpassungen sind in Tabelle 22 abgebildet.

Die Funktion `parseCatalogues` greift auf die Referenzkataloge für Kennzahlen und Prädikate der angegebenen BaseX-Datenbank zu. Sie analysiert die Berechnungsregeln und Verbundbedingungen der berechneten Kennzahlen im Referenzkatalog für Kennzahlen sowie die Ausdrücke der Prädikate im Referenzkatalog für Prädikate und ergänzt darauf basierend die für die Kennzahlen respektive Prädikate benötigten Modellelemente in den entsprechenden Subelementen in den Referenzkatalogen. Wie in Abschnitt 3.2.3 und Abschnitt 3.2.4 erläutert, muss die Funktion `parseCatalogues` angewendet werden, um die Verwendung der in diesen Abschnitten beschriebenen Funktionen `markMandatoryOverMeasureInMeasureCatalogue`, `markMandatoryOverAttributeInMeasure-Catalogue`, `markMandatoryMeasureInPredicateCatalogue` und `markMandatoryAttributeInPredicate-Catalogue` zu ermöglichen.

Auch für die Ausführung der in Tabelle 22 beschriebenen Funktion `createFiles` ist es notwendig, zuvor die Funktion `parseCatalogues` auszuführen. Der Funktion `createFiles` sind die Namen der gewünschten Anpassungen für multidimensionale Referenzmodelle und Analysesituationen zu übergeben. Dabei darf für jedes multidimensionale Referenzmodell und für jede Analysesituation nur je eine Anpassung angegeben werden. Zusätzlich dazu ist der Name der gewünschten Anpassung für den Analysegraphen anzugeben. Soll ein Analysegraph ohne Anpassung verwendet werden, ist stattdessen der Name des gewünschten Analysegraphen anzugeben.

Wurden über eine der bei der Funktion `createFiles` angegebenen Anpassungen für multidimensionale Referenzmodelle Neudefinitionen für Kennzahlen oder Prädikate erstellt, ist die Funktion `parseCatalogues` nach der Ausführung der Funktion `createFiles` erneut auszuführen. Erst im Anschluss kann die Funktion `createProject` angewendet werden. Wie die Funktion `createFiles` dient sie dem Erstellen von Dateien in der angegebenen BaseX-Datenbank, die für die Erstellung eines Indyco-Projekts basierend auf den gewählten Anpassungen und die spätere Erstellung eines ausführbaren Analysegraphen benötigt werden.

Nachdem die in Tabelle 22 beschriebenen XQuery-Funktionen nacheinander angewendet wurden, kann die BaseX-Datenbank über die in Tabelle 2 beschrieben Funktion `exportDB` in ein gewünschtes Verzeichnis exportiert werden. Das Verzeichnis `indycoProject` innerhalb des gewählten Verzeichnisses enthält daraufhin ein Indyco-Projekt, welches in Indyco Builder geöffnet werden kann.

Funktion	Parameter	Beschreibung
parseCatalogues	$database xs:string	Ergänzt in den Kennzahl- und Prädikatkatalogen der angegebenen BaseX-Datenbank ($database) die für die Verwendung der Kennzahlen und Prädikate notwendigen Elemente.
createFiles	$datatbase xs:string $graphCustomization xs:string $modelCustomizations xs:string $situationCustomizations xs:string	Erzeugt auf Basis der angegebenen Anpassungen für multidimensionale Modelle ($modelCustomizations), Analysesituationen ($situationCustomizations) und Analysegraph ($graphCustomization) für die Erstellung des angepassten Indyco-Projekts notwendige Dateien in der angegebenen BaseX-Datenbank ($database).
createProject	$datatbase xs:string	Erzeugt für die Erstellung des angepassten Indyco-Builder-Projekts notwendige Dateien und das Indyco-Projekt selbst in der angegebenen BaseX-Datenbank ($database).

Tabelle 22: Funktionen zur Anwendung von Anpassungen

Abbildung 16 zeigt die Projekt-Ansicht in Indyco Builder bei Anwendung der in den letzten Abschnitten beschriebenen Beispiele. Das erzeugte Projekt trägt den Namen **`Graph1GraphCustomization1`**, der einer Kombination des Namens des Graphen aus dem Referenzkatalog **`Graph1`** und der auf diesen angewendeten Anpassung **`GraphCustomization1`** entspricht. Das Projekt enthält das multidimensionale Modell **`InvoiceModelCustomization1`**, dessen Name sich aus dem Namen des multidimensionalen Referenzmodells **`Invoice`** und dem der auf dieses angewendeten Modellanpassung **`ModelCustomization1`** ergibt. Das multidimensionale Modell enthält die nicht über die Modellanpassung abgewählten Hierarchien **`Currency`**, **`Product`**, und **`Time`**, die Hierarchie **`ReturnJust`** aus dem Referenzkatalog für Hierarchien sowie die direkt in der Modellanpassung definierte Hierarchie **`ProfitCenter`**.

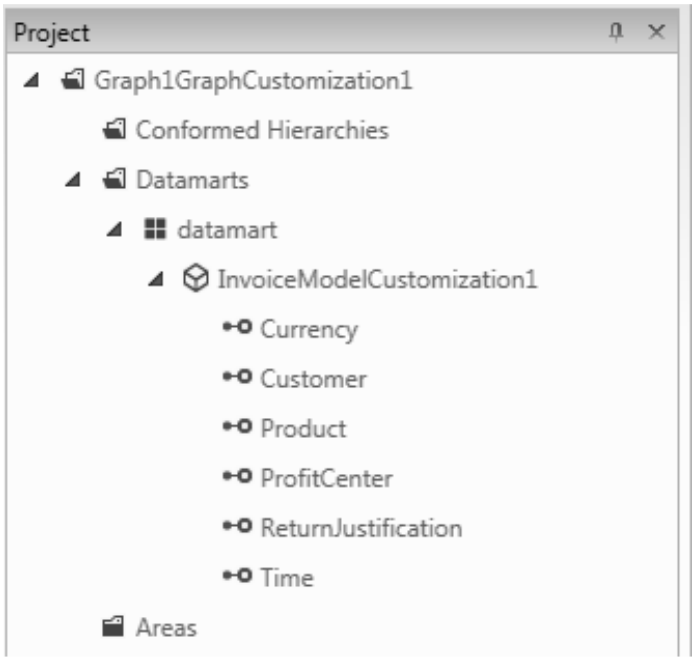

Abbildung 16: Indyco-Projekt-Ansicht eines angepassten multidimensionalen Referenzmodells

In Abbildung 17 wird die Fakt-Ansicht in Indyco Builder des Projektes aus Abbildung 16 gezeigt. Die Faktklasse enthält die nicht über die Modellanpassung abgewählten Kennzahlen **`quantity`**, **`netValue`** und **`sellInMargin`** sowie die direkt in der Modellanpassung definierte Basiskennzahl

netAveragePrice und die Basiskennzahl **vat** aus dem Referenzkatalog für Kennzahlen. Die über die Modellanpassung hinzugefügten berechneten Kennzahlen werden wie bereits erwähnt nicht in Indyco Builder importiert.

Bei den Hierarchien zeigt sich, dass über die Modellanpassung die Levels **county** und **state** der Hierarchie Customer, sowie das Attribut **ageRange** der Hierarchie **Product** abgewählt wurden. Zusätzlich zu den nicht über die Modellanpassung abgewählten Hierarchien des ursprünglichen multidimensionalen Referenzmodells, werden die über die Modellanpassung hinzugefügten Hierarchien **ReturnJust** und **ProfitCenter** angezeigt. Die Hierarchie **Agent** des ursprünglichen multidimensionalen Referenzmodells wurde über die Modellanpassung abgewählt und ist im neuen multidimensionalen Modell deshalb nicht mehr enthalten.

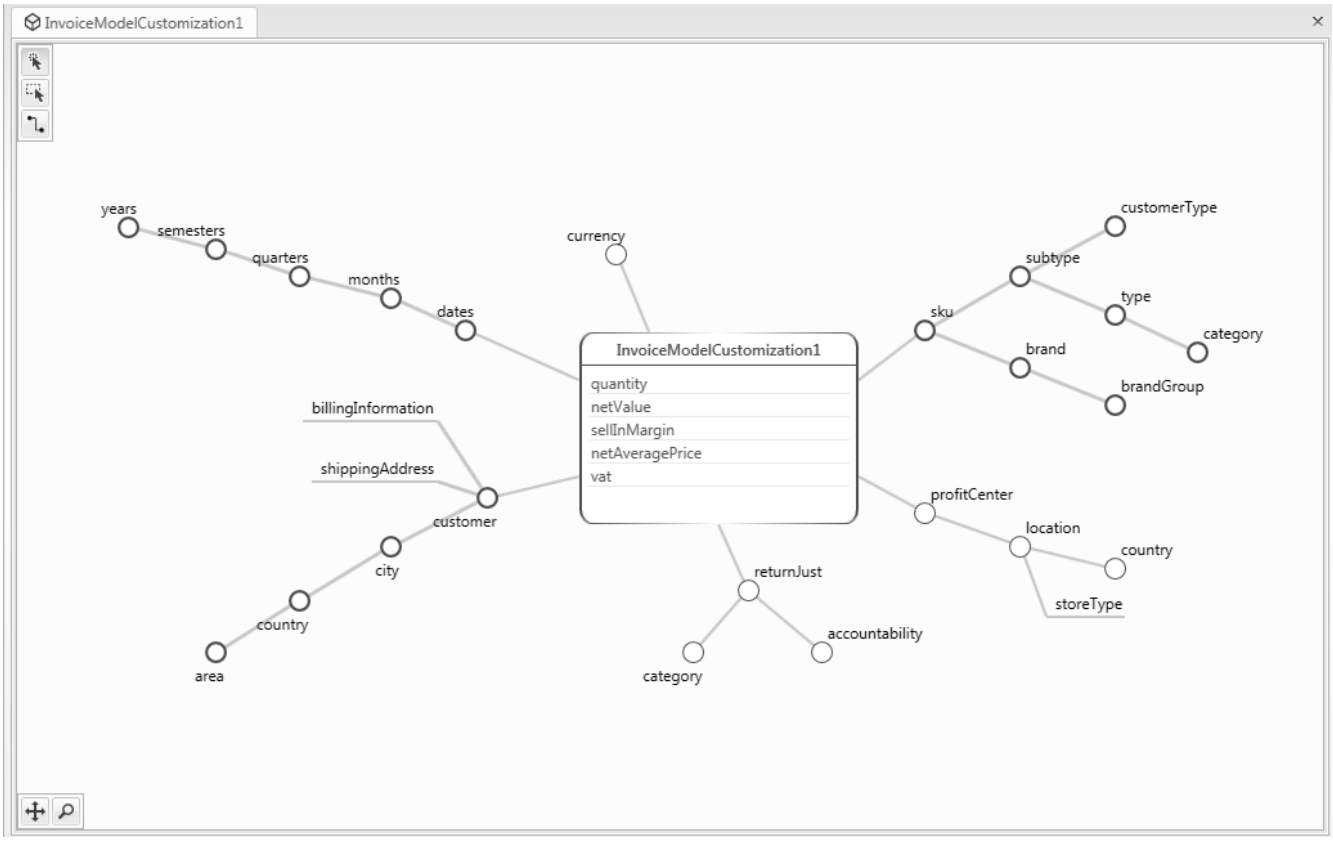

Abbildung 17: Indyco-Fakt-Ansicht eines angepassten multidimensionalen Referenzmodells

Indyco Builder ermöglicht über den Menüpunkt „Tools“ / „Logical modeling“ / „Relational database“ die automatische Erstellung von SQL-Befehlen für die Erstellung von Star-Schema-Tabellen für angepasste multidimensionale Modelle. Für die weiterführenden Funktionen der vorliegenden Implementierung und die Erstellung eines ausführbaren Analysegraphen ist es notwendig, dies für alle Datamarts des Projektes durchzuführen. Dabei ist als Exportformat der von Lane und Potineni (2014) beschriebene Oracle-Standard zu verwenden, den Indyco Builder unter anderem anbietet, und anzugeben, dass neben den Primärschlüsseln auch die Fremdschlüssel exportiert werden sollen. Außerdem soll auch für Dimensionen mit nur einem Level (degenerate dimensions) eine eigene Dimensionstabelle angelegt werden. Um die entsprechende Einstellung in Indyco Builder vornehmen zu können, ist beim Erstellen der Star-Schema-Tabellen als Profil (Profile) der Wert „Custom“ auszuwählen. Für alle anderen Einstellungen können die von Indyco Builder vorgeschlagenen Standard-Werte verwendet werden.

Indyco Builder überprüft bei den auf diese Art generierten Tabellen die Tabellenbezeichnungen auf ihre Länge und kürzt Bezeichner, die die von Oracle festgelegte maximale Länge von 30 Zeichen überschreiten, dementsprechend ab. Bei Spaltennamen ist in Indyco Builder keine derartige Prüfung

hinterlegt, es ist deshalb beim Design und auch bei einer eventuellen Anpassung eines multidimensionalen Modells darauf zu achten, dass die aus Hierarchiename und Level- beziehungsweise Attributname bestehende ID (`Hierarchiename.Levelname` beziehungsweise `Hierarchiename.Attributname`) nicht mehr als 30 Zeichen umfasst. Zusätzlich ist darauf zu achten, keine von Oracle geschützten Schlüsselwörter als Bezeichner für Levels oder Attribute zu wählen.

Wie erwähnt enthält das verwendete Beispielprojekt lediglich einen Datamart. Werden für diesen wie oben beschrieben die SQL-Befehle für die Erstellung der Star-Schema-Tabellen generiert, erstellt Indyco Builder eine Datei vom Typ SQL. Diese ist in Liste 11 dargestellt. Es wird eine Tabelle für die Faktklasse und je eine Tabelle für jede Hierarchie respektive Dimension erstellt, wobei in Liste 11 nur die Dimension `Customer` als Beispiel gezeigt wird. Der Primärschlüssel jeder Dimensionstabelle wird als Fremdschlüssel in der Fakttabelle gespeichert. Die Kombination all dieser Fremdschlüssel bildet den Primärschlüssel der Fakttabelle.

```
01 CREATE TABLE "FT_INVOICEMODELCUSTOMIZATION1"
02 (
03   "CUSTOMER_CUSTOMER" NUMBER (9),
04   "TIME_DATES" NUMBER (9),
05   "CURRENCY_CURRENCY" NUMBER (9),
06   "PRODUCT_SKU" NUMBER (9),
07   "PROFITCENTER_PROFITCENTER" NUMBER (9),
08   "RETURNJUST_RETURNJUST" NUMBER (9),
09   "QUANTITY" NUMBER (18,3),
10   "NETVALUE" NUMBER (18,3),
11   "SELLINMARGIN" NUMBER (18,3),
12   "NETAVERAGEPRICE" NUMBER (18,3),
13   "VAT" NUMBER (18,3)
14 );
15 ALTER TABLE "FT_INVOICEMODELCUSTOMIZATION1" ADD CONSTRAINT
  "PK_FT_INVOICEMODELCUSTOMIZATIO" PRIMARY KEY ("CUSTOMER_CUSTOMER", "TIME_DATES",
  "CURRENCY_CURRENCY", "PRODUCT_SKU", "PROFITCENTER_PROFITCENTER", "RETURNJUST_RETURNJUST");
16 CREATE TABLE "DT_CUSTOMER"
17 (
18   "ID_CUSTOMER" NUMBER (9),
19   "CUSTOMER" VARCHAR2 (255 CHAR),
20   "BILLINGINFORMATION" VARCHAR2 (255 CHAR),
21   "CITY" VARCHAR2 (255 CHAR),
22   "COUNTRY" VARCHAR2 (255 CHAR),
23   "AREA" VARCHAR2 (255 CHAR),
24   "SHIPPINGADDRESS" VARCHAR2 (255 CHAR)
25 );
26 ALTER TABLE "DT_CUSTOMER" ADD CONSTRAINT "PK_DT_CUSTOMER" PRIMARY KEY ("ID_CUSTOMER");
   ...
72 ALTER TABLE "FT_INVOICEMODELCUSTOMIZATION1" ADD CONSTRAINT
  "FK_FT_INVC1_CSTM_CSTM_ID_CSTM" FOREIGN KEY ("CUSTOMER_CUSTOMER") REFERENCES "DT_CUSTOMER"
  ("ID_CUSTOMER");
   ...
```

Liste 11: SQL-Befehle für die Erstellung von Star-Schema-Tabellen

Die Funktionen für die Erstellung von ausführbaren Analysegraphen werden in Tabelle 23 beschrieben. Über die Funktion `importSQL` wird die in Liste 11 beschriebene von Indyco Builder erstellte Datei in die entsprechende BaseX-Datenbank importiert. Ist dies erfolgt, kann mit der Funktion `createExecutableGraph` ein ausführbarer Analysegraph in Form eines SCXML-Schemas erstellt werden.

