

# Automatically Generating OLAP Schemata from Conceptual Graphical Models

Karl Hahn  
FORWISS  
Orleansstr. 34  
D-81667 Munich, Germany  
+49-89-48095225  
hahnk@forwiss.de

Carsten Sapia  
FORWISS  
Orleansstr. 34  
D-81667 Munich, Germany  
+49-89-48095219  
sapia@forwiss.de

Markus Blaschka  
FORWISS  
Orleansstr. 34  
D-81667 Munich, Germany  
+49-89-48095230  
blaschka@forwiss.de

## ABSTRACT

Generating tool specific schemata and configuration information for OLAP database tools from conceptual graphical models is an important prerequisite for a comprehensive tool support for computer aided data warehouse engineering (CAWE). This paper describes the design and implementation of such a generation component in the context of our BabelFish data warehouse design tool environment. It identifies the principal issues that are involved in the design and implementation of such a component and discusses possible solutions. The paper lists typical mismatches between the data model of commercial OLAP tools and conceptual graphical modeling notations, and proposes methods to overcome these expressive differences during the generation process. Further topics are the use of graph grammars for specifying and parsing graphical MD schema descriptions and the integration of the generation process into a metadata centered modeling tool environment.

## Categories and Subject Descriptors

H.2.1 [Database Management]: Logical Design – *data models, schema and subschema*

## Keywords

OLAP, Data Warehouse, Conceptual Design, Graphical Multi-dimensional Modeling Notation, Multidimensional Schema Generation

## 1. INTRODUCTION

Designing and implementing a data warehouse environment is a highly complex engineering task that calls for methodological support. An indication of this is a large interest in data warehouse specific design methodologies in practice and by researchers (e.g. [5], [7], [11]). In order to be successful, such a methodology must be supported by a tool environment helping the designer in specifying and implementing the warehouse system. Being a

special form of CASE tools, we call these environments *Computer Aided Warehouse Engineering* (CAWE) tools. Supported by such a design tool environment, the warehouse modeler specifies the system using a set of graphical notations. Our approach to building a CAWE system is the BabelFish tool environment (which is described in more detail in section 3). It is designed with the following objectives in mind:

- the designer uses a set of graphical notations to specify the warehouse system
- the system design is performed on a conceptual level not taking into account any implementation details.
- the specifications cover all relevant aspects of the warehouse design (data model design, data transformation design, analysis application design, security design etc.).
- the BabelFish environment provides support for the whole lifecycle of a data warehouse project (initial design, implementation, maintenance and redesign/evolution)
- the BabelFish system automatically ensures the consistency between the different parts of the specification (e.g. the static and the dynamic system view) and between the specification and the implementation ('round trip engineering').

One of the immediate consequences of this set of objectives is that the implementation of the data warehouse must be automatically generated from the graphical conceptual models. This generation process has long been recognized as a key feature of CASE tools that can generate e.g. database schemata, class definitions or running prototypes from the graphical specification.

Data warehouse systems are implemented using a set of commercial components (e.g. extraction tools, database systems, metadata management systems, OLAP tools, report generators) that are configured according to the needs of the project. The syntax and the extent to which they can be configured are largely tool-dependent. According to our basic design principles, these peculiarities of the systems should be hidden from the designer by the generation process. This implies that the generation process must perform a mapping between the semantics of the tool-independent graphical notation and the semantics of the tool specific configuration (e.g. the multidimensional data model and the tuning parameters). If the tool does not provide native constructs to represent each element of the graphical notation, adequate transformations of the original model (with a minimal loss of semantics) have to be performed during the generation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*CIKM/DOLAP '00*, November 10, 2000, Washington DC, USA.  
Copyright 2000 ACM

process (see section 5). On the other hand, if the target systems supports more than one representation for an element of the graphical notation, the generation process should (semi-)automatically choose the optimal representation (with respect to an optimality measure specified by the designer, e.g. best access performance). This requires that the generation process contains a product-specific component that uses heuristics, cost-models and maybe information provided by the user to drive these decisions.

As part of the BabelFish environment, we implemented a generator producing configurations for the core components of a warehouse, namely the warehouse database schema and the OLAP tool configuration. In order to study the principal issues involved in automatic data warehouse generation we chose two typical real world OLAP products as target systems. This paper presents our experiences with implementing the generator and its integration into our design tool environment.

## 2. RELATED WORK

Different proposals have been made regarding how to graphically represent a conceptual multidimensional schema for interactive modeling purposes (see e.g. [2], [3], [4], and [5]). Additionally, automatic generation of complete OLAP tools has recently been addressed by the GOLD model proposed by C. Trujillo et al. ([13]). The approach which comes closest to our ideas and objectives is the ODAWA project [7]. However, none of these approaches addresses the issue of how to translate the graphical schema representations to configurations for real-world OLAP tools and how to cope with the different expressiveness of the data model used by the target system and the data model assumed by the modeling notation.

