Transformation mehrdimensionaler Datenmodelle[♦]

Michael Hahne

cundus AG

Data Warehousing, OLAP und Data Mining gewinnen als Basistechnologien zur Versorgung betrieblicher Entscheidungsträger durch eine stärkere Ausrichtung der Informationsverarbeitung auf analyseorientierte und strategische Fragestellungen an Bedeutung. Die in solchen analyseorientierten Informationssystemen vorgehaltenen Daten sind im allgemeinen mehrdimensional aufgebaut und stellen an die Modellierung eine besondere Anforderung. Eine werkzeuggestützte Modellierung basiert auf der Möglichkeit, ausgehend von der semantischen Modellebene losgelöst von möglichen Zielplattformen auf Ebene des Fachkonzeptes abzubilden und dieses Modell dann auf die weiteren Ebenen herunterzubrechen sowie durch geeignete Transformationsverfahren in verschiedenne Datenbanksystemen Strukturen zu generieren. In dem vorliegenden Artikel wird auf der logischen Datenmodell-Ebene ein Metamodell definiert und als zentraler Punkt der Transformation positioniert. Semantische Datenmodelle können dann in dieses Metamodell auf logischer Ebene übertragen werden, von wo aus danach die Transformation in verschiedene zielsystemspezifische Modelle ebenfalls auf der logischen Modellebene erfolgen kann. Daran schließt sich die Generierung durch Übertragung in entsprechende Anweisungen der jeweiligen Datendefinitionssprache an. Als Grundlage des gesamten Transformationsmechanismus werden Techniken aus dem Bereich des Compilerbaus eingesetzt, die Modelle werden als formale Sprachen definiert.

♦ Veröffentlicht als: *Hahne, Michael: Transformation mehrdimensionaler Datenmodelle, in: von Maur, Eitel; Winter, Robert (Hrsg.): Vom Data Warehouse zum Corporate Knowledge Center, Physica-Verlag, Heidelberg, 2002, S. 399-420.* Siehe auch http://www.springer.de/cgi/svcat/search_book.pl?isbn=3-7908-1536-5.

1 Einleitung

Aufgrund einer zunehmenden Ausrichtung der Informationsverarbeitung auf analyseorientierte und strategische Anforderungen erhält die Versorgung von Fach- und Führungskräften mit adäquaten analyserelevanten Informationen in der Informationstechnologie einen neuen Stellenwert. Data Warehousing, OLAP und Data Mining werden zunehmend zu Basistechnologien, die damit die Grundlagen für Anwendungen des Knowledge Management, Customer Relationship Management sowie E-Commerce bilden und zunehmende Relevanz in der Praxis erhalten. Diese Systeme basieren zu einem wesentlichen Teil auf dem mehrdimensionalen Paradigma der in ihnen verwendeten Daten, die eine angemessene Möglichkeit der dauerhaften Ablage benötigen. Problemadäquate Datenbanktechnologien zur Speicherung mehrdimensionaler Daten sind hier gefordert. In diesem Artikel liegt der Fokus auf dem logischen Datenmodell, das eingebettet in die drei Ebenen der semantischen, logischen und physischen Modellierung eine zentrale Rolle einnimmt.

Eine spezifische Datenbank-Technologie basiert auf dem Relationenmodell. Die spezielle Form zur Abbildung mehrdimensionaler Datenstrukturen in diesem Modell ist unter dem Namen Star Schema eingeführt und mittlerweile in einer Vielzahl von Ausprägungen verfügbar. Andere Datenbanktechnologien sind speziell auf die Mehrdimensionalität hin ausgerichtet und arbeiten mit proprietären Modellen. Diese werden oft auch als OLAP-Datenbanken bezeichnet. Für das Star Schema liegt eine formale Spezifikation des zugrunde liegenden logischen Datenmodells vor. Diese ist zugleich die Grundlage zur Klassifikation von Star Schemata und der Definition eines Metamodells zur Beschreibung solcher Strukturen. Für die mehrdimensionalen Datenbanksysteme gibt es keine einheitliche logische Modellbasis.

Beim Aufbau analyseorientierter Informationssysteme erfolgt die Modellierung über die verschiedenen Ebenen der Modellierung hinweg bis zum konkreten System auf physischer Ebene. Eine allgemeine Forderung ist dabei, dass die Modelle auf einer allgemeinen Ebene des Fachkonzeptes losgelöst von möglichen Systemen der Realisierung sind und erst in einer Generierungsphase eine Transformation in ein Zielsystem erfolgt. Dabei sollen aus einem semantischen Modell heraus über das logische Metamodell in verschiedenen möglichen Datenbanksystemen Schemata generiert werden.

Zunächst erfolgt in Abschnitt 2 eine kurze Darstellung der Modellierungsebenen und der Möglichkeiten der Modelltransformation über diese Ebenen hinweg. Im anknüpfenden Abschnitt 3 wird ein Metamodell als zentraler Ausgangspunkt möglicher Transformationen aufgebaut und als formale Sprache dargestellt. In Abschnitt 4 erfolgt die Darstellung der Transformation auf Basis formaler Sprachen.

2 Modelltransformation und Datenbank-Generierung

Ausgehend von der Darstellung der verschiedenen Ebenen der Modellierung im ersten Abschnitt folgt im zweiten Abschnitt die Aufarbeitung der grundsätzlichen Transformationsmöglichkeiten zwischen den einzelnen Ebenen und innerhalb der logischen Modellebene.

2.1 Ebenen der Datenmodellierung

Eine geläufige Strukturierung des Modellierungsvorganges ist in Abb. 2.1 dargestellt. Danach werden die Ebenen der semantischen, logischen und physischen Datenmodellierung unterschieden.[1] Ein Datenmodell soll die Bedeutung und Repräsentation von Daten beschreiben.

[1] Vgl. Lockemann/Radermacher (1990).

Abb. 2.1 Modell-Ebenen

Der Realwelt am nächsten ist dabei die semantische Ebene. In der zweiten Ebene, in der die logische Modellierung erfolgt, sind die Modelle ebenfalls noch unabhängig von der physischen Repräsentation, richten sich jedoch an der für die Speicherung einzusetzenden Datenbanktechnologie aus. Demzufolge spricht man hier auch von konzeptioneller Modellierung.[2] Ein solches Modell heißt auch Datenbankschema. Bei relationalen Datenbanksystemen wird vom Relationenmodell gesprochen.