Die Funktion `clearExecutableGraph` entfernt die über die Funktionen `createFiles`, `createProject` und `importSQL` erstellten Dateien aus der angegebenen BaseX-Datenbank und ermöglicht so die neuerliche Auswahl und Anwendung von Anpassungen und damit die Erstellung eines neuen ausführbaren Analysegraphen. Die erstellte Implementierung kann im Rahmen zukünfter Arbeit einfach dahingehend erweitert werden, bei einer einzigen Ausführung mehrere ausführbare Analysegraphen zu erstellen.

Funktion	Parameter	Beschreibung
importSQL	$database xs:string $sql as xs:string	Importiert die angegebene Datei vom Typ SQL ($sql) in die angegebene Datenbank ($database).
createExecutableGraph	$datatbase xs:string	Erzeugt aus der angegebenen BaseX-Datenbank ($database) einen ausführbaren Analysegraph als SCXML-Schema und gibt diesen aus.
clearExecutableGraph	$database	Löscht die Verzeichnisse für Analysesituationen, Indyco-Projekt und Star-Schema und den angepassten Analysegraph aus der angegebenen BaseX-Datenbank ($database), sodass ein neuer ausführbarer Analysegraph erstellt werden kann.

Tabelle 23: Funktionen zur Erstellung von ausführbaren Analysegraphen

Liste 12 zeigt die Struktur des mit der Funktion `createExecutableGraph` erstellten ausführbaren Analysegraphen unter Anwendung der verwendeten Beispieldaten, wobei aus Gründen der Übersichtlichkeit nur die Analysesituation `Situation1` dargestellt wird. Die Struktur der Analysesituationen `Situation2` und `Situation3` ist analog aufgebaut. Liste 13 bis Liste 18 zeigen Detailansichten bestimmter Elemente des ausführbaren Analysegraphen. Die Zeilennummerierung dieser Listen stimmt mit der aus Liste 12 überein.

Das Wurzelelement des ausführbaren Analysegraphen definiert die verwendeten Namensräume und enthält die Analysesituationen des Graphen als Zustände (States). Für jede Analysesituation ist die SQL-Abfrage angegeben, die die der Analysesituation entsprechende Sicht auf die Fakten im Data Warehouse liefert. Der `SELECT`-Teil dieser SQL-Abfrage enthält die in der Analysesituation angegebenen berechneten Kennzahlen und Basiskennzahlen in Verbindung mit deren Aggregationsoperationen.

Zusätzlich dazu sind im `SELECT`-Teil der SQL-Abfrage die in der Analysesituation angegebenen Granularitätslevels der Dimensionen enthalten. Wurde für eine Dimension kein Granularitätslevel bestimmt, wird für diese der entsprechende Fremdschlüssel aus der Fakttabelle ausgewählt. Liste 12 zeigt, dass für die Dimension `Product` der Granularitätslevel `type` angegeben wurde. Für alle anderen Dimensionen wurden keine Granularitätslevels in der Analysesituation bestimmt.

Der `FROM`-Teil der SQL-Abfrage enthält eine oder mehrere Subabfragen. Werden in der Analysesituation Vergleichskennzahlen verwendet, wird für jede dieser Kennzahlen eine Subabfrage erstellt. Selbes gilt für Kennzahlen, die nicht konkret, sondern als Variable angegeben sind. Sofern die Analysesituation Kennzahlen enthält, die weder Vergleichskennzahlen noch variabel angegebene Kennzahlen sind, werden alle diese in einer einzigen Subabfrage zusammengefasst. Beispiele für Subabfragen finden sich in Liste 13, Liste 14 und Liste 15.

Der Aufbau der SQL-Subabfragen ist für allen Arten von Kennzahlen ähnlich: Der `SELECT`-Teil enthält die Kennzahlen samt eventuellen Berechnungsregeln und die Fremdschlüssel der Fakttabelle für alle Dimensionen, der `FROM`-Teil die Fakttabelle und die Dimensionstabellen, verknüpft über die von Indyco Builder erstellten Fremdschlüssel. Im Fall von Vergleichskennzahlen wird zusätzlich ein Verbund entsprechend der Verbundbedingung der Kennzahl durchgeführt. Da bei Kennzahlen, die als Variable angegeben werden, nicht im Vorhinein bekannt ist, ob es sich um eine arithmetische, kumulative, Vergleichs- oder Basiskennzahl handelt, wird die Subabfrage dementsprechend dynamisch aufgebaut. Wie im Beispiel in Liste 12 und Liste 15 dargestellt, ist die Variable bei ihrem ersten Vorkommen durch den Namen der Kennzahl, beim zweiten durch die Berechnungsregel, beim dritten wieder durch den Namen und beim vierten Vorkommen durch einen eventuellen Vergleichsverbund samt Verbundbedingung zu ersetzen. Der `WHERE`-Teil der Subabfrage enthält die in der Analysesituation angegebenen multidimensionalen Prädikate, Dimensionsprädikate und Kennzahlprädikate.

```
001 <sc:scxml xmlns:ag="http://www.dke.jku.at/AnalysisGraph" xmlns:sc=
   "http://www.w3.org/TR/scxml">
002   <sc:state id="Situation1">
003     <sc:datamodel>
004       <sc:data id="Situation1">
005 SELECT
006 SUM(Fact.annualGrowth),
007 AVG(Fact.grossValue),
008 SUM(Fact.?m1),
009 AVG(Fact.quantity),
010 Fact.CUSTOMER_CUSTOMER,
011 Fact.TIME_DATES,
012 Fact.CURRENCY_CURRENCY,
013 Product.type,
014 Fact.PROFITCENTER_PROFITCENTER,
015 Fact.RETURNJUST_RETURNJUST
016 FROM
017 (
018 SELECT * FROM
    ... Subquery für nicht variable arithmetische, kumulative und Basiskennzahlen
042 NATURAL JOIN
    ... Subquery für eine Vergleichskennzahl
075 NATURAL JOIN
    ... Subquery für eine variable Kennzahl
099 ) Fact JOIN
100 DT_CUSTOMER Customer ON Fact.customer_customer = Customer.id_customer JOIN
101 DT_DATES Time ON Fact.time_dates = Time.id_dates JOIN
102 DT_CURRENCY Currency ON Fact.currency_currency = Currency.id_currency JOIN
103 DT_SKU Product ON Fact.product_sku = Product.id_sku JOIN
104 DT_PROFITCENTER ProfitCenter ON Fact.profitcenter_profitcenter =
   ProfitCenter.id_profitcenter JOIN
105 DT_RETURNJUST ReturnJust ON Fact.returnjust_returnjust = ReturnJust.id_returnjust
106 GROUP BY
107 Fact.CUSTOMER_CUSTOMER,
    ...
110 Product.type,
    ...
113       </sc:data>
114     </sc:datamodel>
115     <sc:transition event="Step1" target="Situation2">
116       <ag:removeMeasure measure="quantity"/>
117       <ag:addMeasure measure="netValue" aggregationOperation="AVG"/>
118       <ag:addMeasure measure="?m2" aggregationOperation="SUM">
119         <ag:var Name="?m2">
              ... Liste möglicher Kennzahlen, die hinzugefügt werden können, siehe Liste 16
127         </ag:var>
128       </ag:addMeasure>
129       <ag:addPredicate predicate="?p2">
130         <ag:var Name="?p2">
              ... Liste möglicher Prädikate, die hinzugefügt werden können, siehe Liste 17
133         </ag:var>
134       </ag:addPredicate>
135       <ag:changeGranularity dimension="Product" level="?l1">
136         <ag:var Name="?l1">
              ... Liste möglicher Levels, die ausgewählt werden können, siehe Liste 18
144         </ag:var>
145       </ag:changeGranularity>
146       <ag:moveToNode dimension="Product" level="?l2" node="?n1">
147         <ag:var Name="?l2">
              ...
155         </ag:var>
156         <ag:var Name="?n1">SELECT DISTINCT ?l2 FROM DT_SKU</ag:var>
157       </ag:moveToNode>
158       <ag:refocusSliceCondition dimension="Product" removedPredicate="groceries"
   addedPredicate="?p3">
159         <ag:var Name="?p3">
160           <ag:predicate Name="groceries" Expression="Product.category = 'grocery'"/>
161         </ag:var>
162       </ag:refocusSliceCondition>
163     </sc:transition>
164   </sc:state>
442 </sc:scxml>
```

Liste 12: Ausführbarer Analysegraph als SCXML-Schema

```
019 (
020 SELECT
021 (Fact.netvalue + Fact.vat) AS grossValue,
022 Fact.quantity,
023 Fact.CUSTOMER_CUSTOMER,
024 Fact.TIME_DATES,
025 Fact.CURRENCY_CURRENCY,
026 Fact.PRODUCT_SKU,
027 Fact.PROFITCENTER_PROFITCENTER,
028 Fact.RETURNJUST_RETURNJUST
029 FROM
030 FT_INVOICEMODELCUSTOMIZATION1 Fact JOIN
031 DT_CUSTOMER Customer ON Fact.customer_customer = Customer.id_customer JOIN
032 DT_DATES Time ON Fact.time_dates = Time.id_dates JOIN
033 DT_CURRENCY Currency ON Fact.currency_currency = Currency.id_currency JOIN
034 DT_SKU Product ON Fact.product_sku = Product.id_sku JOIN
035 DT_PROFITCENTER ProfitCenter ON Fact.profitcenter_profitcenter =
   ProfitCenter.id_profitcenter JOIN
036 DT_RETURNJUST ReturnJust ON Fact.returnjust_returnjust = ReturnJust.id_returnjust
037 WHERE
038 Product.brandGroup = 'austrianBrands' AND
039 Product.category = 'grocery' AND
040 Product.type = 'beverage' AND Time.years = '2015' AND Fact.netValue = 100
041 )
```

Liste 13: SQL-Subquery für nicht variable arithmetische, kumulative und Basiskennzahlen

```
043 (
044 SELECT
045 (Fact.quantity / ComparisonFact.quantity * 100) AS annualGrowth,
046 Fact.CUSTOMER_CUSTOMER,
047 Fact.TIME_DATES,
048 Fact.CURRENCY_CURRENCY,
049 Fact.PRODUCT_SKU,
050 Fact.PROFITCENTER_PROFITCENTER,
051 Fact.RETURNJUST_RETURNJUST
052 FROM
053 FT_INVOICEMODELCUSTOMIZATION1 Fact JOIN
054 DT_CUSTOMER Customer ON Fact.customer_customer = Customer.id_customer JOIN
055 DT_DATES Time ON Fact.time_dates = Time.id_dates JOIN
056 DT_CURRENCY Currency ON Fact.currency_currency = Currency.id_currency JOIN
057 DT_SKU Product ON Fact.product_sku = Product.id_sku JOIN
058 DT_PROFITCENTER ProfitCenter ON Fact.profitcenter_profitcenter =
   ProfitCenter.id_profitcenter JOIN
059 DT_RETURNJUST ReturnJust ON Fact.returnjust_returnjust = ReturnJust.id_returnjust
060 JOIN (
061 FT_INVOICEMODELCUSTOMIZATION1 ComparisonFact JOIN
062 DT_CUSTOMER ComparisonCustomer ON ComparisonFact.customer_customer =
   ComparisonCustomer.id_customer JOIN
063 DT_DATES ComparisonTime ON ComparisonFact.time_dates = ComparisonTime.id_dates JOIN
064 DT_CURRENCY ComparisonCurrency ON ComparisonFact.currency_currency =
   ComparisonCurrency.id_currency JOIN
065 DT_SKU ComparisonProduct ON ComparisonFact.product_sku = ComparisonProduct.id_sku JOIN
066 DT_PROFITCENTER ComparisonProfitCenter ON ComparisonFact.profitcenter_profitcenter =
   ComparisonProfitCenter.id_profitcenter JOIN
067 DT_RETURNJUST ComparisonReturnJust ON ComparisonFact.returnjust_returnjust =
   ComparisonReturnJust.id_returnjust
068 ) ON
069 Fact.product_sku = ComparisonFact.product_sku AND Time.years = (ComparisonTime.years +
   1)
070 WHERE
071 Product.brandGroup = 'austrianBrands' AND
072 Product.category = 'grocery' AND
073 Product.type = 'beverage' AND Time.years = '2015' AND Fact.netValue = 100
074 )
```

Liste 14: SQL-Subquery für eine Vergleichskennzahl

Der gravierendste Unterschied zwischen den Subabfragen der einzelnen Kennzahltypen ist der, dass im Fall von Vergleichskennzahlen wie in Liste 14 und als Variable angegebenen Kennzahlen wie

in Liste 15 für jede Kennzahl eine eigene Subabfrage vorhanden ist. Alle anderen Kennzahlen sind, wie in Liste 13 dargestellt, in einer gemeinsamen Subabfrage zusammengefasst.

```
076 (
077 SELECT
078 %?m1%%?m1%,
079 Fact.CUSTOMER_CUSTOMER,
080 Fact.TIME_DATES,
081 Fact.CURRENCY_CURRENCY,
082 Fact.PRODUCT_SKU,
083 Fact.PROFITCENTER_PROFITCENTER,
084 Fact.RETURNJUST_RETURNJUST
085 FROM
086 FT_INVOICEMODELCUSTOMIZATION1 Fact JOIN
087 DT_CUSTOMER Customer ON Fact.customer_customer = Customer.id_customer JOIN
088 DT_DATES Time ON Fact.time_dates = Time.id_dates JOIN
089 DT_CURRENCY Currency ON Fact.currency_currency = Currency.id_currency JOIN
090 DT_SKU Product ON Fact.product_sku = Product.id_sku JOIN
091 DT_PROFITCENTER ProfitCenter ON Fact.profitcenter_profitcenter =
   ProfitCenter.id_profitcenter JOIN
092 DT_RETURNJUST ReturnJust ON Fact.returnjust_returnjust = ReturnJust.id_returnjust
093 %?m1%
094 WHERE
095 Product.brandGroup = 'austrianBrands' AND
096 Product.category = 'grocery' AND
097 Product.type = 'beverage' AND Time.years = '2015' AND Fact.netValue = 100
098 )
```

Liste 15: SQL-Subquery für eine variable Kennzahl

Neben der SQL-Abfrage sind für jede Analysesituation die Navigationsschritte, denen die jeweilige Analysesituation als Ausgangssituation dient, als Transitionen (Transitions) gespeichert. Dabei sind für jede Transition neben der Zielanalysesituation die zum Erreichen dieser notwendigen Navigationsoperationen angegeben.