An additional distinctive features of the BabelFish approach compared to all these approaches is the view concept (see section 3) that allows to model interconnections of the static multidimensional schema with other aspects of the warehouse model (dynamic model, transformation model, data source model, security model etc.) and to exploit these interconnections for the design process.

Due to space constraints, we refer the reader to [6] for an in-depth discussion and comparison of related work.

## 3. THE BABELFISH FRAMEWORK

This section presents an overview of the basic architecture and design principles of our modeling framework BabelFish into which the generation algorithm is being embedded. Due to space constraints, we can only give a brief overview of the concepts. For details of the approach we refer to corresponding further publications mentioned in the text.

### 3.1 Basic Concepts

The core of the BabelFish approach is a comprehensive object-oriented model of a data warehouse containing all the details that are necessary to specify a data warehouse (e.g. the data cube names, a description of their dimensions, the classification hierarchies, a description of the data sources, the tool-specific database schema). We refer to the object oriented schema of the warehouse model as warehouse metamodel (e.g. containing a class *dimension*). Such a metamodel is far too complex to use it for modeling purposes or graphical representations (e.g. [8] presents such a comprehensive metamodel). Therefore, we follow the view

based approach (cf. Figure 1) which has already been successfully deployed for Object Oriented Software Engineering tools. This means we defined certain subsets (views) of the warehouse model as part of the BabelFish method design. The designer (CAWE user) indirectly manipulates the warehouse model through graphical visualizations of these views. Each view represents a certain aspect of the warehouse design, e.g.

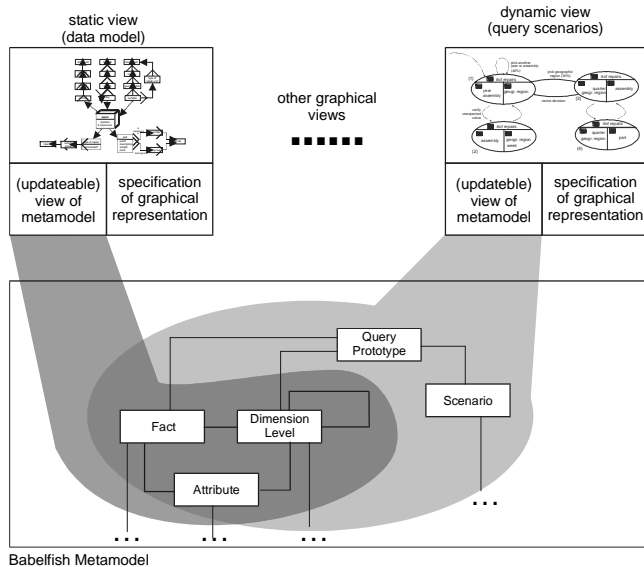
- *static data model view* (cf. [11]) describing the conceptual schema of the warehouse (i.e. a multidimensional model of the data that originate from the business processes). For this purpose we use ME/R, a multidimensional modeling language (see chapter 4).
- *dynamic view* (cf. [10]) specifying typical analysis tasks the end user performs using the warehouse data (comparable to use-case scenarios)
- *functional view* specifying the functional interrelationship between data (e.g. transformations from source to target or how to compute complex derived measures)
- *data source view* describing the static structure of the operational data sources and their interrelationship to the static warehouse model, i.e. a specification of the data transformation and loading process.

A graphical view definition consists of two components: a query defined with respect to the metamodel selecting a part of the warehouse model (e.g. selecting all the conceptual measures, dimensions, cubes, and their relationships) and the specification of how to actually display the selected parts (e.g. stating that a dimension level should be represented as a node in the view graph and depicted as a rectangle which is labeled with the name of the level, cf. [12]).

For BabelFish, we restricted the structure of the views to typed graphs because the syntax of these structures can be elegantly defined via layered graph grammars. Furthermore, classes, relationships, and attributes of the warehouse metamodel can be easily mapped to node and edge types. Additionally, manipulations of the graph structure can be easily transformed to manipulations of the warehouse model (e.g. inserting a new node inserts a new object of the respective class). This ensures that the views are updateable.

The metamodel contains complex integrity constraints and relationships between objects taking part in different views. Furthermore, a single object of the warehouse model can participate in different views. Consequently, interrelationships between the different aspects of warehouse design can be formulated in an elegant way. E.g. objects of the class *dimension level* are part of the static data model view and the dynamic data model view (cf. Figure 1). Whenever a modeler creates a new dimension level (e.g. by adding a node to the ME/R representation of the static data model view), this dimension level is automatically available in the dynamic system view for the description of user query behavior. This can also be used to enforce inter- and intra-view consistency. E.g. if a new dimension level is being added in the dynamic system view as a part of a report, it is also automatically available in the static system view. An integrity constraint defined for the static view states that every dimension level must be connected to at least one fact. Thus, when checking the consistency of the static view, the system can

automatically remind the user to update the static view (e.g. by connecting the new dimension level to a fact node).