2.2 Transformationsmöglichkeiten

In einem ersten Schritt soll ein Metamodell als Bestandteil im Gesamtkontext der mehrdimensionalen Datenmodellierung betrachtet werden. Der Modellierungsprozess setzt im allgemeinen auf semantischer Modellebene an, auf der es viele verschiedene Modellierungsansätze gibt. Eine geeignete graphische Repräsentationsform steht dabei im Vordergrund, da die intuitive Nachvollziehbarkeit hier sehr wichtig ist. Aus diesen semantischen Datenmodellen wird das logische Datenmodell abgeleitet. Dies kann ein Metamodell sein, welches als ein Ausgangspunkt auf logischer Modellebene fungiert. Von diesem Metamodell aus erfolgt eine Transformation in verschiedene Zieldatenbanksysteme. Diese können verschiedenen Paradigmen genügen, unter denen die eines Datenbanksystems auf Basis des Relationenmodells nur eine Möglichkeit darstellt. Andere Zielsysteme können verschiedene OLAP-Plattformen adressieren. Dieser Zusammenhang ist in Abb. 2.2 im Gesamten dargestellt.

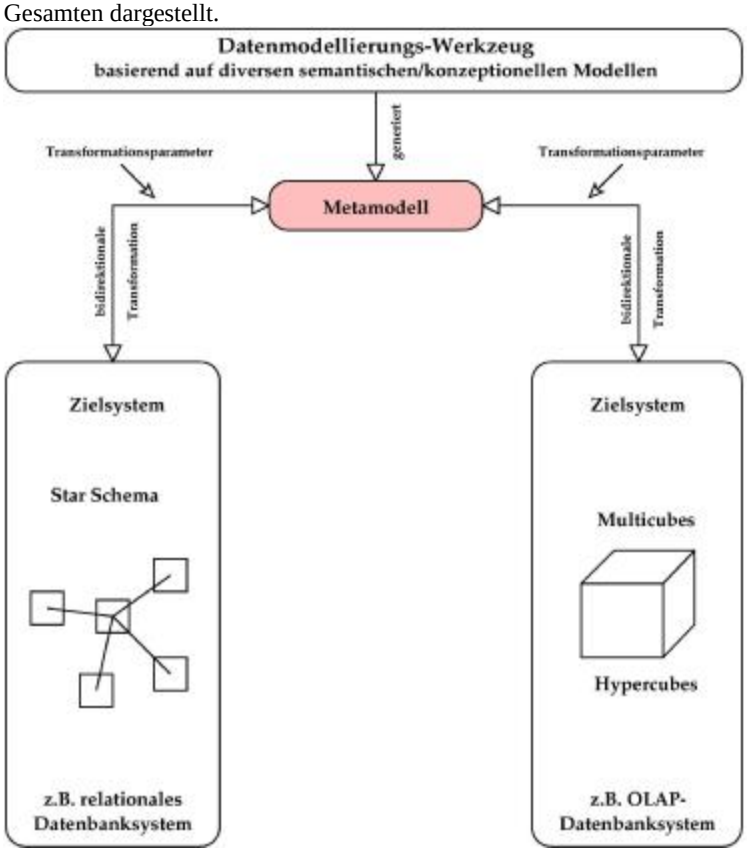

Abb. 2.2 Integration in einen Modellierungsgesamtkontext

[2] Vgl. Gabriel/Röhrs (1995).

Die Transformation ist im allgemeinen nicht ohne Informationsverlust darstellbar, da nicht in allen Modellen alle Aspekte eines zentralen Metamodells abgebildet sind. Auch ist es denkbar, dass ein Modell ein Konstrukt beinhaltet, das so im Metamodell keine Berücksichtigung findet. Desweiteren bedarf dieser Transformationsschritt weitere Parameter, wie im Detail zu konvertieren ist. Für den Fall der Konvertierung in das Star Schema hinein werden z. B. die Informationen benötigt, in welche Variante transformiert werden soll. Dieser Fall der Transformation wird in den weiteren Ausführungen exemplarisch aufgezeigt.

3 Mehrdimensionales Metamodell als formale Sprache

Die Möglichkeiten des Einsatzes eines Datenmodells hängen auch davon ab, welche Sprachen zur Verfügung stehen, um mit diesem Modell arbeiten zu können. Der erste wichtige Punkt ist bereits beim Modellierungsprozess entscheidend, da hierbei die Definition der Schemata erfolgt. Dieser Schritt kann in zwei Ebenen unterteilt werden. Zum einen muss das Datenmodell selbst eine Syntax bzw. eine Sprache zur Verfügung stellen, mit der die Schemata definierbar sind. Zum anderen erfolgt die Modellierung meistens mit Unterstützung eines Modellierungswerkzeuges, so dass die formale Sprache zur Definition von Datenbank-Schemata nicht primär anwenderfreundlich gestaltet sein muss, sondern ihren Fokus auf konsequente Unterstützung der Möglichkeiten und der vollständigen Abbildung des Modells richten kann.

Sprachen zur Schema-Festlegung sind dadurch gekennzeichnet, dass sie auch Konstrukte zur Erweiterung und Änderung bestehender Schemata zur Verfügung stellen. Davon unterscheidet sich ein Metamodell, das ausschließlich die aktuellen Schema-Definitionen beinhaltet. Transformationsalgorithmen setzen nicht an der Datendefinitionssprache selbst an, sondern greifen auf das Metamodell zurück, das meistens als Metadaten- oder Data-Dictionary integraler Bestandteil eines Datenbanksystems ist.

3.1 Formale Sprachen

Sprachen werden über einem Alphabet, d. h. einer endlichen nichtleeren Menge von Zeichen, definiert. Ein Wort über einem beliebigen Alphabet ist eine endliche Folge von Zeichen des Alphabets. Weiter wird das leere Wort bestehend aus keinem Zeichen definiert. Eine *formale Sprache* ist eine beliebige Teilmenge der Menge aller Worte über einem festgelegten Alphabet.

Sprachen sind im allgemeinen unendliche Objekte, so dass zu deren algorithmischer Handhabung eine endliche Beschreibung notwendig ist. Diese Forderung ist für die Gesamtheit aller Sprachen nicht erfüllbar, jedoch für die Menge aller rekursiv aufzählbaren Sprachen, die dadurch gekennzeichnet sind, dass sie durch Grammatiken beschrieben werden können. In der von Chomsky 1959 aufgestellten Sprachhierarchie ist eine Klassifizierung verschiedener Sprachen gegeben. Eine Grammatik dient also der formalen Beschreibung einer Sprache, d. h. sie bestimmt, welche Worte zu dieser Sprache gehören. Im folgenden werden oft Grammatiken wie hier nur durch Angabe der Produktionsregeln definiert, d. h. festgelegten Regeln, wie aus gewissen Grundbausteinen komplette Worte der Sprache abgeleitet werden können.