Wird bei der Navigationsoperation **addMeasure** die hinzufügende Kennzahl nicht als konkreter Wert, sondern als Variable angegeben, enthält die Operation ein Subelement für diese Variable, in welchem alle möglichen Kennzahlen samt eventuellen Berechnungsregeln und Verbundbedingungen aufgelistet sind. Es werden also bereits beim Erstellen des ausführbaren Analysegraphen jene Kennzahlen ermittelt, die der Analysesituation hinzugefügt werden können. So müssen beim Ersetzen der Variable bei der Durchführung der Analyse keine Informationen mehr aus dem Referenzkatalog für Kennzahlen eingeholt werden. Ein Beispiel dafür zeigt Liste 16.

```
120 <ag:measure Name="grossValue" CalculationRule="(Fact.netvalue + Fact.vat) AS "/>
121 <ag:measure Name="annualGrowth" CalculationRule="(Fact.quantity /
   ComparisonFact.quantity * 100) AS " JoinCondition="JOIN ( FT_INVOICEMODELCUSTOMIZATION1
   ComparisonFact JOIN  DT_CUSTOMER ComparisonCustomer ON ComparisonFact.customer_customer =
   ComparisonCustomer.id_customer JOIN  DT_DATES ComparisonTime ON ComparisonFact.time_dates
   = ComparisonTime.id_dates JOIN  DT_CURRENCY ComparisonCurrency ON
   ComparisonFact.currency_currency = ComparisonCurrency.id_currency JOIN  DT_SKU
   ComparisonProduct ON ComparisonFact.product_sku = ComparisonProduct.id_sku JOIN
   DT_PROFITCENTER ComparisonProfitCenter ON ComparisonFact.profitcenter_profitcenter =
   ComparisonProfitCenter.id_profitcenter JOIN  DT_RETURNJUST ComparisonReturnJust ON
   ComparisonFact.returnjust_returnjust = ComparisonReturnJust.id_returnjust ) ON
   Fact.product_sku = ComparisonFact.product_sku AND Time.years = (ComparisonTime.years +
   1)"/>
122 <ag:measure Name="quantity"/>
123 <ag:measure Name="netValue"/>
124 <ag:measure Name="sellInMargin"/>
125 <ag:measure Name="netAveragePrice"/>
126 <ag:measure Name="vat"/>
```

Liste 16: Mögliche Kennzahlen, die über eine Navigationsoperation hinzugefügt werden können

Analog besteht beim Hinzufügen beziehungsweise Ersetzen von Prädikaten über die Navigationsoperationen **addPredicate**, **addSliceCondition** und **refocusSliceCondition** die Möglichkeit, anstelle des neuen Prädikats eine Variable anzugeben. In diesem Fall wird der Operation ein entsprechendes Subelement hinzugefügt, welches wiederum alle für die Analysesituation anwendbaren Prädikate samt den Prädikatausdrücken enthält. Ein Beispiel dafür ist in Liste 17 dargestellt.

```
131 <ag:predicate Name="beveragesNovember2015" Expression="Product.type = 'beverage' AND
    Time.months = 'november' AND Time.years = '2015'"/>
132 <ag:predicate Name="premiumBeverages2015" Expression="Product.type = 'beverage' AND
    Time.years = '2015' AND Fact.netValue = 100"/>
```

Liste 17: Mögliche Prädikate, die über eine Navigationsoperation hinzugefügt werden können

Werden über die Navigationsoperationen **moveToNode** der Dice-Knoten oder über die Funktion **changeGranularity** die Granularität einer Dimension verändert, kann anstelle eines konkreten Levels ebenfalls eine Variable gesetzt werden. Ist dies der Fall, enthält die Operation ein Subelement für diese Variable, in der alle Levels der angegebenen Dimension aufgelistet werden, wie in Liste 18 beispielhaft dargestellt. Wird bei der Operation **moveToNode** der Dice-Knoten selbst als Variable angegeben, wird ein weiteres Subelement erstellt, welches die SQL-Abfrage zum Abfragen der möglichen Werte aus der entsprechenden Dimensionstabelle enthält. Dies wurde beispielhaft bereits in Liste 12 dargestellt.

```
137 <ag:level Name="SKU"/>
138 <ag:level Name="SUBTYPE"/>
139 <ag:level Name="TYPE"/>
140 <ag:level Name="CATEGORY"/>
141 <ag:level Name="CUSTOMERTYPE"/>
142 <ag:level Name="BRAND"/>
143 <ag:level Name="BRANDGROUP"/>
```

Liste 18: Mögliche Levels, die über eine Navigationsoperation ausgewählt werden können

4 Aufbau der BIRD-Implementierung

Nachdem im vorhergehenden Abschnitt die Verwendung der erstellten BIRD-Implementierung aus Benutzersicht erläutert wurde, beschäftigt sich dieser Abschnitt mit der internen Funktionalität der XQuery-Funktionen. Abbildung 18 zeigt die Struktur der Implementierung anhand der umgesetzten XQuery-Module. Diese können gedanklich drei Ebenen zugeordnet werden.

Das Modul `referenceModeling` enthält die in Abschnitt 3.2 erläuterten Interface-Funktionen und entspricht der Interface-Ebene oder Ebene 1 der Implementierung. Die Funktionen des Moduls `referenceModeling` greifen ausschließlich auf Funktionen der Module auf Ebene zwei sowie auf private Funktionen innerhalb des Moduls `referenceModeling` zu.

Ebene zwei umfasst die Module `measureCatalogueParsing`, `predicateCatalogueParsing`, `fileGeneration`, `projectGeneration` und `executableGeneration`. Jedes dieser Module enthält genau eine öffentliche Funktion, auf die von Funktionen des Moduls `referenceModeling` zugegriffen werden kann. Zusätzlich dazu enthalten die Module auf Ebene zwei auch private Funktionen, auf die nur von Funktionen des selben Moduls zugegriffen werden kann. Die Module der Ebene zwei greifen ausschließlich auf Funktionen der Module auf Ebene drei zu.

Ebene drei umfasst die Module `hierarchyRedefinition`, `measureRedefinition`, `predicateRedefinition`, `modelCustomization`, `analysisSituationCustomization` und `analysisGraph-Customization`. Wie die Module der Ebene zwei enthält auch jedes Modul der Ebene drei genau eine öffentliche Funktion. Diese ist den Funktionen der Module auf Ebene zwei zugänglich. Zusätzlich dazu enthalten die Module ebenfalls private Funktionen, auf die nur von Funktionen des selben Moduls zugegriffen werden kann.

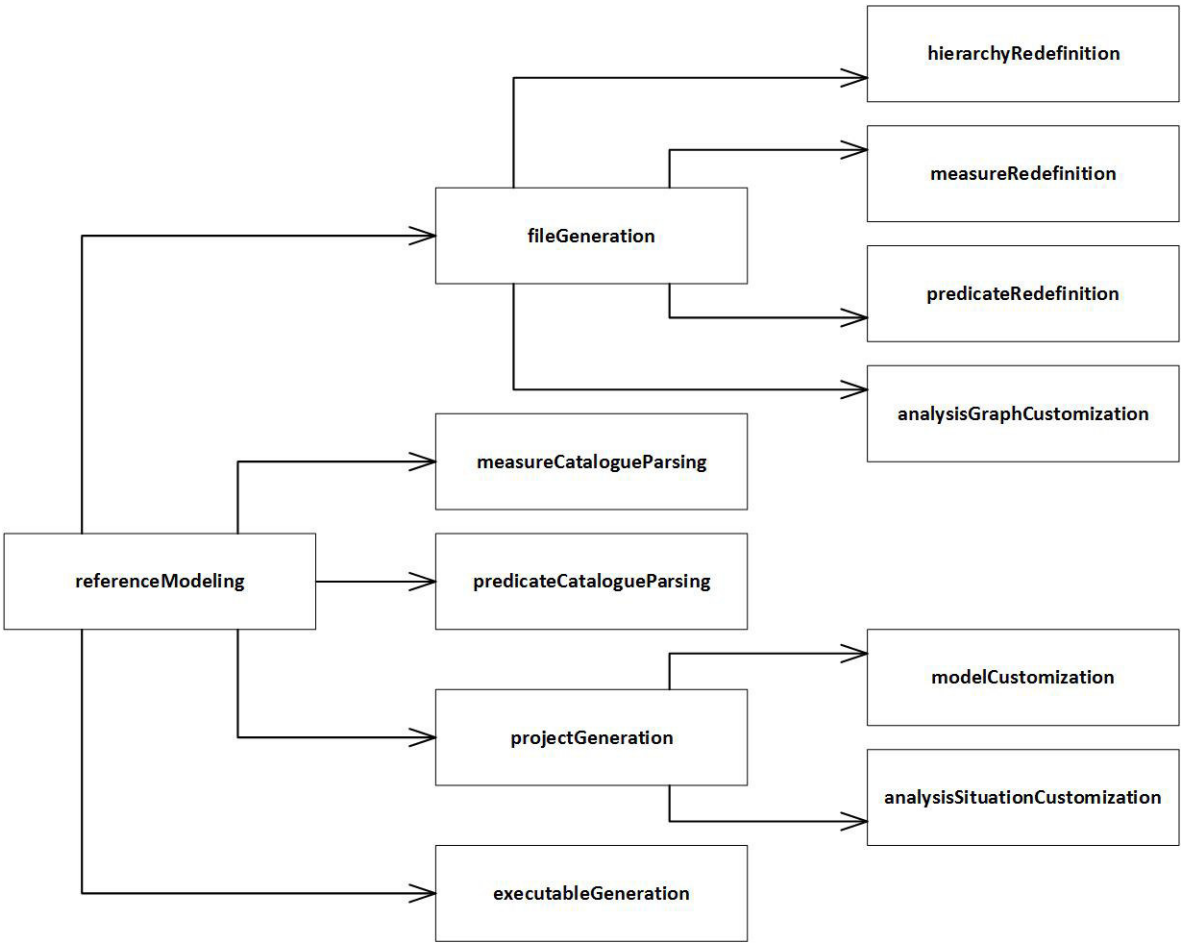

Abbildung 18: Implementierte XQuery-Module

Die Interface-Funktionen des Moduls **`referenceModeling`** wurden bereits in Abschnitt 3.2 detailliert beschrieben, Abschnitt 4.1 beschreibt die privaten Funktionen sowie die Aufrufe zwischen den Funktionen dieses Moduls. Die Funktionen der Module der Ebene zwei und drei werden im Anschluss in Abschnitt 4.2 bis 4.5 im Detail diskutiert. Abschnitt 4.6 widment sich den Fehlermeldungen, die ausgelöst werden können.

analysisGraphCustomizations	GraphCustomization1.xml GraphCustomization2.xml …	
analysisSituationCustomizations	SituationCustomization1.xml SituationCustomization2.xml …	
analysisSituations	Situation1	analysisSituationCustomization.xml customizedModel.xml customizedSituation.xml hierarchyCatalogue.xml measureCatalogue.xml modelCustomization.xml predicateCatalogue.xml uuid.xml
	Situation2	…
	Situation3	…
indycoProject	Datamarts	357a03ba-9a11-430c-b63e-bf0290acc66e.dfm 908c74f3-ed35-49a9-a920-a0b7787c9ffa.dfm eccd4e2a-abcd-47f8-bc99-227658d70748.dfm
	Graph1.dfmproj	
modelCustomizations	ModelCustomization1.xml ModelCustomization2.xml …	
models	7b7d4c05-d2fd-4957-8ad9-ed89bfa6a707.xml 1f9fcf0a-4ea5-4eca-88d8-6c352ac13f6f.xml …	
starSchema	Oracle.xml	
analysisGraphCatalogue.xml analysisSituationCatalogue.xml customizedGraph.xml executableGraph.scxml hierarchyCatalogue.xml predicateCatalogue.xml		

Tabelle 24: Dateistruktur der BaseX-Datenbank

Bevor jedoch in den nächsten Abschnitten im Detail auf die Funktionalität der erstellten Implementierung eingegangen wird, soll noch die Dateistruktur der verwendeten BaseX-Datenbanken beleuchtet werden. Tabelle 24 zeigt, dass für die Speicherung der Anpassungen für multidimensionale

Referenzmodelle, Analysesituationen und Analysegraphen eigene Verzeichnisse namens **modelCustomizations**, **analysisSituationCustomizations** und **analysisGraphCustomizations** vorhanden sind. Auch die von Indyco Builder erstellten Faktklassendateien werden in einem eigenen Verzeichnis, dem Verzeichnis **models** abgelegt. Die Referenzkataloge für Hierarchien (**hierarchyCatalogue**), Kennzahlen (**measureCatalogue**), Prädikate (**predicateCatalogue**), Analysesituationen (**analysisSituationCatalogue**) und Analysegraphen (**analysisGraphCatalogue**) sind im übergeordneten Verzeichnis gespeichert.

Der in Tabelle 24 dargestellte Status entspricht dem Zustand der Datenbank nachdem die in Abschnitt 3.2.10 beschriebenen Funktionen zur Erstellung eines ausführbaren Analysgraphen mit den verwendeten Beispieldaten ausgeführt wurden. Der angepasste Analysegraph ist als **customizedAnalysisGraph** im übergeordneten Verzeichnis abgelegt. Für jede Analysesituation dieses angepassen Graphen wurde im Verzeichnis **analysisSituations** ein Subverzeichnis angelegt, welches die für die Erstellung eines angepassten Indyco-Projekts und des ausführbaren Analysegraphen benötigten Hilfsdateien enthält.

Das angepasste Indyco-Projekt samt seinen Datamarts ist im Verzeichnis **indycoProject** zu finden. Bei Import der von Indyco Builder erstellen SQL-Befehlen zur Erstellung der Star-Schema-Tabellen, werden diese in einer Datei im Ordner **starSchema** gespeichert. Der über die Funktion **createExecutableGraph** erstellte ausführbare Analysegraph ist als Datei vom Typ SCXML im übergeordneten Verzeichnis abgelegt.

4.1 Modul referenceModeling

Das Modul **referenceModeling** stellt das einzige Modul auf Ebene eins dar. Wie bereits erwähnt, wird an dieser Stelle auf die detaillierte Dokumentation der öffentlichen Interface-Funktionen dieses Moduls verzichtet, da diese bereits in Abschnitt 3.2 eingehend diskutiert wurden. Tabelle 25 gibt einen Überblick über die privaten Funktionen dieses Moduls, die von den Interface-Funktionen verwendet werden.

Jede der Interface-Funktionen, die eine Datei, also einen Referenzkatalog, eine Anpassung oder eine Faktklassendatei in einer BaseX-Datenbank erstellt, verwendet für die Erzeugung der Datei die private Funktion **insertFile**. Diese prüft, ob am entsprechenden Pfad der angegebenen BaseX-Datenbank bereits eine Datei vorhanden ist und gibt wenn dem so ist eine Fehlermeldung aus. Ist der Pfad noch nicht belegt, wird die Datei eingefügt. So wird vermieden, dass für eine BaseX-Datenbank mehrere Referenzkataloge des gleichen Typs oder mehrere Anpassungen oder Faktklassendateien mit dem selben Namen angelegt werden.

Die Interface-Funktionen, die ein Element in eine vorhandene Datei einer BaseX-Datenbank einfügen, verwenden dafür die private Funktion **insertNode**. Diese prüft, ob an der gewünschten Stelle bereits ein Element mit der selben eindeutigen ID vorhanden ist, und gibt in diesem Fall eine entsprechende Fehlermeldung aus. Eine Fehlermeldung wird auch ausgegeben, wenn das Superelement, in welches das neue Element eingefügt werden soll, nicht vorhanden ist.