Anhand der Produktionsregeln einer Grammatik können ausgehend von dem Startsymbol die Worte der erzeugten Sprache abgeleitet werden. Wie kann aber über ein vorliegendes Wort über dem gegebenen Alphabet entschieden werden, ob es zu dieser Sprache gehört oder nicht? Diese Aufgabe wurde im Rahmen der Theorie des Compilerbaus ausführlich diskutiert. Allgemein übersetzen Compiler von einer Sprache in eine andere. Die Übersetzung von der einen Sprache in die andere erfolgt dabei im allgemeinen in den drei Phasen der lexikalischen, syntaktischen und semantischen Analyse, welche nicht notwendigerweise sequentiell abgearbeitet werden müssen:

1. In der Phase der *lexikalischen Analyse* wird der Quelltext analysiert, d. h. es wird überprüft, ob er ein Wort über dem Ausgangsalphabet ist, und wird dabei in elementare Ausdrücke (sog. Tokens) transformiert, damit eine spätere Weiterverarbeitung leichter möglich ist.

2. Die Phase der *syntaktischen Analyse* überprüft das von der lexikalischen Analyse als Wort über dem Ausgangsalphabet erkannte Wort auf die syntaktische Korrektheit.[3]
3. In der Phase der *semantischen Analyse* werden die als syntaktisch korrekt erkannten Worte auf ihre semantische Stimmigkeit hin überprüft.

Diese Mechanismen formaler Sprachen sind Grundlage der Definition eines Metamodells zur Darstellung mehrdimensionaler Strukturen im nächsten Abschnitt, auf das in den weiteren Ausführungen zur Transformation wieder Bezug genommen wird.

3.2 Definition eines Mehrdimensionalen Metamodells

Auf Basis einer festgelegten Datendefinitionssprache für ein Zielsystem ist das Erstellen und Modifizieren von mehrdimensionalen Datenbank-Schemata möglich. [4]

Dabei ist ein Skript eine Sammlung von *data definition statements* (DDL-Anweisungen). Bei graphischen Werkzeugen, welche den entsprechenden DDL-Code automatisch erzeugen, liegt die Anforderung an die Benutzerfreundlichkeit bei dem Werkzeug und nicht bei der erzeugten Sprache, so dass die Sprachdefinition auf formale Klarheit und theoretische Konsequenz hin ausgerichtet sein kann. Die aktuelle Struktur eines Schemas ergibt sich nicht nur aus einem einzelnen Skript, sondern kann sich auch aus mehreren einzelnen Skripten zusammengesetzt ergeben. In einem Metadatensystem erfolgte die Ablage dieser Struktur nicht zusammen mit der Historie der Entstehung auf Basis einzelner Anweisungen, sonder als einzelnes Skript. Für Transformationsalgorithmen ist das die geeignetere Ausgangsbasis.

Dies ist auch die Intention bei der Definition eines Metamodells für mehrdimensionale Datenstrukturen. Darüber hinaus können sich noch vielfältige weitere Anforderungen auf Ebene der Metadaten ergeben, die als Erweiterungen des Metamodells mit abbildbar sind und damit zu einer formalen Beschreibung eines Repositories für Datenbanksysteme auf Basis logischer Datenmodelle führen.

Für das Metamodell wird eine Notation in Backus Naur Form (BNF) zur Definition genutzt. In der Darstellung der Produktionsregeln in BNF kann der Ausdruck links des Trennzeichens (zwei Doppelpunkte gefolgt von dem Gleichzeichen) ersetzt werden durch den Teil auf der rechten Seite. Optionale Ersetzungsmöglichkeiten werden durch den senkrechten Strich dargestellt. Ausdrücke, die nicht weiter ersetzt werden können und damit elementare Grundbausteine der beschriebenen Sprache darstellen, unterscheiden sich von den anderen Bausteinen des Aufbaus durch die Notation. Ersetzbare Zeichen, also genau die auf den linken Seiten auftauchenden Begriffe, sind in spitzen Klammern notiert.

Die Darstellung des Metamodells erfolgt in diesem Abschnitt nur auf Ebene der reinen Strukturen, die dynamischen Komponenten und Integritätsbedingungen sind genauso wie weitere für ein Metadatenbanksystem relevante Dinge einer Erweiterung des Metamodells überlassen. An erster Stelle zur Strukturierung im Metamodell steht der Begriff der Datenbank. In jeder Datenbank sind dann die Dimensionen und Kuben definiert, zwischen denen eine m:n-Beziehung besteht. Damit die Regeln in BNF nicht zu lang werden, wurde der Ansatz verfolgt, jede Regel in kleinere Teilregeln zu zergliedern.

```
<meta model>    ::= DATABASES: <database list> | ;
<database list> ::= DATABASE: <database> ; |
                    DATABASE: <database> ; <database list>
<database>      ::= <dbname> <datatypes> <cubes> <dimensions>
<dbname>        ::= <id>
```

[3] Ganz allgemein erfolgt in der Syntaxanalyse keine Prüfung der semantischen Stimmigkeit. Bei Programmiersprachen erfolgt z. B. die Prüfung von Datentypenverträglichkeit ebenfalls nicht in der Phase der Syntaxüberprüfung.

[4] Von Datenbanksystemen wird meistens eine Datendefinitionssprache (*data definition language, DDL*), mit der die Schemata festgelegt werden, eine Datenmanipulationssprache (*data manipulation language, DML*) und eine Abfragesprache (*query language, QL*) zur Verfügung gestellt, vgl. Brodie (1984), S. 20f. In der Literatur werden für Anfragesprachen an mehrdimensionale Modelle oft SQL-ähnliche Sprachen definiert. Eine einheitliche Abfragesprache ist MDX (Multidimensional Expression), die unter anderem beim Microsoft SQL-Server und allgemein bei Schnittstellen auf Basis der Spezifikation für OLE DB für OLAP zum Einsatz kommt.

Eine Liste der verfügbaren Datentypen ist insofern Bestandteil des Metamodells, als die Gesamtheit der verfügbaren Datentypen als erweiterbar oder frei definierbar angesehen werden muss. Die vefügbare Liste der Datentypen ist im Rahmen der Darstellung des Metamodells als konstant vorausgesetzt.

```
<datatype>      ::= integer boolean char double
<datatypes>     ::= DATATYPES: <datatype list>
<datatype list> ::= <datatype> | <datatype> , <datatype list>
```

In einem Beispielskript könnte dieser erste Skripteil folgendermassen aufgebaut sein:

```
DATABASES:
  DATABASE: [Marketing]
    DATATYPES: integer,boolean,char,double
```

Gegenstand des Beispiels ist eine mehrdimensionale Struktur, die eine Anwendung im Marketing repräsentiert. Der Bezeichner (id) der Datenbank ist, wie alle Bezeichner im Beispiel, in eckigen Klammern geschrieben. Die Skriptteile, auf deren Basis die Würfel und Dimensionen definiert sind, fehlen an dieser Stelle noch.

Das Konstrukt eines Würfels basiert auf einer Liste der zugrunde liegenden Dimensionen und dem Datentyp der Zellinhalte. Die Definition eines Würfels erfolgt für eine spezielle Datenbank, die sich aus der Position im Skript des Metamodells direkt ergibt.

```
<cubes>           ::= CUBES: <cube list>
<cube list>       ::= <cube> | <cube>, <cube list>
<cube>            ::= CUBE: <cubename>
                      DATATYPE: <datatype>
                      REFERENCED DIMENSIONS: <referenced dims>
<referenced dims> ::= <dimname> | <dimname> , <referenced dims>
<cubename>        ::= <id>
<dimname>         ::= <id>
```

Die Definition der diversen Datentypen folgt gängigen Konventionen und ist weitgehend an Naur (1963) und die Definition der Datentypen in der Programmiersprache C angelehnt.[5] Das mögliche Beispielskript erweiternd könnte der Bereich der Würfelfestlegung durch den folgen Teil eines Skriptes definiert werden:

```
CUBES:
  CUBE: [Umsatz]
    DATATYPE: float
    REFERENCED DIMENSIONS: [Zeit],[Vertriebsweg],[Produkt]
```

Der betrachtete Datenwürfel beinhaltet Werte zu auf einem bestimmten Vertriebsweg getätigten Produkt-Umsätzen. Die wesentlichen Strukturbestandteile sind durch die Dimensionen festgelegt. Dimensionen bestehen im einfachsten Fall nur aus einer Liste von Knoten und Pfeilen. Eine Erweiterung dieser einfachen Sichtweise ergibt sich durch die Markierungen auf Ebene der Knoten und Pfeile.

```
<dimensions>     ::= DIMENSIONS: <dimension list>
<dimension list> ::= <dimension> | <dimension> , <dimension list>
<dimension>      ::= DIMENSION: DATAYPE: <datatype>
                     NODES: <nodes> EDGES: <edges>
                     LABELS: <labels>
<nodes>          ::= <node list>
<edges>          ::= <edge list> | -
<labels>         ::= <labellist> | -
<labellist>      ::= <label> | <label> , <labellist>
<label>          ::= LABEL: <labelname>
                     LABELVALUES NODES <labelndef> |
                     LABEL: <labelname>
                     LABELVALUES EDGES <labeledef> |
                     LABEL: <labelname>
                     LABELVALUES NODES <labelndef>
                     LABELVALUES EDGES <labeledef>
```

[5] Vgl. Naur (1963), Seite 4f.

Die Definition der Ausdrücke *labelndef* und *labeledef* ergibt sich wie auch die Auflösung der von *node list* und *edge list* aus den folgenden Regeln:

```
<labeldeflist>  ::= <labeldef> | <labeldef> <labeldeflist>
<labeldef>      ::= no label | label nodes <labelndef> <labelname> |
                    label edges <labeledef> <labelname> |
                    label nodes <labelndef> label edges <labeledef> <labelname>
<labelndef>     ::= ( <labelnlist> )
<labeledef>     ::= ( <labelelist> )
<labelnlist>    ::= <node> : <label> | <node> : <label> , <labelnlist>
<labelelist>    ::= <edge> : <label> | <edge> : <label> , <labelelist>
<label>         ::= <const>
<labelname>     ::= named <id>
```

Der Rest des möglichen Skriptes auf Basis des definierten Metamodells könnte demzufolge auszugsweise sein:[6]

```
DIMENSIONS:
  DIMENSION: [Zeit]
    DATATYPE: char
    NODES: '1999',
           '2000',
           '2001',
           'Alle Jahre',
    EDGES: ('Alle Jahre','1999'),
           ('Alle Jahre','2000'),
           ('Alle Jahre','2001'),
    LABELS: -
  DIMENSION: [Vertriebsweg]
    DATATYPE: char
    NODES: 'Partner',
           'Katalog',
           'E-Shop'
    EDGES: -
    LABELS: -
  DIMENSION: [Produkt]
    DATATYPE: char
    NODES: 'F99-12',
           'F99-7',
           'F99-21S',
           'F99-13H',
    EDGES: ('Rennräder','F99-12'),
           ('Rennräder','F99-7'),
           ('Rennräder','F99-21S'),
           ('Rennräder','F99-13H'),
    LABELS:
      LABEL: [Partition]
        LABELVALUES NODES: 'F99-12':'Produkt',
                           'F99-7':'Produkt',
                           'F99-21S':'Produkt',
                           'F99-13H':'Produkt',
                           'Rennräder':'Warengruppe',
```

Dieses Skript beschreibt das Schema. Durch das soweit syntaktisch spezifizierte Metamodell wird bereits implizit eine kontextfreie Sprache definiert, wobei noch angegeben werden muss, welche Ausdrücke ersetzbar sind und was das Startzeichen ist. Die Syntaxregeln in BNF sind gerade die Produktionsregeln. Auf Basis dieses syntaktisch definierten Metamodells erfolgt im nächsten Abschnitt die Darstellung der Transformation in Strukturen auf Basis verschiedener Zielsysteme.

[6] Dieses Beispiel ist Hahne (2002) entnommen.

4 Formalsprachliche Transformation

Bereits bei der Definition des Metamodells wurden die Möglichkeiten der Definition von formalen Sprachen genutzt. Die Definition der Sprachsyntax erfolgte in Backus Naur Form (BNF), auf deren Basis eine kontextfreie Sprache festgelegt wird. Ausgangspunkt für Transformationsmöglichkeiten auf Basis formaler Sprachen ist in jedem Fall zunächst ein *Parser*, der die Syntax der formalen Ausgangssprache erkennt. Für das beschriebene Metamodell wird in Abschnitt 4.2 ein solcher Parser definiert. Die Transformation vom Metamodell in das Star Schema steht in Abschnitt 4.3 im Vordergrund. Anschließend erfolgt eine zusammenfassende Darstellung der Möglichkeiten der Transformation in Abschnitt 4.4.

4.1 Formale Sprachen und Parser

Ein Parser für die Analyse von Metamodell-Skripten kann mit Techniken aus dem Bereich des Compilerbaus erzeugt werden. Die Generierung von Compilern lässt sich gut automatisieren, da dieses Gebiet theoretisch sehr fundiert ist und die eigentliche Aufgabe eines Compilers sehr gut in Phasen aufteilbar ist. Die Grundaufgabe besteht darin, einen Quelltext unter Ausgabe von eventuellen Fehlermeldungen in einen Objektcode bzw. in eine Zielsprache zu übersetzen.

In der ersten Phase erfolgt das Lesen des Quellprogrammes und dessen Zerlegung in handliche Einheiten, den *Token*. Kommentare und überflüssige Leerzeichen können dabei schon wegfallen. Diese Phase ist die lexikalische Analyse. Anschließend erfolgt die Gruppierung dieser Token in einer Form, dass die nachfolgenden Phasen diese besser verarbeiten können, sowie die Prüfung der syntaktischen Korrektheit. Daher heißt diese Phase auch Syntaxanalyse und dieses Programm dazu Parser.[7] Erst anschließend werden diese gruppierten Token von ihrer Semantik her überprüft und zum Zielprogramm hin verarbeitet.