Funktion	Parameter	Beschreibung
insertFile %private	$database xs:string $file item() $path xs:string	Fügt die angegebene Datei ($file) am angegebenen Pfad ($path) in die angegbene BaseX-Datenbank ($database) ein, sofern am angegebenen Pfad noch keine Datei vorhanden ist.
insertNode %private	$element element() $node node()	Fügt den angegebenen Knoten ($node) in das angegebenen Element ($element) ein, sofern noch kein Knoten mit der selben eindeutigen ID vorhanden ist.
$getNodeIds %private	$nodes element()*	Ermittelt für die angegebenen Knoten ($nodes) die für die eindeutige Identifizierung notwendigen IDs.

Tabelle 25: Private Funktionen des Moduls referenceModeling

Zur Prüfung, ob an einer gewünschten Stelle bereits ein Element mit der selben eindeutigen ID wie der des einzufügenden Elements vorhanden ist, verwendet die private Funktion `insertNode` die ebenfalls private Funktion `getNodeIds`. Diese ermittelt je nach Typ des übergebenen Elements eine eindeutige ID. Bei den meisten Elementen entspricht diese dem angegebenen Namen des Elements. Bei Levels und Attributen wird eine Kombination aus Hierarchiename und Level- beziehungsweise Attributname als eindeutige ID herangezogen. Bei Neudefinitionen wird nicht der Name der Neudefinition, sondern der des neu definierten Prädikats oder Navigationsschrittes beziehungsweise der neu definierten Kennzahl oder Dimension als eindeutige ID verwendet. So wird vermieden, dass für eine Anpassung mehrere Neudefinitionen für das selbe Prädikat, den selben Navigationsschritt oder die selbe Hierarchie oder Kennzahl erstellt werden.

4.2 Module measureCatalogueParsing und predicateCatalogueParsing

Tabelle 26 beschreibt die Funktionen des Moduls `measureCatalogueParsing`. Dessen öffentliche Funktion `parse` kann über die Funktion `parseCatalogues` des Moduls `referenceModeling` aufgerufen werden und ruft ihrerseits für jede arithmetische Kennzahl des angegebenen Referenzkatalogs für Kennzahlen die private Funktion `parseCalculation` auf. Diese liest aus der Berechnungsregel der Kennzahl rekursiv die für die Berechnung benötigten Kennzahlen und Attribute aus und gibt die zu erzeugenden Subelemente an die Funktion `insertNode` weiter. Diese prüft mit Hilfe der privaten Funktion `getNodeIds`, die aus der Kombination des Namens des Elements und eines eventuell vorhandenen Hierarchienamens eine eindeutige ID generiert, ob bereits ein identisches Subelement vorhanden ist. Ist dem nicht so, fügt die Funktion `insertNode` das Subelement in den Referenzkatalog für Kennzahlen ein.

Funktion	Parameter	Beschreibung
parse	$measureCatalogue element()	Ergänzt im angegebenen Referenzkatalog für Kennzahlen ($measureCatalogue) die für die Verwendung der Kennzahlen notwendigen Elemente.
parseCalculation %private	$measure element() $string xs:string	Fügt die Kennzahlen und Attribute, auf die die Berechnungsregel der angegebenen Kennzahl ($measure) verweist, in deren Subelemente ein.
parseOver %private	$measure element() $string xs:string	Fügt die Levels und Hierarchien, auf die der OVER-Teil der Berechnungsregel der angegebenen kumulativen Kennzahl ($measure) verweist, in deren Subelemente ein.
parseJoin %private	$measure element() $string xs:string	Fügt die Levels und Hierarchien, auf die die Verbundbedingung der angegebenen Vergleichskennzahl ($measure) verweist, in deren Subelemente ein.
insertNode %private	$element element() $node node() $parsedText xs:string	Fügt den angegebenen Knoten ($node) in das angegebenen Element ($element) ein, sofern noch kein identischer Knoten vorhanden ist.
getNodeIds %private	$nodes element()*	Ermittelt für die angegebenen Kennzahl-, Attribut-, Level- und Hierarchieknoten ($nodes) die für die eindeutige Identifizierung notwendigen IDs.

Tabelle 26: Funktionen des Moduls measureCatalogueParsing

Für Vergleichskennzahlen ist die Vorgehensweise die selbe. Zusätzlich zur Funktion `parseCalculation` ruft die Funktion `parse` für alle Vergleichskennzahlen im angegebenen

Referenzkatalog für Kennzahlen aber auch noch die private Funktion **parseJoin** auf. Diese liest aus der Verbundbedingung der Kennzahl rekursiv die für die Berechnung zusätzlich benötigten Levels und Hierarchien aus. Die Subelemente werden wie bei der Funktion **parseCalculation** über die Funktion **insertNode** eingefügt.

Für die kumulativen Kennzahlen des angegebenen Referenzkatalogs für Kennzahlen teilt die Funktion **parse** die Berechnungsregel in zwei Teile. Die Trennung erfolgt dabei beim Stichwort **over**. Der erste Teil der Berechnungsregel wird an die Funktion **parseCalculation** übergeben, der zweite an die private Funktion **parseOver**. Diese liest die für die Berechnung benötigten Levels und Hierarchien rekursiv aus und übergibt die zu erstellenden Subelemente an die Funktion **insertNode**.

Die in Tabelle 27 dargestellten Funktionen des Moduls **predicateCatalogueParsing** ähneln in ihrer Funktionalität denen des Moduls **measureCatalogueParsing**, jedoch werden hierbei nicht die Berechnungsregeln und Verbundbedingungen von berechneten Kennzahlen, sondern die Ausdrücke von Prädikaten eines Referenzkatalogs für Prädikate analysiert.

Die über die Funktion **parseCatalogues** des Moduls **referenceModeling** aufrufbare öffentliche Funktion **parse** übergibt die Ausdrücke aller Dimensionsprädikate im Referenzkatalog an die private Funktion **parseDimensionPredicate**, die aller multidimensionalen Prädikate an die private Funktion **parseMultidimensionalPredicate** und die aller Kennzahlprädikate an die Funktion **parseMeasurePredicate**. Diese lesen rekursiv die benötigten Kennzahlen, Attribute und Hierarchien aus und übergeben die zu erzeugenden Subelemente an die private Funktion **insertNode**.

Funktion	Parameter	Beschreibung
parse	$predicateCatalogue element()	Ergänzt im angegebenen Referenzkatalog für Prädikate ($predicateCatalogue) die für die Verwendung der Kennzahlen notwendigen Elemente.
parseMeasurePredicate %private	$predicate element() $string xs:string	Fügt die Hierarchien, Attribute und Kennzahlen, auf die der Ausdruck des angegebenen Kennzahlprädikats ($predicate) verweist, in dessen Subelemente ein.
parseMultidimensionalPredicate %private	$predicate element() $string xs:string	Fügt die Hierarchien und Attribute, auf die der Ausdruck des angegebenen multidimensionalen Prädikats ($predicate) verweist, in dessen Subelemente ein.
parseDimensionPredicate %private	$predicate element() $string xs:string	Fügt die Hierarchie und die Attribute, auf die der Ausdruck des angegebenen Dimensionsprädikats ($predicate) verweist, in dessen Subelemente ein.
insertNode %private	$element element() $node node() $parsedText xs:string	Fügt den angegebenen Knoten ($node) in das angegebenen Element ($element) ein, sofern noch kein identischer Knoten vorhanden ist.
getNodeIds %private	$nodes element()*	Ermittelt für die angegebenen Kennzahl-, Attribut-, und Hierarchieknoten ($nodes) die für die eindeutige Identifizierung notwendigen IDs.

Tabelle 27: Funktionen des Moduls predicateCatalogueParsing

Die Funktion **insertNode** gleicht der gleichnamigen Funktion des Moduls **measureCatalogueParsing** und arbeitet wie diese mit einer weiteren privaten Funktion namens **getNodeIds**. Auch diese gleicht der gleichnamigen Funktion des Moduls **measureCatalogueParsing**. Diese Funktionen wurden sowohl im Modul **measureCatalogueParsing** als auch im Modul **predicateCatalogueParsing** umgesetzt, um zu

gewährleisten, dass jedes der Module lediglich eine öffentliche Funktion aufweist und kein Modul der Ebene zwei auf Funktionen eines Moduls der gleichen Ebene zugreift.

4.3 Modul fileGeneration und von diesem verwendete Module

Das Modul **`fileGeneration`** enthält als einziges keine privaten Funktionen, umfasst also nur eine öffentliche Funktion namens **`createFiles`**. Diese wird von der gleichnamigen Funktion des Moduls **`referenceModeling`** aufgerufen und ist in Tabelle 28 beschrieben. Die Funktion benötigt als Eingabewerte neben dem Namen der verwendeten BaseX-Datenbank den Namen der gewünschten Anpassung für einen Analysegraphen. Soll ein Analysegraph ohne vorherige Anpassung direkt aus dem Referenzkatalog der BaseX-Datenbank verwendet werden, ist anstelle des Namens der Anpassung der Name des Analysegraphen anzugeben.

Außerdem sind der Funktion die Namen der anzuwendenden Anpassungen für multidimensionale Modelle und Analysesituationen zu übergeben. Dabei kann für jedes multidimensionale Modell und für jede Analysesituation maximal eine Anpassung angegeben werden. Werden keine Anpassungen gewünscht, ist beim entsprechenden Übergabewert ein leerer String zu übergeben.

Auf Basis der übergebenen Werte wendet die Funktion **`createFiles`** über die öffentliche Funktion des Moduls **`analysisGraphCustomization`** auf Ebene drei der Implementierung die übergebene Anpassung für einen Analysegraphen an und speichert den so angepassten Analysegraphen. Wurde anstelle der Anpassung direkt ein Graph aus dem Referenzkatalog übergeben, so wird dieser als angepasster Analysegraph gespeichert.

Für jede Analysesituation des angepassten Analysegraphen wird die entsprechende Anpassung gespeichert. Wurde keine Anpassung für die Analysesituation übergeben, wird eine leere Anpassung gespeichert. Das selbe erfolgt für die Anpassung der von der Analysesituation referenzierten Faktklasse.

Außerdem werden für jede Analysesituation des angepassten Analysegraphen unter Berücksichtigung der Neudefinitionen in der Anpassung für das multidimensionale Modell der von der Analysesituation referenzierten Faktklasse angepasste Referenzkataloge für Kennzahlen und Prädikate erstellt. Zusätzlich wird ein angepasster Referenzkatalog für Hierarchien erstellt, der die in der Anpassung des multidimensionalen Modells der referenzierten Faktklasse abgewählten Levels und Attribute berücksichtigt. Für die Bewältigung dieser Aufgaben werden die öffentlichen Funktionen der Module **`hierarchyRedefinition`**, **`measureRedefinition`** und **`predicateRedefinition`** der Ebene drei verwendet.

Die Funktion **`createFiles`** speichert weiters für jede Analysesituation des angepassten Analysegraphen eine eindeutige UUID zur späteren Verwendung. Außerdem erstellt sie für den angepassten Analysegraphen eine Indyco-Projektdatei, der jedoch noch keine Faktklassendateien zugeordnet werden.

Funktion	Parameter	Beschreibung
createFiles	$database xs:string $graphCustomization xs:string $modelCustomizations xs:string* $situationCustomizations xs:string*	Erzeugt auf Basis der angegebenen Anpassungen für multidimensionale Modelle ($modelCustomizations), Analysesituationen ($situationCustomizations) und Analysegraph ($graphCustomization) für die Erstellung des angepassten Indyco-Projekts und des ausführbaren Analysegraphen notwendige Dateien in der angegebenen BaseX-Datenbank ($database).

Tabelle 28: Funktionen des Moduls fileGeneration

Tabelle 29 enthält die Funktionen des Moduls **hierarchyRedefinition**. Die öffentliche Funktion **redefine** dieses Moduls erstellt auf Basis des angegebenen Referenzkatalogs für Hierarchien einen neuen, angepassten Referenzkatalog, der die über eine angegebene Anpassung für ein multidimensionales Referenzmodell abgewählten Levels und Attribute berücksichtigt.

Die Funktion **redefine** prüft dafür für jede Hierarchie des angegebenen Referenzkatalogs für Hierarchien, ob diese über die angegebene Anpassung einem multidimensionalen Modell hinzugefügt wurde. Ist dem nicht so, wird die Hierarchie unverändert in den angepassten Referenzkatalog übernommen.

Wurde die Hierarchie über die angegebene Anpassung hinzugefügt, prüft die Funktion **redefine** über die private Funktion **checkConsistency**, ob über die Anpassung Levels oder Attribute abgewählt wurden, die im Referenzkatalog als verpflichtend gekennzeichnet sind. Ist dies der Fall, wird eine entsprechende Fehlermeldung ausgegeben und die Verarbeitung abgebrochen.

Anschließend prüft die Funktion **redefine** über die private Funktion **getDeselectedNodes**, ob Levels dieser Hierarchie über die Anpassung abgewählt wurden. Diese private Funktion vergleicht dafür den Namen der zu prüfenden Hierarchie mit denen der Hierarchien in der Liste der abgewählten Levels der Anpassung. Der entsprechenden Hierarchie werden im angepassten Referenzkatalog lediglich die Levels hinzugefügt, die nicht auf diese Art abgewählt wurden.

Für jedes Attribut der auf diese Art dem angepassten Referenzkatalog hinzugefügten Levels erfolgt in der Funktion **redefine** die selbe Prüfung. Dazu wird wiederum die Funktion **getDeselectedNodes** verwendet, die den Namen der zu prüfenden Hierarchie mit denen der Hierarchien in der Liste der abgewählten Attribute der Modellanpassung vergleicht. Nur Attribute, die nicht über diese Liste abgewählt wurden, werden dem angepassten Referenzkatalog hinzugefügt.

Eine analoge Vorgehensweise gilt für die Superlevels der nicht abgewählten Levels der Hierarchie. Wurde der Superlevel eines Levels abgewählt, ermittelt die Funktion **redefine** über die private Funktion **getSuperLevel** rekursiv den neuen Superlevel in der angepassten Hierarchie.

Funktion	Parameter	Beschreibung
redefine	$modelCustomization elment() $hierarchyCatalogue element()	Erzeugt unter Berücksichtigung der abgewählten Levels und Attribute in der angegebenen Anpassung für ein multidimensionales Modell ($modelCustomization) einen neuen Referenzkatalog für Hierarchien auf Basis des angegebenen Referenzkatalogs für Hierarchien ($hierarchyCatalogue).
getDeselectedNodes %private	$hierarchy element() $deselectedNodes element()*	Ermittelt aus der angegebenen Liste von abgewählten Elementen ($deselectedNodes) diejenigen, die der angegebenen Hierarchie ($hierarchy) angehören.
getSuperLevels %private	$level element() $deselectedLevels element()*	Ermittelt unter Berücksichtigung der angegebenen Liste von abgewählten Levels ($deselectedLevels) rekursiv die Superlevels des angegebenen Levels ($level).
checkConsistency %private	$modelCustomization element() $hierarchy element()	Überprüft die Konsistenz der angegebenen Anpassung für ein multidimensionales Modell ($modelCustomization) gegenüber der angegebenen Hierarchie ($hierarchy).

Tabelle 29: Funktionen des Moduls hierarchyRedefinition

Wie die Funktionen des Moduls **hierarchyRedefinition** dienen auch die Funktionen des Moduls **measureRedefinition** der Erstellung eines angepassten Referenzkatalogs. In diesem Fall wird jedoch ein angegebener Referenzkatalog für Kennzahlen angepasst, indem die in einer angegebenen

Anpassung für ein multidimensionales Referenzmodell enthaltenen Neudefintionen für berechnete Kennzahlen umgesetzt werden.