Für die Phasen der lexikalischen und der syntaktischen Analyse gibt es die Hilfsmittel *lex*[8] (lexical analyzer) und *yacc*[9] (Akronym für *yet another compiler compiler*), die in den BELL Laboratories als Werkzeuge zur Compiler-Generierung entstanden und seitdem, in immer weiterentwickelten und verbesserten Versionen, zur Grundausstattung von UNIX-Systemen gehören. Im Rahmen des GNU-Projektes[10] entstanden public-domain Versionen von lex und yacc, der lexikalische Analysator *flex* und der parser-Generator *bison*.[11] Diese sind weitestgehend kompatibel zu den Original-Werkzeugen, aber in vielen Fällen etwas leistungsfähiger und flexibler. Beide Werkzeuge werden für die Erzeugung der Parser in diesem Artikel verwendet.

Der grundsätzliche Aufbau eines flex-Skriptes ist folgender:[12]

```
  ...definition section...
%%
  ...rules section...
%%
  ...user subroutines...
```

Der Definitionsteil eines lex-Skriptes (*definition section*) beinhaltet die Hilfsdefinitionen, wie z. B. Abkürzungen für reguläre Ausdrücke sowie durch %{ und %} geklammerter C-Code, der direkt in den von lex generierten C-

---

[7] Der Parser ist demzufolge genau das Teilprogramm, das die Syntaxanalyse durchführt. Im weiteren Sinne wird aber auch das Programm der Kombination aus der lexikalischen Analyse und der Syntaxanalyse als Parser bezeichnet.

[8] Vgl. Ambrosch (1990).

[9] Vgl. Ambrosch (1991).

[10] Das GNU-Projekt wurde 1984 mit dem Ziel der Entwicklung eines UNIX-ähnlichen Betriebssystems als freie Software gestartet. GNU ist ein rekursives Akronym für „GNU's not UNIX“ und ist ein Projekt der *Free Software Foundation*, die den Begriff *freie Software* genau definiert. Informationen zu GNU und der Free Software Foundation sind im World Wide Web unter der URL www.gnu.org und www.gnu.org/fsf verfügbar.

[11] Vgl. Mason (1990), S. 277ff. Eine ausführliche Beschreibung der Werkzeuge *flex* und *bison* sowie Abgrenzung gegenüber *lex* und *yacc* bietet die Online-Dokumentation der Werkzeuge.

[12] Vgl. Mason (1990), S. 147ff.

Code integriert wird. Im Übersetzungsteil (*rules section*) befinden sich die lexikalischen Konstrukte, die vom Programm erkannt werden sollen. Dieser Bereich ist durch zeilenweise Angaben der Form eines regulären Ausdrucks gefolgt von geklammertem C-Code aufgebaut. Erkennt lex diesen regulären Ausdruck in der Eingabe, wird der entsprechende C-Code ausgeführt. In Zusammenarbeit mit bison als Generator für den syntaktischen Analysator erfolgt der Aufruf der lexikalischen Analyse immer dann automatisch durch den Parser, wenn ein neues Token von der Eingabe benötigt wird.

Der grundsätzliche Aufbau eines bison-Skriptes entspricht dem eines flex-Skriptes.[13] Im Deklarationsteil kann wie bei einem lex-Skript auch durch %{ und %} geklammerter C-Code eingefügt werden, der direkt in das generierte Programm eingefügt wird. Des weiteren erfolgt in diesem Bereich die Festlegung der Token-Konstanten und einiger weiterer Einstellungen zur Steuerung der Syntaxanalyse wie etwa die Definition des Datentypen für Token mit Rückgabewerten.

Das „Herzstück“ eines bison-Skriptes sind die Regeln (*rules section*). Hier ist die zu erkennende Grammatik hinterlegt. Das Erkennen einer Regel führt zu der Abarbeitung des zugeordneten C-Codes. Im dritten Bereich (*user subroutines*) ist wieder C-Code hinterlegbar, der direkt in das generierte C-Programm kopiert wird.

4.2 Parser für das Metamodell

Grundsätzlich ist jeder Parser komplett individuell programmierbar, aber für den Parser des Metamodells sollen die zuvor beschriebenen Techniken eingesetzt werden. Der erste Analyseschritt besteht in dem Aufnehmen der Eingabe und dem Herausfiltern der wichtigen erkannten Wörter, den Token, mit dem Werkzeug flex, die von überflüssigem Fülltext befreit und dann im zweiten Schritt der Analyse auf Basis des Werkzeuges bison auf ihre grammatikalische Korrektheit hin untersucht werden.

In der ersten Phase erfolgt also die Analyse des Quelltextes, in der dieser auf Basis regulärer Ausdrücke in einzelnen Teile zerstückelt wird. Einige abkürzende Definitionen zur Festlegung von regulären Ausdrücken werden in flex im Vorspann mit festgelegt. Dies sind im Fall des Parsers für das Metamodell auszugsweise:

```
blank                       [ \t\n]+
cunsignedinteger            [0-9]+
cinteger                    [+-][0-9]+
```

Ein `blank` bezeichnet also einen beliebig langen leeren Zwischraum. Der zweite Ausdruck bezeichnet eine beliebige Ziffernfolge, die aus mindestens einer Ziffer besteht.[14] Die Aufgabe der lexikalischen Analyse besteht in dem Erkennen der Eingabe-Token. Für einige exemplarische Token ist der Aufbau im Parser der folgende:

```
"database:" {fprintf(yylog,"DATABASE:\n"); return DATABASE;}
"cube:" {fprintf(yylog,"CUBE:\n"); return CUBE;}
```

Beim Erkennen des Token, z. B. des Schlüsselwortes `database:`, wird ein Eintrag in die Logdatei gemacht und eine interne Konstante an die Syntaxanalyse übergeben. Hier wird das Zusammenspiel der beiden Komponenten deutlich. Der Aufruf des Parsers erfolgt mit den entsprechenden Parametern auf Ebene der Syntaxanalyse. Diese fordert immer bei Bedarf vom Lexikalischen Analysator ein neues Eingabe-Token an. Eine spezielle Funktion haben Token mit Wert, die mit als Rückgabewert übergeben werden:

```
{creal} {yylval.doubleval=strtod(yytext,NULL); fprintf(yylog,"%s: flex: float:
%s\n",progname,yytext); return VDOUBLE; }
{cchar} {yylval.charval=strdup(yytext); fprintf(yylog,"%s: flex: char:
%s\n",progname,yytext); return VCHAR; }
```

Diese Wertübergabe basiert auf der folgenden Token-Definition im bison-Skript:

```
%union { char             *charval;
         double           doubleval; }

%token <charval>         VCHAR
%token <doubleval>       VDOUBLE
```

[13] Vgl. Mason (1990), S. 181ff.

[14] Zum Aufbau regulärer Ausdrücke vgl. Mason (1990), S. 28f.

Dies sind die Token mit Wert. Die Definition der Token ohne Wert erfolgt auszugsweise durch:

```
%token DATABASE CUBE DIMENSION
```

Die Nichtterminalzeichen werden ebenfalls alle mit Wert definiert, da in ihnen der analysierte Regeltext gespeichert wird.

```
%type <charval> database , dbname , datatypes , cubes , dimensions
```

Bei den folgenden einfachen Grammatik-Regeln, die in bison formuliert sind, wird deutlich, wie der grundsätzliche Aufbau ist und wie der analysierte Text rekursiv durchgereicht wird. Der Aufbau der Regel ist immer Nichtterminalzeichen der linken Seite gefolgt von dem Teil der rechten Seite der Regel. Der Wert des Nichtterminalzeichens der linken Seite wird mit $$ addressiert, die der rechten Seite von links nach rechts durchnummeriert beginnend bei $1.

```
   'DATABASES:' databaselist
      {
       tc=strdup(maxc);
       strcpy(tc,"DATABASES:\n");
       strcat(tc,$2);
       $$=tc;
      }
 | ';'
      {
       $$=";";
      }
   ;
```

Komplexer ist die Auflösung konstanter Werte, die von flex als Token mit Wert übergeben werden, im Parser. Hier erfolgt die Übernahme der Werte mit spezieller Information, von welchem Datentyp die Konstanten sind, und die Eingabezeichenfolge wird zu einem Wert des Datentyps konvertiert.[15]

Dieser Parser für das Metamodell ist dann die algorithmische Basis für einen Transformationsalgorithmus vom Metamodell in ein anderes Zielmodell. Die nötigen Transformationsschritte für den einen Spezialfall der Transformation vom Metamodell in das Star Schema in der Darstellung des Relationenmodells sind Gegenstand des folgenden Abschnittes.

4.3 Transformation in das Star Schema

Das dargestellte Metamodell kennt im Gegensatz zum Star Schema keine Variantenbildung, denn es gibt die folgenden Bestandteile des Modells:

- Dimensionen bzw. Dimensions-Schemata,
- Kuben und Kuben-Schemata,
- Datenbanken und Datenbank-Schemata sowie
- Markierungen von Dimensionen und des Datenbank-Graphen.

Diese Bestandteile sind der Ausgangspunkt einer Transformation eines Quellmodells auf Basis des definierten Metamodells. Als Beschreibung dieses Quellmodells wird die Darstellung in Form des Metamodells als formale Sprache genutzt.

In diesem Abschnitt wird exemplarisch für die vielen Transformationen und diversen Zielmodelle die Transformation in das Relationenmodell und das Star Schema als Form zur Darstellung mehrdimensionaler Datenstrukturen im Relationenmodell entwickelt.

Das Star Schema ist nicht ein einzelnes Modell, da es in vielen verschiedenen Varianten definiert ist und sehr unterschiedliche Ausprägungen haben kann. Eine Transformation in dieses Zielmodell muss also präzisieren, in

[15] Die Darstellung erfolgt an dieser Stelle sehr kurz gefasst, ausführlich wird der Aufbau eines Parsers für das beschriebene Metamodell in Hahne (2002) beschrieben.

welche explizite Form des Star Schemas hinein konvertiert wird. Diese Transformation ist in Abb. 4.1 im Überblick dargestellt.

Die in Hahne (2002) entwickelte Klassifikation von Star Schema-Strukturen ist eine geeignete Grundlage, die konkrete Ausgestaltung des Zielmodells zu parametrisieren. Bezüglich des Aufbaus von Faktentabellen sind Star Schemata danach differenzierbar, wieviele zentrale Faktentabellen vorhanden sind und wo ggf. die Ablage von Aggregaten erfolgt. Ein wesentliches Unterscheidungskriterium für Star Schema-Varianten ist der Grad der Normalisierung und der Schlüsselbildung der Dimensionstabellen sowie die Modellierung der Dimensionsstrukturen der jeweiligen Dimensionen z. B. in Form von rekursiven Beziehungen oder durch festgelegte Gruppierungsattribute. Weiterhin sind die vorkommenden Strukturen bezüglich des Orts der Ablage von Kennzahlen zu unterscheiden. Üblich ist dabei die Generierung einer Wertspalte in der Faktentabelle für jede Kennzahl.

Ausgangspunkt ist ein Skript des beschriebenen Metamodells, welches z. B. auf Basis eines semantischen Datenmodells generiert wird. Dieses Metamodell wird von einem Algorithmus unter Bezugnahme auf Parameter zur Transformation in das Metamodell des Star Schemas überführt.

Die Pfeile in der graphischen Repräsentation deuten an, dass diese Transformation vom Grundsatz her so angelegt ist, dass auch der umgekehrte Weg möglich ist, in der die Transformation in die andere Richtung durchgeführt wird. Dabei ist aber nicht zu erwarten, dass diese Transformation hin und her ohne Informationsverlust durchführbar ist. Diese Transformation ist nicht bijektiv.

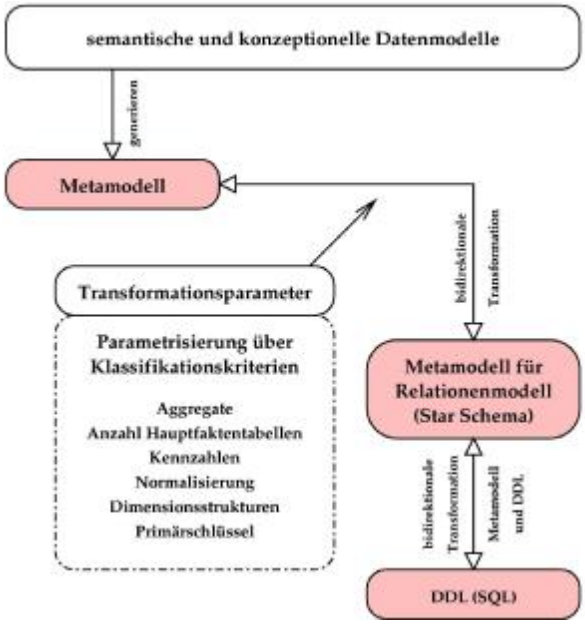

Abb. 4.1 Transformation vom Metamodell in das Star Schema

In einem weiteren Schritt kann dann dieses Metamodell in eine zielplattformspezifische Datendefinitionssprache (DDL) überführt werden. Der Schritt zwischen dem Metamodell und dieser DDL ist im allgemeinen ohne Informationsverlust umkehrbar.