Tabelle 30 zeigt die Funktionen des Moduls **measureRedefinition**. Die öffentliche Funktion **redefine** übergibt zuerst den angegebenen Referenzkatalog für Kennzahlen und die angegbene Anpassung für ein multidimensionales Modell an die private Funktion **checkConsistency**. Diese ermittelt über die private Funktion **getRedefinition**, welche berechneten Kennzahlen des Referenzkatalogs über die Anpassung neu definiert werden. Dazu werden die Namen der Kennzahlen im Referenzkatalog mit den Namen der neu definierten Kennzahlen in der Liste der Kennzahl-Neudefinitionen der Anpassung verglichen und eventuelle Neudefinitionen zurückgegeben.

Im Anschluss prüft die Funktion **checkConsistency**, ob alle als verpflichtend gekennzeichneten Kennzahlen, Attribute, Levels und Hierarchien auch in den Neudefinitionen der jeweiligen Kennzahlen vorhanden sind. Ist dem nicht so, wird eine entsprechende Fehlermeldung ausgegeben und die Verarbeitung abgebrochen.

Nach der Konsistenzprüfung prüft die Funktion **redefine** für jede berechnete Kennzahl im angegebenen Referenzkatalog über die Funktion **getRedefinition** erneut, ob es in der angegebenen Anpassung neu definiert wurde. Nicht neu definierte berechnete Kennzahlen sowie Basiskennzahlen werden unverändert in den angepassten Referenzkatalog übernommen. Gibt es eine Neudefinition, wird die Kennzahl mit der neuen Berechnungsregel im angepassten Referenzkatalog angelegt. Die Subelemente für die Erfassung der für die Berechnung benötigten Kennzahlen, Attribute, Levels und Hierarchien werden dabei geleert. Diese müssen im Anschluss erneut wie im vorhergehenden Abschnitt erläutert über die Funktion **parseMeasureCatalogue** befüllt werden.

Funktion	Parameter	Beschreibung
redefine	$modelCustomization elment() $measureCatalogue element()	Erzeugt unter Berücksichtigung der Kennzahl-Neudefinitionen in der angegebenen Anpassung für ein multidimensionales Modell ($modelCustomization) einen neuen Referenzkatalog für Kennzahlen auf Basis des angegebenen Referenzkatalogs für Kennzahlen ($measureCatalogue).
getRedefinition %private	$measure element() $redefinitions element()*	Ermittelt aus der angegebenen Liste von Kennzahl-Neudefinitionen ($redefinitions) die Neudefinition für die angegebene Kennzahl ($measure).
checkConsistency %private	$measureCatalogue element() $redefinitions element()*	Überprüft die Konsistenz der angegebenen Kennzahl-Neudefinitionen ($redefinitions) gegenüber dem angegebenen Referenzkatalog für Kennzahlen ($measureCatalogue).

Tabelle 30: Funktionen des Moduls measureRedefinition

Die in Tabelle 31 dargestellten Funktionen des Moduls **predicateRedefinition** ähneln in ihrer Funktionalität und Struktur denen des Moduls **measureRedefinition**, allerdings in Bezug auf Prädikate. Die öffentliche Funktion **redefine** übergibt den angegebenen Referenzkatalog für Prädikate sowie die angegebene Anpassung für ein multidimensionales Referenzmodell an die private Funktion **checkConsistency**. Diese ermittelt über die Funktion **getRedefinition**, welche Prädikate des Referenzkatalogs über die Anpassung neu definiert wurden. Die Funktion **getRedefinition** vergleicht dazu die Namen der Prädikate im Referenzkatalog mit denen der neu definierten Prädikate in der Liste der Prädikat-Neudefinitionen der Anpassung.

Daraufhin prüft die Funktion **checkConsistency**, ob alle als verpflichtend markierten Kennzahlen, Attribute und Hierarchien auch in den neu definierten Ausdrücken der entsprechenden Prädikate vorhanden sind. Verstößt eine Neudefinition gegen diese Vorschrift, wird eine Fehlermeldung ausgegeben und die Verarbeitung abgebrochen.

Funktion	Parameter	Beschreibung
redefine	$modelCustomization elment() $predicateCatalogue element()	Erzeugt unter Berücksichtigung der Prädikat-Neudefinitionen in der angegebenen Anpassung für ein multidimensionales Modell ($modelCustomization) einen neuen Referenzkatalog für Prädikate auf Basis des angegebenen Referenzkatalogs für Prädikate ($predicateCatalogue).
getRedefinition %private	$predicate element() $redefinitions element()*	Ermittelt aus der angegebenen Liste von Prädikat-Neudefinitionen ($redefinitions) die Neudefinition für das angegebene Prädikat ($predicate).
checkConsistency %private	$predicateCatalogue element() $redefinitions element()*	Überprüft die Konsistenz der angegebenen Prädikat-Neudefinitionen ($redefinitions) gegenüber dem angegebenen Referenzkatalog für Prädikate ($predicateCatalogue).

Tabelle 31: Funktionen des Moduls predicateRedefinition

Die Funktionen des Moduls **`analysisGraphCustomization`** sind für die Anwendung einer angegebenen Anpassung auf den von dieser referenzierten Analysegraphen im Referenzkatalog für Analysegraphen verantwortlich und sind in Tabelle 32 abgebildet. Die öffentliche Funktion **`customize`** prüft zuerst die Konsistenz der Anpassung gegenüber dem referenzierten Analysegraphen und ruft dafür die private Funktion **`checkConsistency`** auf. Diese überprüft, ob nur verfügbare Analysesituationen und Navigationsschritte über die Anpassung abgewählt wurden und ob nur im Referenzkatalog für Analysesituationen erfasste Analysesituationen über die Anpassung hinzugefügt wurden.

In Bezug auf die Navigationsschritte prüft die Funktion **`checkConsistency`**, ob alle Analysesituationen des angepassten Analysegraphen durch einen Navigationsschritt verbunden sind und ob die bei den Navigationsschritten angegebenen Analysesituationen auch nach der Anpassung noch vorhanden sind. Außerdem wird überprüft, ob innerhalb der Navigationsschritte nur in BIRD gültige Navigationsoperationen aus der in Tabelle 1 angeführten Liste verwendet wurden. Stellt die Funktion eine Konsistenzverletzung fest, wird eine entsprechende Fehlermeldung ausgegeben und die Verarbeitung abgebrochen.

Nach erfolgreicher Konsistenzprüfung erstellt die Funktion **`customize`** den angepassten Analysegraphen. Diesem werden alle nicht abgewählten Analysesituationen und Navigationsschritte des referenzierten Analysegraphen aus dem Referenzkatalog für Analysegraphen und die über die Anpassung hinzugefügten Analysesituationen und Navigationsschritte hinzugefügt. Dabei werden eventuelle Neudefinitionen von Navigationsschritten aus der Anpassung berücksichtigt.

Funktion	Parameter	Beschreibung
customize	$database xs:string $graphCustomization element()	Wendet die angegebene Anpassung für einen Analysegraphen auf den entsprechenden Analysegraphen im Referenzkatalog für Analysegraphen der angegebenen BaseX-Datenbank ($database) an.
checkConsistency %private	$database xs:string $graphCustomization element()	Überprüft die angegebene Anpassung für einen Analysegraphen auf ihre Konsistenz gegenüber dem entsprechenden Analysegraphen im Referenzkatalog für Analysegraphen der angegebenen BaseX-Datenbank ($database).

Tabelle 32: Funktionen des Moduls analysisGraphCustomization

4.4 Modul projectGeneration und von diesem verwendete Module

Die Funktion `createProject` des Moduls `projectGeneration` erweitert die über die Funktion `createFiles` erstellten Dateien und wird über die gleichnamige Funktion im Modul `referenceModeling` aufgerufen. Wie in Tabelle 33 ersichtlich, verwendet sie eine private Funktion `insertSchemas`.

Die Funktion `createProject` wendet für jede Analysesituation des angepassten Analysegraphen die gespeicherte Anpassung an uns speichert die so angepasste Analysesituation. Ebenso werden die gespeicherten Anpassungen der multidimensionalen Modelle der referenzierten Faktklassen angewendet und die angepassten multidimensionalen Modelle gespeichert.

Die angepassten multidimensionalen Modelle werden außerdem unter Verwendung der über die Funktion `createFiles` angelegten UUIDs als Faktklassendateien für das angepasste Indyco-Projekt gespeichert. Mit Hilfe der privaten Funktion `insertSchemas` wird rekursiv ermittelt, welche Faktklassen von der Indyco-Projektdatei referenziert werden müssen. Diese private Funktion ist auch für das Einfügen der entsprechenden Elemente in die Indyco-Projektdatei verantwortlich.

Neben der Funktion `insertSchemas` ruft die Funktion `createProject` auch die öffentlichen Funktionen der Module `measureCatalogueParsing` und `referenceCatalogueParsing` auf. Diese wiederholen für die neu definierten berechneten Kennzahlen und Prädikate in den über die Funktion `createFiles` erstellten angepassten Referenzkatalogen das Einfügen der Subelemente für die benötigten Kennzahlen, Attribute, Levels und Hierarchien.

Funktion	Parameter	Beschreibung
createProject	$database xs:string	Erzeugt für die Erstellung des angepassten Indyco- Projekts und des ausführbaren Analysegraphen notwendige Dateien in der angegebenen BaseX-Datenbank ($database).
insertSchemas %private	$database xs:string $modelCustomizations element()* $string xs:string	Ergänzt auf Basis der angegebenen Anpassungen für multidimensionale Modelle ($modelCustomizations) in der Indyco-Projektdatei der angegebenen BaseX-Datenbank ($database) Verweise auf die Faktklassendateien.

Tabelle 33: Funktionen des Moduls projectGeneration

Die in Tabelle 34 beschriebenen Funktionen des Moduls `modelCustomization` sind für die Anwendung einer Anpassung auf ein multidimensionales Referenzmodell verantwortlich. Konkret wird über die für eine angegebene Analysesituation gespeicherte Anpassung auf Basis des referenzierten multidimensionalen Referenzmodells ein neues, angepasstes multidimensionales Modell erstellt. Sowohl das multidimensionale Referenzmodell als auch später das angepasste Modell haben dabei die in Liste 2 in Abschnitt 3.1.2 beschriebene Struktur einer Indyco-Faktklassendatei.

Die Anwendung der Anpassung wird von der öffentlichen Funktion `customize` verwaltet. Diese verwendet wie in Tabelle 34 ersichtlich mehrere private Funktionen, deren Struktur an dieser Stelle aufgrund ihrer Vielzahl nicht im Detail erläutert wird. Zu Beginn prüft die Funktion `customize` über die private Funktion `checkConsistency` die Konsistenz der Anpassung gegenüber dem referenzierten multidimensionalen Referenzmodell. Wurde diese verletzt, wird eine entsprechende Fehlermeldung ausgegeben und die Verarbeitung abgebrochen.

Nach der erfolgreichen Konsistenzprüfung fügt die Funktion `customize` dem Graph-Element des angepassten multidimensionalen Modells neben den nicht über die Anpassung abgewählten Kennzahlen des multidimensionalen Referenzmodells auch für jede über die Anpassung hinzugefügte Basiskennzahl ein entsprechendes Kennzahl-Element hinzu. Dafür wird die private Funktion `buildMeasure` verwendet.

Funktion	Parameter	Beschreibung
customize	$database xs:string $situation element()	Wendet die für die angegebene Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database) hinterlegte Anpassung für ein multidimensionales Modell auf die Faktklasse, auf die diese verweist, an.
buildMeasure %private	$measure element()	Erzeugt für die angegebene Kennzahl ($measure) ein Kennzahl-Element mit allen entsprechenden Subelementen.
buildLevel %private	$level element()	Erzeugt für den angegebenen Level ($level) ein Knoten-Element mit allen entsprechenden Subelementen.
buildAttribute %private	$attribute element()	Erzeugt für das angegebene Attribut ($attribute) ein Knoten-Element mit allen entsprechenden Subelementen.
buildLevelEdge %private	$level element() $superLevel element()	Erzeugt ein Kanten-Element mit allen entsprechenden Subelementen, das den angegebenen Level ($level) mit dem angegebenen Superlevel ($superLevel) verbindet.
buildAttributeEdge %private	$level element() $attribute element()	Erzeugt ein Kanten-Element mit allen entsprechenden Subelementen, das den angegebenen Level ($level) mit dem angegebenen Attribut ($attribute) verbindet.
buildSourceEdge %private	$level element() $graph element()	Erzeugt ein Kanten-Element, das den angegebenen Level ($level) mit dem Faktknoten des angegebenen Graphen ($graph) verbindet.
buildHierarchy %private	$hierarchy element()	Erzeugt für die angegebene Hierarchie ($hierarchy) ein Hierarchie-Element.
buildItem %private	$measure element() $hierarchy element()	Erzeugt für die Kombination der angegebenen Kennzahl ($measure) und der angegebenen Hierarchie ($hierarchy) ein Item-Element.
getSourceNode %private	$edge element() $deselectedLevels element()*	Ermittelt rekursiv den Ausgangsknoten der angegebenen Kante ($edge) unter Berücksichtigung der angegebenen Liste von abgewählten Levels ($deselectedLevels).
getHierarchy %private	$node element()	Ermittelt die Hierarchie des angegebenen Level- oder Attributknotens ($node).
getHierarchyNodes %private	$hierarchy xs:string $graph element()	Ermittelt die Level- und Attributknoten des angegebenen Graphen ($graph), die der angegebenen Hierarchie ($hierarchy) angehören.
getDeselectedLevels %private	$hierarchy xs:string $deselectedLevels element()*	Ermittelt aus der angegebenen Liste von abgewählten Levels ($deselectedLevels) diejenigen, die der angegebenen Hierarchie ($hierarchy) angehören.
getAddedLevels %private	$hierarchy xs:string $hierarchyCatalogue element()	Ermittelt aus dem angegebenen Referenzkatalog für Hierarchien ($hierarchyCatalogue) die Levels, die der angegebenen Hierarchie ($hierarchy) angehören.
checkConsistency %private	$database xs:string $situation xs:string	Überprüft die für die angegebene Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database) hinterlegte Anpassung für ein multidimensionales Modell auf deren Konsistenz.

Tabelle 34: Funktionen des Moduls modelCustomization

Die privaten Funktionen **buildLevel** und **buildAttribute** dienen dem Aufbau von Knoten-Elementen (Nodes) für Levels und Attribute. Solche werden für alle nicht über die Anpassung abgewählten Levels und Attribute des multidimensionalen Referenzmodells erstellt. Bei der Bestimmung der abgewählten Levels und Attribute werden auch jene berücksichtigt, die nicht direkt, sondern über die Abwahl einer ganzen Hierarchie abgewählt wurden. Die Bestimmung dieser wird von den privaten Funktionen **getHierarchy** und **getHierarchyNodes** unterstützt.

Zusätzlich zu den nicht abgewählten Levels und Attributen werden auch für die Levels und Attribute der über die Anpassung dem multidimensionalen Modell hinzugefügten Hierarchien entsprechende Knoten-Elemente erstellt. Die Bestimmung der abgewählten beziehungsweise neu hinzugefügten Levels übernehmen dabei die privaten Funktionen **getDeselectedLevels** und **getAddedLevels**.

Im nächsten Schritt fügt die Funktion **customize** dem Graphen des angepassten multidimensionalen Modells die nötigen Kanten-Elemente (Edges) hinzu. Für die Bildung von Kanten zwischen Levels und deren Superlevels wird die private Funktion **buildLevelEdge** verwendet, für die Bildung von Kanten zwischen Levels und deren Attributen die private Funktion **buildAttributeEdge**. Die private Funktion **buildSourceEdge** ist für das Bilden von Kanten zwischen dem Faktknoten des Graphen und den Basislevels der Hierarchien verantwortlich.

Auch beim Einfügen von Kanten ist es notwendig, über die Anpassung aus dem multidimensionalen Referenzmodell abgewählte Levels und Attribute zu berücksichtigen. Die private Funktion **getSourceNode** ist dafür verantwortlich, für eine Kante, deren Ausgangsknoten entfernt wurde, rekursiv den neuen Ausgangsknoten im angepassten multidimensionalen Modell zu ermitteln.