Die konkrete Transformation soll im folgenden für den Fall des klassischen Star Schemas ausführlich dargestellt werden. Für die Transformation wird die folgende Vorgehensweise angewendet:

- Transformation der Dimensionen mit Markierungen
- Transformation der Kuben
- Generierung der Datenbank-Integritätsbedingungen

Schon der erste Punkt führt zu einem Problem, da die beiden Modelle für Dimensionen nicht in der gleichen Form zwischen Schema und Instanz differenzieren. Im Metamodell beinhaltet das Schema schon die konkreten Dimensionselemente als Strukturbeschreibung des Modells, diese sind im Relationenmodell konkrete Ausprägungen (Instanzen) der Relationen-Schemata für die Dimensionstabellen. Von ihrem Wesensgehalt her hat aber auch diese Ausprägung einer Relation strukturellen Charakter. Diese wird in der Transformation aber

ausgelassen, da dies der sauberen Trennung von Schema und Instanz auf logischer Modellebene widerspricht. Diese Komponente eines Transformationsalgorithmus wird als eine Erweiterungsoption offen gehalten, wobei das beschriebene Metamodell für das Star Schema bzw. das Relationenmodell in der beschriebenen Form keine Ausprägungen von Relationen-Schemata kennt und dafür entsprechend erweitert werden müsste.

Zunächst erfolgt also die Transformation der Dimensionen, im Beispielskript sind drei Dimensionen definiert. Die erste hat im Metamodell die folgende Beschreibung:

```
DIMENSION: [Zeit]
  DATATYPE: char
  NODES: '1999',
         '2000',
         '2001',
         'Alle Jahre',
  EDGES: ('Alle Jahre','1999'),
         ('Alle Jahre','2000'),
         ('Alle Jahre','2001'),
  LABELS: -
```

Diese Dimension wird im Relationenmodell im Rahmen des klassischen Star Schemas durch eine Struktur der mit den Attributen Schlüssel, Monat, Quartal, Jahr und Gesamt abgebildet, wobei die Wertebereiche und der Primärschlüssel entsprechend festgelegt sind.

Bei der Transformation ist aus dem Quellmodell auf Basis des Metamodells nicht erkennbar, welche Attributnamen zu bilden sind. Für das eigentliche Dimensionselement ist das ableitbar, für die Ebenen in der Hierarchie auf Basis der Kantendefinitionen ergibt sich lediglich eine Ebenenidentifikation. Für das Generieren der Attribute für die Hierarchie der Dimension braucht der Transformationsalgorithmus teilweise zusätzliche Informationen über den Dimensionstyp, der aber algorithmisch bestimmbar ist.

Die zweite Dimension des Beispiel-Schemas ist die Dimension Vertriebsweg und wird im Metamodell durch ein ähnlich aufgebautes Skriptstück wie im Fall der Zeitdimensions abgebildet. Die verbleibende dritte Dimension im Beispiel hat zusätzlich zu der eigentlichen Definition auch noch eine Markierung, welche die Hierarchie repräsentiert. Im Star Schema korrespondiert das mit einem level-Attribut.

```
DIMENSION: [Produkt]
  DATATYPE: char
  NODES: 'F99-12',
         'F99-7',
         'F99-21S',
         'F99-13H',
  EDGES: ('Rennräder','F99-12'),
         ('Rennräder','F99-7'),
         ('Rennräder','F99-21S'),
         ('Rennräder','F99-13H'),
  LABELS:
    LABEL: [Partition]
      LABELVALUES NODES: 'F99-12':'Produkt',
                         'F99-7':'Produkt',
                         'F99-21S':'Produkt',
                         'F99-13H':'Produkt',
                         'Rennräder':'Warengruppe',
```

In der Darstellung des Relationenmodells ergibt sich die Definition eines Schemas mit den Komponenten Schlüssel, Produkt, Warengruppe, Partition sowie wieder der entsprechenden Festlegung von Wertebereichen und Primärschlüssel. Die Attributnamen könnne in diesem Fall aus dem Metamodell übernommen werden (*label*).

Die Definition des Würfels „Umsatz“ basiert im Metamodell lediglich auf der Angabe des Datentypen und der referenzierten Dimensionen:

```
CUBE: [Umsatz]
  DATATYPE: float
  REFERENCED DIMENSIONS: [Zeit],[Vertriebsweg],[Produkt]
```

Im klassischen Star Schema wird der Würfel durch genau eine Faktentabelle dargestellt, deren Schema die Abhängigkeiten zu den referenzierten Dimensionstabellen berücksichtigen muss. Im Relationenmodell sind dies Inklusionsbedingungen, die im dritten Schritt der Transformation in die Integritätsbedingungen auf Datenbank-Ebene münden.

Das Schema der Faktentabelle im Relationenmodell besteht aus den Attributen des zusammengesetzten Primärschlüssels sowie dem Wertattribut für den Umsatz. Neben die Definition von Wertebereichen und Schlüsseleigenschaften kommt noch die Festlegung der Integritätsbedingungen auf Datenbank-Ebene. Diese sind genau die Inklusionsbedingungen, die sich aus den Fremdschlüsselbeziehungen ergeben.

Für die Transformation wird noch eine eindeutige Korrespondenz zwischen den im Metamodell festgelegten Datentypen und den im Relationenmodell verfügbaren Wertebereichen gefordert.

Die dargestellte Transformation in das klassische Star Schema ist der einfachste Fall unter den möglichen Zielmodellen als Varianten des Star Schemas. Für einen allgemeinen Algorithmus, der verschiedene Star Schema-Varianten generieren kann, sind die Klassifikationskriterien als Parametrisierung die Grundlage für den Algorithmus. Hierbei sind im einzelnen für die Transformation der Dimensionen die folgenden vier Parameter zu beachten:

- Art der Abbildung von Kennzahlen
- Form der Normalisierung
- Zugrunde liegende Dimensionsstruktur
- Ausgestaltung des Primärschlüssels

Die Transformation der Kuben hat zwei Parameter:

- Umgang mit Aggregaten
- Anzahl der Hauptfaktentabellen

Als Eingabe dient ein Skript auf Basis des definierten Metamodells. Für jede zu transformierende Dimension und für jeden Kubus erfolgt die Übersetzung in die Zielstruktur auf Basis der Parameter, wobei die Einstellungen für verschiedene Dimensionen auch unterschiedlich sein können, ebenso für unterschiedliche Kuben.

Das Relationenmodell mit dem Star Schema als Modellierungsgrundlage ist nur eines von mehreren denkbaren Zielmodellen. Im folgenden Abschnitt erfolgt eine allgemeinere Darstellung des Algorithmus zur Transformation in verschiedene Zielmodelle.

4.4 Möglichkeiten der Transformation im Überblick

Die dargestellte Transformation des vorigen Abschnittes ist eine unidirektionale Verbindung von zwei Modellen. Naheliegend ist also eine Verallgemeinerung zu mehreren Knoten und bidirektionalen Transformationsmöglichkeiten. Die verallgemeinerte Transformation soll sich zunächst auf die logische Modellebene beschränken. Ausgangspunkt für die Umformung eines Modells ist das definierte Metamodell als zentrales logisches Datenmodell zur Abbildung mehrdimensionaler Datenstrukturen. Zielmodelle werden ebenfalls durch adäquate Metasprachen repräsentiert, in denen die transformierten Strukturen definiert werden können. Der Kern der Transformation liegt dann in der Übertragung von mehrdimensionalen Strukturen vom zentralen Metamodell in ein anderes Metamodell oder umgekehrt. Die Transformation verschiedener Zielmodelle untereinander ist in diesem Kontext nicht gewünscht und soll ausgeschlossen bleiben.

Die Möglichkeiten der gegenseitigen Umformung der verschiedenen durch jeweilige Metamodelle repräsentierten logischen Datenmodelle ergibt in graphischer Darstellung die Form eines Sternes mit dem dargestellten Metamodell als Zentrum. Für jede einzelne Transformationsrichtung erfolgt die Umsetzung auf Basis individueller Parametrisierungen. Für den Fall des Star Schemas und damit dem Relationenmodell als Zielmodell sind die Klassifikationskriterien die Grundlage für diese Parametrisierung. Der Gesamtzusammenhang der Transformation ist in der folgenden Abb. 4.2 visualisiert. Dort ist das Datenmodell einer OLAP-Datenbank (z. B. MIS-Alea) als weiteres exemplarisches Zielmodell mit angeführt.

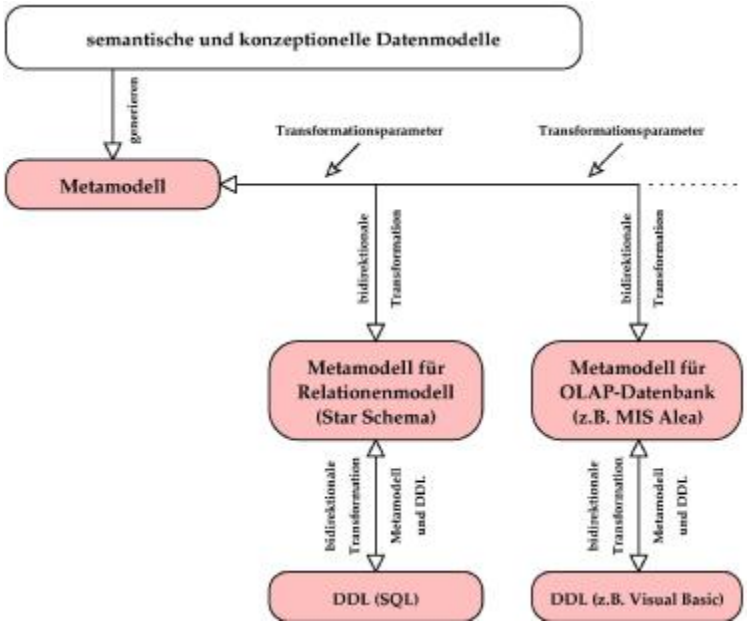

Abb. 4.2 Möglichkeiten der Transformation im Überblick

Als weiterer Schritt im Rahmen eines Transformationsprozesses ist die Generierung, d. h. der Übergang zur physischen Modellebene, zu berücksichtigen. In der Abbildung ist dies durch die weiterführende Übertragung des Metaskriptes in die zielsystemspezifische DDL angedeutet. Im Falle des Relationenmodells sind u. a. alle relationalen Datenbanksysteme eine mögliche Zielplattform. Die dort gängige Datendefinitionssprache SQL ist vom Programm aus dem Metamodell heraus zu generieren. Einige spezifische physische Parameter müssten in der Graphik der Vollständigkeit halber noch als Parameter mit angeführt werden.

Für den Fall der OLAP-Datenbank ist die dort definierte DDL zu verwenden, in dem Fall des Beispiels mit MIS-Alea könnte dies auf Basis von Visual Basic mit dedizierter systemspezifischer DDL-Bibliothek definiert werden. Andere Zielsysteme haben eine eigene ebenfalls proprietäre Schnittstelle, z. B. über eine API (*application programming interface*).

Ein weiterer Aspekt betrifft die semantische Modellierung. In diesem Transformation-Gesamtkonzept gibt es eine Konvertierung zwischen einem semantischen Datenmodell, das seinerseits in einer formalen Sprache auf Metaebene fixiert sein sollte, und dem Metamodell auf der logischen Ebene.

5 Ausblick

Der hier dargestellte Ansatz der Transformation mehrdimensionaler Datenmodelle über ein Metamodell auf logischer Modellebene ist dahingehend erweiterbar, das dargestellte Verfahren in ein Werkzeug zur mehrdimensionalen Modellierung zu integrieren und damit einen Framework zu ermöglichen, der auch weitere Funktionalitäten, wie etwa das automatisches Generieren von Dimensionen auf Basis vorhendener Quellsysteme, beinhaltet.

Die dynamischen Eigenschaften insbesondere von speziellen mehrdimensionalen Datenmodellen sind bei der Betrachtung aussen vor geblieben, um diese ebenfalls zu berücksichtigen ist der dargestellte Ansatz eines Metamodells zu erweitern. Die Unterstützung vielfältiger Zieldatenbanksysteme erfordert dabei insbesondere für die dynamischen Eigenschaften ein komplexe Erweiterung der Transformations-Algorithmen.

6 Literatur

Ambrosch, Wolfgang (1990): Die Compiler-Macher – LEX auf ST und PC, in: c't, o.Jg., Nr. 12, 1990, S. 232-236.

Ambrosch, Wolfgang; Beer, Felix (1991): Die Compiler-Macher – Teil 2: YACC, in: c't, o.Jg., Nr. 2, 1991, S. 222-228.

Brodie, Michael L. (1984): On the developement of data models, in: Brodie, Michael L.; Mylopoulos, John; Schmidt, Joachim (Hrsg.): On conceptual modelling, Springer, New York et al. 1984, S. 19-47.

Gabriel, Roland; Röhrs, Heinz-Peter (1995): Datenbanksysteme: Konzeptionelle Datenmodellierung und Datenbankarchitekturen, 2. Aufl., Springer, Berlin et al. 1995.

Hahne, Michael (2002): Logische Modellierung mehrdimensionaler Datenbanksysteme, Deutscher Universitäts-Verlag, Wiesbaden 2002.

Lockemann, Peter C.; Radermacher, Klaus (1990): Konzepte, Methoden und Modelle zur Datenmodellierung, in: HMD, 1990, S. 132-134.

Mason, Tony; Brown, Doug (1990): lex & yacc, Nutshell Handbooks, O'Reilly & Associates, o.O., 1990.

Naur, Peter (1963): Revised Report on the Algorithmic Language ALGOL 60, in: Communications of the ACM, 6. Jg., Nr. 1, 1963, S. 1-17.