Nachdem dem Graphen des angepassten multidimensionalen Modells die Kennzahlen, Knoten und Kanten hinzugefügt wurden, werden abschließend noch Elemente für die nicht über die Anpassung abgewählten Hierarchien des multidimensionalen Referenzmodells, sowie für die über die Anpassung hinzugefügten Hierarchien erstellt. Diese Aufgabe bewältigt die private Funktion **buildHierarchy**.

Damit ist der Aufbau des Graphen des angepassten multidimensionalen Modells abgeschlossen. Zusätzlich zum Graphen enthält die angepasste Faktklassendatei noch eine Additivitätsmatrix, der die Funktion **customize** über die private Funktion **buildItem** für jede mögliche Kombination einer Kennzahl und einer Hierarchie ein entsprechendes Element hinzufügt. Wieder werden dabei abgewählte und hinzugefügte Basiskennzahlen und Hierarchien berücksichtigt.

Während die Funktionen des Moduls **modelCustomization** der Anpassung multidimensionaler Referenzmodelle dienen, dienen die in Tabelle 35 abgebildeten Funktionen des Moduls **analysisSituationCustomization** der Anpassung von Analysesituationen aus dem Referenzkatalog für Analysesituationen. Die öffentliche Funktion **customize** wendet dafür die für eine angegebene Anlaysesituation gespeicherte Anpassung unter Berücksichtigung des von der Analysesituation referenzierten multidimensionalen Modells an.

Im ersten Schritt ruft die Funktion **customize** die private Funktion **checkConsistency** zur Prüfung der Konsistenz der Anpassung gegenüber der anzupassenden Analysesituation und deren referenzierter Faktklasse auf. Diese prüft, ob auch nach Anwendung der Anpassung nur verfügbare Kennzahlen, Prädikate und Dimensionen verwendet werden und ob alle für die berechneten Kennzahlen und Prädikate benötigten Kennzahlen, Attribute, Levels und Hierarchien in der angepassten Analysesituation vorhanden sind. Außerdem wird die Struktur der Dimensionen auf ihre Korrektheit überprüft. Stellt die Funktion **checkConsistency** eine Verletzung der Konsistenz fest, wird eine entsprechende Fehlermeldung ausgegeben und die Verarbeitung abgebrochen.

Für die Konsistenzprüfung verwendet die private Funktion **checkConsistency** weitere private Funktionen des Moduls **analysisSituationCustomization**: Die Funktion **getHierarchyNodes** ermittelt für einen Faktklassen-Graphen, welche Knoten des Graphen einer angegebenen Hierarchie angehören, während die Funktion **getCataloguePredicates** Prädikate der Analysesituation mit den entsprechenden Prädikaten eines angegebenen Referenzkatalogs für Prädikate verknüpft. Die Funktionen **getDeselectedLevels** und **getDeselectedAttributes** ermöglichen es, aus einer Liste von abgewählten Levels respektive Attributen einer Anpassung diejenigen zu ermitteln, die einer

angegebenen Hierarchie angehören. Alle dieser erwähnten Funktionen vergleichen zur Ermittlung der Rückgabewerte die Namen der übergebenen Elemente beziehungsweise derer Hierarchien.

Nach erfolgreicher Konsistenzprüfung erstellt die Funktion `customize` aus den nicht abgewählten Kennzahlen und Prädikaten der zu Grunde liegenden Analysesituation aus dem Referenzkatalog und den über die Anpassung hinzugefügten Kennzahlen und Prädikaten eine angepasste Analysesituation. Außerdem werden von der Funktion `customize` die Neudefinitionen der Dimensionen in der angepassten Analysesituation umgesetzt.

Funktion	Parameter	Beschreibung
customize	$database xs:string $situation xs:string $fact element()	Wendet die für die angegebene Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database) hinterlegte Anpassung unter Berücksichtigung der angegebenen Faktklasse ($fact) an.
getHierarchyNodes %private	$hierarchy xs:string $graph element()	Ermittelt die Level- und Attributknoten des angegebenen Graphen ($graph), die der angegebenen Hierarchie ($hierarchy) angehören.
getCataloguePredicates %private	$predicates element()* $predicateCatalogue element()*	Ermittelt das dem angegebenen Prädikat ($predicate) entsprechende Prädikat im angegebenen Referenzkatalog für Prädikate ($predicateCatalogue).
getDeselectedLevels %private	$hierarchy xs:string $deselectedLevels element()*	Ermittelt aus der angegebenen Liste von abgewählten Levels ($deselectedLevels) diejenigen, die der angegebenen Hierarchie ($hierarchy) angehören.
getDeselectedAttributes %private	$hierarchy xs:string $deselectedAttributes element()*	Ermittelt aus der angegebenen Liste von abgewählten Attributen ($deselectedAttributes) diejenigen, die der angegebenen Hierarchie ($hierarchy) angehören.
checkConsistency %private	$database xs:string $situation element() $fact element()	Überprüft die für die angegebene Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database) hinterlegte Anpassung unter Berücksichtigung der angegebenen Faktklasse ($fact) auf ihre Konsistenz.

Tabelle 35: Funktionen des Moduls analysisSituationCustomization

4.5 Modul executableGeneration

Tabelle 36 beschreibt die Funktionen des Moduls `executableGeneration`. Die öffentliche Funktion `createExecutableGraph` dieses Moduls wird von der gleichnamigen Funktion im Modul `referenceModeling` aufgerufen und verwendet mehrere private Funktionen. Auf Basis der über die vorhergehenden Funktionen erstellten Dateien erstellt sie aus dem angepassten Analysegraphen einen ausführbaren Analysegraphen in Form eines SCXML-Schemas. Das Modul `executableGeneration` greift dabei nicht auf Module der Ebene drei zu.

Für jede Analysesituation des angepassten Analysegraphen wird ein Zustand (State) erzeugt. Für jede dieser Analysesituationen wird eine der Analysesituation entsprechende SQL-Abfrage auf die Daten im Data Warehouse erstellt. Dafür werden neben den von den bereits beschriebenen Funktionen erzeugten Dateien auch die über Indyco Builder erzeugen SQL-Befehle zur Erstellung von Star-Schema-Tabellen herangezogen.

Zuerst werden dem `SELECT`-Teil der SQL-Abfrage alle Kennzahlen der Analysesituation hinzugefügt. Dabei wird für jede der Kennzahl zugeordnete Aggregationsoperation ein eigenes `SELECT`-Fragment erstellt. Wurde für eine Kennzahl keine Aggregationsoperation angegeben, wird standardmäßig die Aggregationsoperation `SUM` (Summe) verwendet.

Anschließend werden dem `SELECT`-Teil der SQL-Abfrage die in der Analysesituation angegebenen Granularitätslevels der Dimensionen hinzugefügt. Wurde für eine Dimension kein Granularitätslevel angegeben, wird der der Dimension entsprechende Fremdschlüssel der Faktklasse angegeben. Die Ermittlung der Fremdschlüssel übernimmt dabei die private Funktion `getForeignKeys`. Diese liest die Fremdschlüssel rekursiv aus den von Indyco Builder erstellten Befehlen zur Erstellung von Star-Schema-Tabellen aus.

Für die Erstellung des `FROM`-Teils der SQL-Abfrage verwendet die Funktion `createExecutableGraph` die privaten Funktionen `getComparisonQueries`, `getVariableQueries` und `getMeasureQuery`. Die Funktion `getComparisonQueries` erstellt für jede Vergleichskennzahl der Analysesituation eine SQL-Subabfrage. Für die Ermittlung der Fremdschlüssel der Fakttabelle verwendet sie wie die Funktion `getGranularityLevels` die private Funktion `getForeignKeys`. Für den FROM-Teil der Subabfrage verwendet die Funktion `getComparisonQueries` die ebenfalls private Funktion `getFactJoin`. Diese erstellt auf Basis der von Indyco Builder erstellten SQL-Befehle zur Erstellung von Star-Schema-Tabellen rekursiv die für den Verbund der Dimensionstabellen und der Fakttabelle benötigten SQL-Fragmente. Die private Funktion `getComparisonFactJoin` erfüllt die selbe Funktion für den Vergleichsfakt `ComparisonFact`, der für jede Vergleichskennzahl benötigt wird, und wird ebenfalls von der Funktion `getComparisonQueries` aufgerufen. Anschließend fügt die Funktion `createExecutableGraph` noch die multidimensionalen Prädikate, Dimensionsprädikate und Kennzahlprädikate in den `WHERE`-Teil der Subabfrage ein.

Die private Funktion `getVariableQueries` ist ähnlich aufgebaut wie die Funktion `getComparisonQueries` und ist für das Erstellen der SQL-Subabfragen für diejenigen Kennzahlen verantwortlich, die nicht konkret, sondern als Variable angegeben sind. Der `SELECT`-Teil der Subabfrage ist dynamisch aufgebaut, sodass sowohl für Basiskennzahlen als auch für berechnete Kennzahlen korrekte `SELECT`-Fragmente erstellt werden. Die Fremdschlüssel der Fakttabelle werden auch hier über die private Funktion `getForeignKeys` ermittelt. Für den Verbund der Dimensionstabellen und der Fakttabelle im `FROM`-Teil der Subabfrage wird wie bei der Funktion `getComparisonQueries` die private Funktion `getFactJoin` verwendet. Auch der `FROM`-Teil der Subabfrage ist dynamisch aufgebaut: Nur wenn die Variable durch eine Vergleichskennzahl ersetzt wird, wird das entsprechend benötigte SQL-Fragment für den Verbund eingefügt. Der `WHERE`-Teil dieser Subabfragen ist identisch mit dem der Subabfragen für Vergleichskennzahlen.

Die private Funktion `getMeasureQuery` unterscheidet sich von den beiden vorhergehenden Funktionen insofern, als sie nicht für jede Kennzahl eine eigene SQL-Subabfrage erstellt, sondern alle Kennzahlen in eine Subabfrage zusammenfasst. In dieser Subabfrage werden alle Kennzahlen der Analysesituation abgefragt, die keine Vergleichskennzahlen oder über Variablen angegebene Kennzahlen sind. Die privaten Funktionen `getForeignKeys` und `getFactJoin` werden wie von den beiden oben beschriebenen Funktionen auch von der Funktion `getMeasureQuery` verwendet. Wieder werden im `WHERE`-Teil der Subabfrage alle Prädikate angeführt.

Für den Verbund der Ergebnisse aus den Subabfragen mit den Dimensionstabellen in der Hauptabfrage verwendet die Funktion `createExecutableGraph` noch einmal die private Funktion `getFactJoin`. Anschließend werden dem `GROUP-BY`-Teil der SQL-Abfrage über die private Funktion `getGranularityLevels` die Granularitätslevels aller Dimensionen der Analysesituation hinzugefügt. Wurde für eine Dimension kein Granularitätslevel bestimmt, wird wie schon im `SELECT`-Teil der entsprechende Fremdschlüssel aus der Fakttabelle verwendet. Die Funktion liest dafür über die private Funktion `getForeignKeys` die Fremdschlüssel für alle Dimensionen aus den von Indyco Builder erstellen SQL-Befehlen zur Erstellung von Star-Schema-Tabellen aus und vergleicht deren Dimensionen mit den in der Analysesituation definierten.

Funktion	Parameter	Beschreibung
createExecutableGraph	$database xs:string	Erzeugt aus der angegebenen BaseX-Datenbank ($database) einen ausführbaren Analysegraph als SCXML-Schema und gibt diesen aus.
getDimensionLevels %private	$string xs:string	Ermittelt rekursiv aus dem angegebenen SQL-Fragment einer Dimensionstabelle ($string) alle Levels einer Dimension.
getForeignKeys %private	$fact element() $sql xs:string	Ermittelt rekursiv aus den angegebenen SQL-Befehlen zur Erzeugung von Star-Schema-Tabellen ($sql) die Fremdschlüssel der Fakttabelle der angegebenen Faktklasse ($fact).
getGranularityLevels %private	$database xs:string $situation element() $sql xs:string	Ermittelt aus den angegebenen SQL-Befehlen zur Erzeugung von Star-Schema-Tabellen ($sql) und der angegebenen Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database) die Granularitätslevels aller Dimensionen, sofern angegeben.
getComparisonQueries %private	$database xs:string $situation element()	Erzeugt SQL-Subabfragen für alle Vergleichskennzahlen der angegebenen Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database).
getVariableQueries %private	$database xs:string $situation element()	Erzeugt SQL-Subabfragen für alle als Variable angegebenen Kennzahlen der angegebenen Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database).
getMeasureQuery %private	$database xs:string $situation element()	Erzeugt eine SQL-Subabfrage für alle Kennzahlen, der angegebenen Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database), die keine Vergleichskennzahlen und nicht als Variable angegeben sind.
getFactJoin %private	$fact element() $sql xs:string	Erzeugt rekursiv aus den angegebenen SQL-Befehlen zur Erzeugung von Star-Schema-Tabellen ($sql) den SQL-Verbund für die mit der angegebenen Faktklasse ($fact) verknüpften Dimensionstabellen.
getComparisonFactJoin %private	$fact element() $sql xs:string	Erzeugt rekursiv aus den angegebenen SQL-Befehlen zur Erzeugung von Star-Schema-Tabellen ($sql) den SQL-Vergleichsverbund für Vergleichkennzahlen für die mit der angegebenen Faktklasse ($fact) verknüpften Dimensionstabellen.
getSliceConditions %private	$database xs:string $situation element() $dimension xs:string	Ermittelt die Dimensionsprädikate für die angegebene Dimension ($dimension), die der angegebenen Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database) zugeordnet werden können.
getPredicates %private	$database xs:string $situation element()	Ermittelt die multidimensionalen und Kennzahlprädikate, die der angegebene Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database) zugeordnet werden können
getMeasures %private	$database xs:string $situation element()	Ermittelt die Kennzahlen, die der angegebenen Analysesituation ($situation) in der angegebenen BaseX-Datenbank ($database) zugeordnet werden können.
redefineCalculationRule %private	$fact element() $sql xs:string $calculationRule xs:string	Ersetzt rekursiv die in der angegebenen Berechnungsregel ($calculationRule) vorkommenden Dimensionsbezeichnungen auf Basis der angegebenen SQL-Befehle zur Erzeugung von Star-Schema-Tabellen durch die entsprechenden Fremdschlüssel der Fakttabelle der angegebenen Faktklasse ($fact).

Tabelle 36: Funktionen des Moduls executableGeneration

Neben einer SQL-Abfrage fügt die Funktion `createExecutableGraph` jeder Analysesituation Transitionen (Transitions) für alle Navigationsschritte des angepassten Analysegraphen, die die Analysesituation als Ausgangssituation aufweisen, hinzu. Innerhalb dieser Transitionen werden wiederum alle Navigationsoperationen des jeweiligen Navigationsschrittes erfasst.

Enthalten die Navigationsoperationen `moveToNode` oder `changeGranularity` anstelle eines konkreten Levels eine Variable, wird für diese Variable ein Subelement innerhalb der Navigationsoperation erstellt. Diesem werden über die private Funktion `getDimensionLevels` alle Levels der entsprechenden Dimension hinzugefügt. Die Funktion liest die Levels dabei aus dem von Indyco Builder erstellten SQL-Befehl für die Erstellung der entsprechenden Dimensionstabelle aus.

Bei der Navigationsoperation `moveToNode` besteht auch die Möglichkeit, den Dice-Knoten selbst als Variable anzugeben. Ist dies der Fall, erstellt die Funktion `createExecutableGraph` auch hier ein entsprechendes Subelement. Dieses enthält die SQL-Abfrage zur Abfrage der möglichen Werte aus der entsprechenden, über die von Indyco Builder erstellten SQL-Befehle erstellten Dimensionstabelle.

Wird bei den Navigationsoperationen `addSliceCondition` oder `refocusSliceCondition` das neue beziehungsweise ersetzende Dimensionsprädikat als Variable angegeben, wird innerhalb der Navigationsoperation ebenfalls ein Subelement für diese Variable erstellt. Diesem werden mit Hilfe der privaten Funktion `getSliceConditions` alle für die Analysesituation anwendbaren Dimensionsprädikate hinzugefügt. Dabei wird für jedes Dimensionsprädikat aus dem Referenzkatalog für Prädikate geprüft, ob es sich auf die in der Navigationsoperation angegebene Dimension bezieht und ob alle benötigten Attribute in der angepassten Faktklasse, auf die sich die Analysesituation bezieht, vorhanden sind.

Auch bei der Navigationsoperation `addPredicate` kann das hinzuzufügende Prädikat als Variable angegeben werden. Ist dem so, wird auch in diesem Fall ein entsprechendes Subelement angelegt. Mit Hilfe der privaten Funktion `getPredicates` werden diesem alle für die Analysesituation anwendbaren multidimensionalen Prädikate und Kennzahlprädikate hinzugefügt. Wie bei den Dimensionsprädikaten wird dabei für jedes Prädikat im Referenzkatalog für Prädikate geprüft, ob alle für die Anwendung des Prädikats benötigten Modellelemente in der von der Analysesituation referenzierten angepassten Faktklasse verfügbar sind.

Wird bei der Navigationsoperation `addMeasure` die hinzuzufügende Kennzahl als Variable angegeben, wird ebenfalls ein entsprechendes Subelement in der Navigationsoperation angelegt, welches alle für die Analysesituation anwendbaren Kennzahlen enthält. Die Auswahl der Kennzahlen erfolgt dabei mit Hilfe der Funktion `getMeasures`. Diese ermittelt alle berechneten und Basiskennzahlen der angepassten Faktklasse. Für berechnete Kennzahlen wird anstelle der Berechnungsregel bereits das `SELECT`-Fragment gespeichert, um diese abzufragen. Für Vergleichskennzahlen wird außerdem anstelle der Verbundbedingung bereits das SQL-Fragment für den gesamten Verbund gespeichert. Dafür nutzt die Funktion `getMeasures` die private Funktion `getComparisonFactJoin`. So können die SQL-Subabfragen für die Abfrage von als Variablen angegebenen Kennzahlen dynamisch aufgebaut werden.

Die private Funktion `redefineCalculationRule` wird sowohl von den Funktionen `getComparisonQueries`, `getVariableQueries` und `getMeasureQuery` als auch von der Funktion `getMeasures` verwendet. Sie traversiert rekursiv die angegebene Berechnungsregel oder Verbundbedingung und ersetzt eventuell angegebene Dimensionsnamen ohne Angabe eines Levels durch die entsprechenden Fremdschlüssel der Fakttabelle.

4.6 Fehlermeldungen der Implementierung

Nachdem in den vorhergehenden Abschnitten die in der vorliegenden BIRD-Implementierung umgesetzten XQuery-Funktionen behandelt wurden, soll an dieser Stelle abschließend ein Überblick über die Fehlermeldungen gegeben werden, die aufgrund fehlerhafter Benutzereingaben ausgegeben

werden können. Auf eine genaue Beschreibung der Fehlergründe wird aus Gründen der Übersichtlichkeit verzichtet. Für die Beschreibungen der Fehlermeldungen wird auf Tabelle 37 bis Tabelle 41 innerhalb dieses Abschnitts verwiesen, wobei anzumerken ist, dass in diesen sehr ähnliche Fehlermeldungen sinngemäß zusammengefasst werden.

Tabelle 37 zeigt die Fehlermeldungen, die von den Interface-Funktionen beziehungsweise den von diesen genutzten privaten Funktionen des Moduls **`referenceModeling`** ausgegeben werden. Die erste Fehlermeldung dieser Tabelle vom Typ **`FileError`** wird von der privaten Funktion **`insertFile`** erzeugt, die beiden Fehlermeldungen vom Typ **`NodeError`** und die vom Typ **`ElementError`** von der privaten Funktion **`insertNode`**. Die drei Fehlermeldungen vom Typ **`CustomizationError`** werden von der Funktion **`createFiles`**, die vom Typ **`DatabaseError`** von der Funktion **`createDB`** ausgegeben. Detaillierte Beschreibungen dieser Funktionen finden sich in den entsprechenden vorhergehenden Abschnitten.

Fehlermeldung	Beschreibung
FileError: duplicate file $path	Es wurde versucht, eine Datei am bereits vorhandenen Pfad $path einzufügen.
NodeError: duplicate node $node	Es wurde versucht, einen Knoten mit einer bereits vorhandenen ID $node einzufügen.
NodeError: name of inserted element must not be empty	Es wurde versucht, einen Knoten ohne Bezeichnung einzufügen.
ElementError: Parent element unknown.	Es wurde versucht, einen Knoten in ein nicht vorhandenes Element einzufügen.
CustomizationError: analysis graph customization or analysis graph $graphCustomization unknown.	Es wurde versucht, einen ausführbaren Analysegraphen basierend auf einer nicht vorhandenen Anpassung beziehungsweise einem nicht vorhandenen Analysegraphen $graphCustomization zu erstellen.
CustomizationError: model customization $modelCustomization / analysis situation customization $situationCustomization unknown.	Es wurde versucht, eine nicht vorhandene Anpassung für ein multidimensionales Modell $modelCustomization / eine Analysesituation $situationCustomization anzuwenden.
CustomizationError: duplicate customization for model $model / analysis situation $analysisSituation.	Es wurde versucht, auf ein multidimensionales Modell $model / eine Analysesituation $analysisSituation mehrere Anpassungen anzuwenden.
DatabaseError: duplicate database $database.	Es wurde versucht, eine BaseX-Datenbank zu erstellen, deren Name $database dem einer bereits bestehenden BaseX-Datenbank entspricht.

Tabelle 37: Fehlermeldungen des Moduls referenceModeling

Tabelle 38 zeigt die Fehlermeldungen, die von der privaten Funktion **`checkConsistency`** im Modul **`hierarchyRedefinition`** ausgegeben werden. Dieses Modul der Ebene drei wird vom Modul **`fileGeneration`** der Ebene zwei verwendet. Dieses wiederum wird von der Funktion **`createFiles`** im Interface-Modul **`referenceModeling`** aufgerufen.

Fehlermeldung	Beschreibung
Level/AttributeError: mandatory level $level / attribute $attribute deselected.	Der verpflichtende Level $level / das verpflichtende Attribut $attribute wurde über eine Anpassung eines multidimensionalen Referenzmodells abgewählt.
Level/AttributeError: non-existent level $level / attribute $attribute deselected.	Der nicht vorhandene Level $level / das nicht vorhandene Attribut $attribute wurde über eine Anpassung eines multidimensionalen Referenzmodells abgewählt.

Tabelle 38: Fehlermeldungen des Moduls hierarchyRedefinition

Die in Tabelle 39 dargestellten Fehlermeldungen werden von der privaten Funktion **checkConsistency** des Moduls **measureRedefinition** ausgegeben. Dieses wird vom Modul **fileGeneration** und damit indirekt von der Funktion **createFiles** des Moduls **referenceModeling** verwendet.

Fehlermeldung	Beschreibung
Measure/AttributeError: mandatory measure $overMeasure / attribute $overAttribute not in redefinition of measure $measure.	Die verpflichtende Kennzahl $measure / das verpflichtende Attribut $attribute wurde bei der Neudefinition einer berechneten Kennzahl im Zuge einer Anpassung eines multidimensionalen Referenzmodells nicht verwendet.

Tabelle 39: Fehlermeldungen des Moduls measureRedefinition

Fehlermeldung	Beschreibung
Measure/AttributeError: mandatory measure $measure / attribute $attribute not in redefinition of predicate $predicate.	Die verpflichtende Kennzahl $measure / das verpflichtende Attribut $attribute wurde bei der Neudefinition eines Prädikats im Zuge einer Anpassung eines multidimensionalen Referenzmodells nicht verwendet.

Tabelle 40: Fehlermeldungen des Moduls predicateRedefinition

Fehlermeldung	Beschreibung
SituationError: source / target analysis situation $situation of navigation step $step removed.	Die Ausgangs- / Zielanalysesituation $situation des Navigationsschrittes $step wurde über die Anpassung des Analysegraphen entfernt.
SituationError: source / target analysis situation $situation of navigation step $step non-existent.	Die Ausgangs- / Zielanalysesituation $situation des Navigationsschrittes $step ist nicht vorhanden.
Step/SituationError: duplicate navigation step $step / analysis situation $situation added.	Der Navigationsschritt $step / die Analysesituation $situation wurde über die Anpassung des Analysegraphen hinzugefügt, obwohl bereits vorhanden.
SituationError: non-existent analysis situation $situation added.	Die nicht vorhandene Analysesituation $situation wurde über die Anpassung des Analysegraphen hinzugefügt.
SituationError: non-existent analysis situation $situation in analysis graph.	Die nicht vorhandene Analysesituation $situation ist im Analysegraph vorhanden und wurde nicht über die Anpassung aus diesem abgewählt.
SituationError: no measure in analysis situation $situation.	Die Analysesituation $situation verfügt über keine Kennzahl.
SituationError: no dimension in analysis situation $situation.	Die Analysesituation $situation verfügt über keine Dimension.
SituationError: unconnected analysis situation $situation.	Die Analysesituation $situation ist nicht über einen Navigationsschritt mit den anderen Analysesituationen des Analysegraphen verbunden.
Step/SituationError: non-existent navigation step $step / analysis situation $situation removed.	Der nicht vorhandene Navigationsschritt $step / die nicht vorhandene Analysesituation $situation wurde über die Anpassung des Analysegraphen entfernt.
StepError: non-existent navigation step $step redefined.	Der nicht vorhandene Navigationsschritt $step wurde über die Anpassung des Analysegraphen neu definiert.
OperationError: unknown operation $operation in navigation step $step.	Die Navigationsoperation $operation des Navigationsschrittes $step ist unbekannt.

Tabelle 41: Fehlermeldungen des Moduls analysisGraphCustomization

Die Fehlermeldungen der privaten Funktion **checkConsistency** des Moduls **predicateRedefinition** in Tabelle 40 entsprechen denen der gleichnamigen Funktion im Modul **measureRedefinition**,

allerdings nicht in Bezug auf Kennzahlen, sondern auf Prädikate. Wie das Modul **measureRedefinition** wird auch das Modul **predicateRedefinition** vom Modul **fileGeneration** und damit indirekt von der Funktion **createFiles** des Moduls **referenceModeling** aufgerufen.

Tabelle 41 zeigt die Fehlermeldungen, die von der privaten Funktion **checkConsistency** des Moduls **analysisGraphCustomization** ausgegeben werden. Wie die Module **hierarchyRedefinition**, **measureRedefinition** und **predicateRedefinition** wird auch das Modul **analysisGraphCustomization** von den Funktionen des Moduls **fileGeneration** aufgerufen, welches wiederum von der Interface-Funktion **createFiles** des Moduls **referenceModeling** verwendet wird.

Fehlermeldung	Beschreibung
Hierarchy/Measure/Level/AttributeError: mandatory hierarchy $hierarchy / measure $measure / level $level / attribute $attribute deselected.	Die verpflichtende Hierarchie $hierarchy / Kennzahl $measure / der verpflichtende Level $level / das verpflichtende Attribut $attribute wurde über eine Anpassung eines multidimensionalen Referenzmodells abgewählt.
Hierarchy/Measure/Level/AttributeError: non-existent hierarchy $hierarchy / measure $measure / level $level / attribute $attribute deselected.	Die nicht vorhandene Hierarchie $hierarchy / Kennzahl $measure / der nicht vorhandene Level $level / das nicht vorhandene Attribut $attribute wurde über eine Anpassung eines multidimensionalen Referenzmodells abgewählt.
Hierarchy/MeasureError: non-existent hierarchy $hierarchy / measure $measure added.	Die nicht vorhandene Hierarchie $hierarchy / Kennzahl $measure aus dem Referenzkatalog wurden über eine Anpassung eines multidimensionalen Referenzmodells hinzugefügt.
Hierarchy/MeasureError: duplicate hierarchy $hierarchy / measure $measure added.	Die Hierarchie $hierarchy / Kennzahl $measure wurde über eine Anpassung eines multidimensionalen Referenzmodells hinzugefügt, obwohl bereits vorhanden.
HierarchyError: multiple base levels in hierarchy $hierarchy.	Die Hierarchie $hierarchy besitzt mehrere Basislevels.
HierarchyError: no base level in hierarchy $hierarchy.	Die Hierarchie $hierarchy besitzt keinen Basislevel.
Measure/PredicateError: non-existent measure $measure / predicate $predicate redefined.	Die nicht vorhandene Kennzahl $measure / das nicht vorhandene Prädikat $predicate wurde über eine Anpassung eines multidimensionalen Referenzmodells neu definiert.
Hierarchy/Measure/Level/AttributeError: hierarchy $hierarchy / measure $measure / level $level / attribute $attribute needed for calculation deselected.	Die Hierarchie $hierarchy / Kennzahl $measure der Level $level / das Attribut $attribute wurde über eine Anpassung eines multidimensionalen Referenzmodells abgewählt, obwohl für eine berechnete Kennzahl oder ein Prädikat benötigt.
Hierarchy/LevelError: hierarchy $hierarchy / level $level needed for join deselected.	Die Hierarchie $hierarchy der Level $level wurde über eine Anpassung eines multidimensionalen Referenzmodells abgewählt, obwohl für eine Vergleichskennzahl benötigt.
Hierarchy/Measure/Level/AttributeError: hierarchy $hierarchy / measure $measure / level $level / attribute $attribute needed for calculation non-existent.	Die nicht vorhandene Hierarchie $hierarchy / Kennzahl $measure / der nicht vorhandene Level $level / das nicht vorhandene Attribut $attribute wird für eine berechnete Kennzahl benötigt.
Hierarchy/LevelError: hierarchy $hierarchy / level $level needed for join non-existent.	Die nicht vorhandene Hierarchie $hierarchy der nicht vorhandene Level $level wird für eine Vergleichskennzahl benötigt.

Tabelle 42: Fehlermeldungen des Moduls modelCustomization

Tabelle 42 beschreibt die Fehlermeldungen, die von der privaten Funktion **`checkConsistency`** des Moduls **`modelCustomization`** erzeugt werden. Das Modul **`modelCustomization`** auf Ebene drei wird von den Funktionen des Moduls **`projectGeneration`** auf Ebene zwei aufgerufen. Dieses wiederum wird von der Interface-Funktion **`createProject`** des Moduls **`referenceModeling`** verwendet.

Fehlermeldung	Beschreibung
MeasureError: deselected measure $measure added.	Die über eine Anpassung für ein multidimensionales Referenzmodell abgewählte Kennzahl $measure wurde über eine Anpassung einer referenzierenden Analysesituation hinzugefügt.
Measure/PredicateError: non-existent measure $measure / predicate $predicate added.	Die nicht vorhandene Kennzahl $measure / das nicht vorhandene Attribut $attribute wurde über eine Anpassung einer Analysesituation hinzugefügt.
Measure/PredicateError: duplicate measure $measure / predicate $predicate added.	Die Kennzahl $measure / das Prädikat $predicate wurde über eine Anpassung einer Analysesituation hinzugefügt, obwohl bereits vorhanden.
Measure/PredicateError: non-existent measure $measure / predicate $predicate deselected.	Die nicht vorhandene Kennzahl $measure / das nicht vorhandene Attribut $attribute wurde über eine Anpassung einer Analysesituation abgewählt.
MeasureError: deselected measure $measure in anaysis situation.	Die über eine Anpassung für ein multidimensionales Referenzmodell abgewählte Kennzahl $measure ist in einer Analysesituation vorhanden und wurde nicht über eine Anpassung aus dieser abgewählt.
Measure/PredicateError: non-existent measure $measure / predicate $predicate in analysis situation.	Die nicht vorhandene Kennzahl $measure / das nicht vorhandene Attribut $attribute ist in einer Analysesituation vorhanden und wurde nicht über eine Anpassung aus dieser abgewählt.
DimensionError: deselected dimension $dimension redefined.	Die über eine Anpassung für ein multidimensionales Referenzmodell abgewählte Dimension $dimension wurde über eine Anpassung einer Analysesituation neu definiert.
DimensionError: non-existent dimension $dimension redefined.	Die nicht vorhandene Dimension $dimension wurde über eine Anpassung einer Analysesituation neu definiert.
DimensionError: deselected dimension $dimension in analysis situation.	Die über eine Anpassung für ein multidimensionales Referenzmodell abgewählte Dimension $dimension ist in einer Analysesituation vorhanden und wurde nicht über eine Anpassung aus dieser abgewählt.
DimensionError: non-existent dimension $dimension in analysis situation.	Die nicht vorhandene Dimension $dimension ist in einer Analysesituation vorhanden und wurde nicht über eine Anpassung aus dieser abgewählt.
LevelError: granularity / dice level $level does not match dimension $dimension.	Der Granularitätslevel / Dice-Level $level ist nicht in der Dimension $dimension vorhanden.
LevelError: fixed granularity / dice level $level redefined.	Der unveränderliche Granularitätslevel / Dice-Level $level wurde über eine Anpassung einer Analysesituation verändert.
PredicateError: slice condition $condition does not match dimension $dimension.	Die Slice-Bedingung $condition wurde einer Dimension $dimension zugeordnet, die nicht der Hierarchie der Slice-Bedingung entspricht.
Hierarchy/Measure/AttributeError: hierarchy $hierarchy / measure $measure / attribute $attribute needed for calculation deselected.	Die Hierarchie $hierarchy / Kennzahl $measure / das Attribut $attribute wurde über eine Anpassung eines multidimensionalen Modells abgewählt, obwohl für ein Prädikat benötigt.
Hierarchy/Measure/AttributeError: hierarchy $hierarchy / measure $measure / attribute $attribute needed for calculation non-existent.	Die nicht vorhandene Hierarchie $hierarchy / Kennzahl $measure / das nicht vorhandene Attribut $attribute wird für ein Prädikat benötigt.

Tabelle 43: Fehlermeldungen des Moduls analysisSituationCustomization

Die in Tabelle 43 dargestellten Fehlermeldungen sind die der privaten Funktion **checkConsistency** im Modul **situationCustomization**. Auch dieses wird über das Modul **projectGeneration** und damit indirekt über die Funktion **createProject** des Interface-Moduls **referenceModeling** aufgerufen.

Wird eine Fehlermeldung aus einem Modul der Ebene drei oder eine der in Tabelle 37 beschriebenen Fehlermeldungen vom Typ **CustomizationError** ausgegeben, ist es empfehlenswert, die in Abschnitt 3.2.10 beschriebene Funktion **clearExecutableGraph** auszuführen und erneut mit der Ausführung der Funktionen **parseCatalogues**, **createFiles**, **createProject** und **createExecutableGraph** zu beginnen, um die BaseX-Datenbank konsistent zu halten.

5 Fazit

Referenzmodelle erleichtern den Umgang mit Data Warehouses, besonders in Bezug auf die Einführung von Data Warehouses und Business Intelligence Software im Allgemeinen. Speziell kleine und mittlere Unternehmen können so bei der Einführung und Verwendung solcher Software unterstützt werden.

Der von Schütz u. a. (2016) entwickelte BIRD-Ansatz stellt ein leichtgewichtiges Rahmenwerk für Referenzmodellierung im Data Warehousing zur Verfügung, welches im Zuge der vorliegenden Arbeit unter Verwendung verbreiteter und leicht zugänglicher sowie leicht verständlicher Technologien umgesetzt wurde. Diese Tatsache fördert zusätzlich den einfachen Einstieg in die Arbeit mit Data Warehouses. Neben Referenzmodellen für multidimensionale Modelle bieten BIRD und die vorliegende Implementierung mit der Verwendung von Referenzkatalogen für Hierarchien, Kennzahlen, Prädikate, Analysesituationen und Analysegraphen umfangreiche Möglichkeiten zur Personalisierung von multidimensionalen Modellen sowie den darauf basierenden Analysesituationen und Analysegraphen.

Schütz u. a. (2016) schlagen als Zielsetzung zukünftiger, auf der vorliegenden Arbeit basierender Arbeiten unter anderem Referenzmodellierung für Verfahrensrichtlinien sowie die Verwendung von Referenzmodellen für Geschäftssituationen für die automatisierte Erstellung von Analysegraphen im Bereich des aktiven Data Warehousing („Active Data Warehousing“) vor.

Literaturverzeichnis

Acerbis R, Bongio A, Brambilla M, Butti S (2015) Model-Driven Development Based on OMG's IFML with WebRatio Web and Mobile Platform. In: Cimiano P, Frasincar F, Houben G-J, Schwabe D (Hrsg) Engineering the Web in the Big Data Era. Springer International Publishing, S 605–608

Aligon J, Gallinucci E, Golfarelli M, et al (2015) A collaborative filtering approach for recommending OLAP sessions. Decision Support Systems 69:20–30. doi: 10.1016/j.dss.2014.11.003

Battaglia A, Golfarelli M, Rizzi S (2011) QBX: A CASE Tool for Data Mart Design. In: De Troyer O, Bauzer Medeiros C, Billen R, u. a. (Hrsg) Advances in Conceptual Modeling. Recent Developments and New Directions. Springer Berlin Heidelberg, S 358–363

Becker J, Delfmann P, Knackstedt R (2007) Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In: Becker J, Delfmann P (Hrsg) Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models. Physica-Verlag HD, S 27–58

Becker J, Knackstedt R (2004) Referenzmodellierung im Data-Warehousing — State-of-the-Art und konfigurative Ansätze für die Fachkonzeption. Wirtschaftsinformatik 46:39–49. doi: 10.1007/BF03250994

Becker J, Knackstedt R, Eggert M, Fleischer S (2012) Fachkonzeptionelle Modellierung von Berichtspflichten in Finanzaufsicht und Verwaltung mit dem H2-Toolset. In: Von Lucke J, Geiger CP, Kaiser S, u. a. (Hrsg) Auf dem Weg zu einer offenen, smarten und vernetzten Verwaltungskultur. Gesellschaft für Informatik, Bonn, S 83–94

Ceri S, Brambilla M, Fraternali P (2009) The History of WebML Lessons Learned from 10 Years of Model-Driven Development of Web Applications. In: Borgida AT, Chaudhri VK, Giorgini P, Yu ES (Hrsg) Conceptual Modeling: Foundations and Applications. Springer Berlin Heidelberg, S 273–292

Chan APC, Chan APL (2004) Key performance indicators for measuring construction success. Benchmarking: An International Journal 11:203–221. doi: 10.1108/14635770410532624

Diamantini C, Genga L, Potena D, Storti E (2014) Collaborative Building of an Ontology of Key Performance Indicators. In: Meersman R, Panetto H, Dillon T, u. a. (Hrsg) On the Move to Meaningful Internet Systems: OTM 2014 Conferences. Springer Berlin Heidelberg, S 148–165

Fettke P, Vom Brocke J (2013) Referenzmodell. In: Enzyklopädie der Wirtschaftsinformatik. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Softwarearchitektur/Wiederverwendung-von-Softwarebausteinen/Referenzmodell. Accessed 2 Jän 2016

Goeken M, Knackstedt R (2007) Multidimensional Reference Models for Data Warehouse Development. In: ICEIS 2007: Proceedings of the Ninth International Conference on Enterprise Information Systems, Funchal, Madeira, Portugal, June 12-16, 2007. Institute for Systems and Technologies of Information, Control and Communication, Universidade da Madeira, S 347–354

Golfarelli M, Maio D, Rizzi S (1998) The Dimensional Fact Model: A Conceptual Model for Data Warehouses. International Journal of Cooperative Information Systems 07:215–247. doi: 10.1142/S0218843098000118

Golfarelli M, Rizzi S (1998) A Methodological Framework for Data Warehouse Design. In: DOLAP '98 Proceedings of the 1st ACM international workshop on Data warehousing and OLAP. ACM New York, S 3–9

Heer J, Shneiderman B (2012) Interactive Dynamics for Visual Analysis: A taxonomy of tools that support the fluent and flexible use of visualizations. Communications of the ACM 55:45–54. doi: 10.1145/2133416.2146416

Horschitz C (2016) Prototypische Implementierung eines Werkzeuges zur Modellierung und Ausführung von Business Intelligence Analysgraphengraphen (unter Begutachtung). Johannes Kepler Universität Linz

Knackstedt R, Klose K (2005) Configurative reference model-based development of data warehouse systems. In: Proceedings of the 16th information resources management association conference 2005. S 32–39

Knackstedt R, Seidel S, Janiesch C (2006) Konfigurative Referenzmodellierung zur Fachkonzeption von Data Warehouse-Systemen mit dem H2-Toolset. In: Schelp J, Winter R, Frank U, u. a. (Hrsg) DW2006. Integration, Informationslogistik und Architektur. Gesellschaft für Informatik, S 61–80

Lane P, Potineni P (2014) Oracle Database Data Warehousing Guide, 12c Release 1 (12.1). Oracle Corporation

Malinowski E, Zimányi E (2006) Hierarchies in a multidimensional model: From conceptual modeling to logical representation. Data and Knowledge Engineering 59:348–377. doi: 10.1016/j.datak.2005.08.003

Maté A, Trujillo J, Mylopoulos J (2012) Conceptualizing and Specifying Key Performance Indicators in Business Strategy Models. In: Atzeni P, Cheung D, Ram S (Hrsg) Conceptual Modeling. 31st International Conference ER 2012. Proceedings. Springer Berlin Heidelberg, S 282–291

Neuböck T, Neumayr B, Rossgatterer T, et al (2012) Multi-dimensional Navigation Modeling Using BI Analysis Graphs. In: Castano S, Vassiliadis P, Lakshmanan L, Lee ML (Hrsg) Advances in Conceptual Modeling, ER 2012 Workshops. Springer Berlin Heidelberg, S 162–171

Neuböck T, Schrefl M (2015) Modelling Knowledge about Data Analysis Processes in Manufacturing. In: Proceedings of the 15th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2015). Elsevier Ltd., S 277–282

Neumayr B, Schütz C, Schrefl M (2013) Semantic Enrichment of OLAP Cubes: Multi-dimensional Ontologies and Their Representation in SQL and OWL. In: Meersman R, Panetto H, Dillon T, u. a. (Hrsg) On the Move to Meaningful Internet Systems: OTM 2013 Workshops. Springer Berlin Heidelberg, S 624–641

Olszak CM, Ziemba E (2012) Critical Success Factors for Implementing Business Intelligence Systems in Small and Medium Enterprises on the Example of Upper Silesia, Poland. Interdisciplinary Journal of Information, Knowledge, and Management 7:129–150. doi: 15551229

Pokorný J, Sokolowsky P (1999) A conceptual modelling perspective for data warehouses. In: Scheer A-W, Nüttgens M (Hrsg) Electronic Business Engineering / 4. Internationale Tagung Wirtschaftsinformatik 1999. Physica-Verlag Heidelberg, S 665–684

Recker J, Mendling J, Van Der Aalst W, Rosemann M (2006) Model-Driven Enterprise Systems Configuration. In: Dubois E, Pohl K (Hrsg) 18th International Conference on Advanced Information Systems Engineering (CAiSE 2006). Springer Berlin Heidelberg, S 369–383

Romero O, Marcel P, Abelló A, Peralta V (2011) Describing Analytical Sessions Using a Multidimensional Algebra. In: Cuzzocrea A, Dayal U (Hrsg) 13th International Conference on Data Warehousing and Knowledge Discovery - DaWaK 2011. Springer Berlin Heidelberg, S 224–239

Rosemann M, Van Der Aalst W (2007) A configurable reference modelling language. Information Systems 32:1–23. doi: 10.1016/j.is.2005.05.003

Rumpe B (2011) Modellierung mit UML. Springer Heidelberg Dordrecht London New York

Sapia C (1999) On Modeling and Predicting Query Behavior in OLAP Systems. In: Gatziu S, Jeusfeld M, Staudt M, Vassiliou Y (Hrsg) Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99).

Scheer A-W, Nüttgens M (2000) ARIS Architecture and Reference Models for Business Process Management. In: Van Der Aalst W (Hrsg) Business Process Management. Springer Berlin Heidelberg, S 376–389

Scholz P, Schieder C, Kurze C, et al (2010) Benefits and Challenges of Business Intelligence Adoption in Small and Medium-Sized Enterprises. In: 18th European Conference on Information Systems.

Schütz C, Schrefl M (2014) Customization of Domain-Specific Reference Models for Data

Warehouses. In: 2014 IEEE 18th International Enterprise Distributed Object Computing Conference (EDOC). IEEE Computer Society, S 61–70

Schütz C, Schrefl M, Neumayr B, Neuböck T (2016) Customization of Domain-Specific Reference Models for Data Warehouses and Analysis Processes. International Journal of Cooperative Information Systems (Arbeitstitel; unter Begutachtung, akzeptiert mit geringfügigen Änderungen)

Steiner D, Neumayr B, Schrefl M (2015) Judgement and Analysis Rules for Ontology-driven Comparative Data Analysis in Data Warehouses. In: Köhler H, Saeki M (Hrsg) Proceedings of the 11th Asica-Pacific Conference on Conceptual Modelling (APCCM 2015), Sydney, Austrialia, 27-30 January 2015. Australian Computer Society, Inc., S 71–80

Strecker S, Frank U, Heise D, Kattenstroth H (2011) MetricM: a modeling method in support of the reflective design and use of performance measurement systems. Information Systems and e-Business Management 10:241–276. doi: 10.1007/s10257-011-0172-6

Thalhammer T, Schrefl M, Mohania M (2001) Active data warehouses: Complementing OLAP with analysis rules. Data and Knowledge Engineering 39:241–269. doi: 10.1016/S0169-023X(01)00042-8

Thomas O (2006) Understanding the Term Reference Model in Information Systems Research: History, Literature Analysis and Explanation. In: BPM 2005 Workshops. Springer Berlin Heidelberg, S 484–496

Thomas O, Scheer A-W (2006) Tool Support for the Collaborative Design of Reference Models - A Business Engineering Perspective. In: Proceedings of the 39th Hawaii International Conference on System Sciences 2006. IEEE Computer Society,

Trujillo J, Gomez J, Palomar M (2000) Modeling the Behavior of OLAP Applications Using an UML Compliant Approach. In: Yakhno T (Hrsg) ADVIS. Springer Berlin Heidelberg, S 14–23

Trujillo J, Palomar M, Gomez J, Song IY (2001) Designing Data Warehouses with OO Conceptual Models. Computer 34:66–75. doi: 10.1109/2.970579

Vom Brocke J (2009) Design Principles for Reference Modeling: Reusing Information Models by Means of Aggregation, Specialisation, Instantiation, and Analogy. In: Halpin T, Krogstie J, Proper E (Hrsg) Innovations in Information Systems Modeling: Methods and Best Practices. IGI Global, S 47–76

W3C (2015) State Chart XML (SCXML): State Machine Notation for Control Abstraction - W3C Recommendation 1 September 2015. http://www.w3.org/TR/2015/REC-scxml-20150901/. Accessed 2 Jän 2016