**Figure 1. Graphical views of the comprehensive BabelFish metamodel are used to specify the DW**

Since the objects of the metamodel are assigned to different layers of abstraction, we additionally see the BabelFish metamodel divided into three layers:

- on the *conceptual layer*, objects like the multidimensional schema are modeled independently of implementation decisions (e.g., system architecture, tools). These objects represent a model of the universe of discourse to the designer.
- the *logical layer* contains technical objects (e.g., to model the used architecture). The schema of this layer is determined by the external interfaces (data models) exhibited by the tools that are used for the implementation (e.g., when using a relational database system and a star schema to store the data, the logical layer describes the relational mapping of the conceptual schema).
- the scope of the *physical layer* are tool-specific issues like clustering or indexing strategies (in the case of database systems).

This layer model reflects our objectives that the OLAP modeler performs the conceptual design tasks using a graphical modeling tool and that the BabelFish tool environment ensures consistency between the conceptual specification and the implementation (corresponding to the logical and physical layers).

Following the philosophy that the specification of the system should be done at the conceptual level, all the modeler's views are defined using the objects of the conceptual layer. Specialized generation programs automatically propagate changes made to the warehouse model to the lower layers (e.g. generation of an OLAP database schema, an extraction program). The generator component described in this paper is an example of such a program.

The objects of the different layers are linked via relationships in the metamodel, e.g. making it is possible to trace which object on the logical layer belongs to which object on the conceptual layer. This information is important, for example, when transforming schema evolution operations from the conceptual to the logical layer [1].

It is also possible to define additional views on the logical and physical level in order to provide system administrators with the possibility to inspect and revise optimization decision taken by the generator. An analyzer component should propagate these changes back to the higher level layers.

### 3.2 The Prototype Implementation

The complete metamodel (containing all three layers of abstraction) is stored in a repository system (Softlab enabler for our prototype) together with a description of the view definitions themselves and a definition of the graphical representation of the elements (cf. [12]). The repository is the central coordination channel for communication between the different components of the framework. Ideally, the repository should offer updateable views. If this service is not provided (like in our case) it has to be simulated by an additional layer on top of the repository system [12].

A generic graph editor (GraMMi) is used by the modeler to specify the different models via the view provided onto the metamodel. GraMMi reads the information about the graphical representation of the different elements in a view from the repository at runtime, configures its interface accordingly and enforces the integrity checks that can be performed at the graph level (formulated as graph grammar rules). Therefore, the same graphical editor can be used for all the views ([12]).

Since the repository system does not support views on metadata objects, an additional layer provides the view functionality. The generator component described in this paper is called MERTGEN in the prototype implementation. MERTGEN reads the information about the multidimensional schema from the repository and generates a corresponding tool specific executable configuration for an OLAP tool (see chapters 4 to 6 for details).

When the schema designer modifies the multidimensional schema (represented by the ME/R graph), GraMMi creates corresponding schema evolution jobs. The evolution component (called EWOK in the prototype) reads these evolution jobs from the repository and generates corresponding logical evolution commands that transform the database schema representing the multidimensional schema and adapt the data persistently stored in the database [1].

Both MERTGEN and EWOK have tool-specific plugins for the commercial products we chose in our prototype implementation. Currently, our prototype supports Cognos Powerplay and Informix Metacube as target OLAP tools.

## 4. THE STATIC DATA MODEL VIEW

The generation process producing the database schema and configuration information to be used by the OLAP tool uses the part of the metamodel which is visualized by the static view (using the ME/R notation). Therefore, for the rest of this paper, we concentrate on this view using the visualization in order to illustrate the concepts. Note however, that the generation process is naturally unaware of the graphical layout of the representation.

The static data model view is the central view of the BabelFish methodology as described in chapter 3. The purpose of the view is to describe the multidimensional structure of the OLAP schema. ME/R (Multidimensional Entity-Relationship Model) is an extension of the Entity-Relationship Model especially for multi-dimensional modeling.

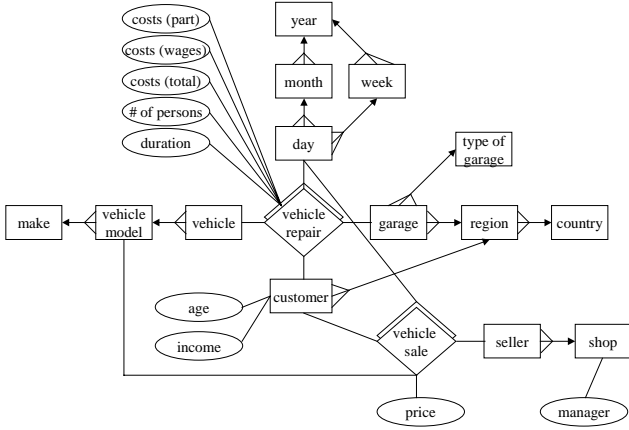


Figure 2. ME/R example schema

For the rest of this paper we will refer to the example schema shown in Figure 2. It illustrates the schema of a data warehouse that is used to analyze data about vehicle repairs and vehicle sales. Consequently, it contains two fact relationships (vehicle repair and vehicle sales). Each fact is characterized by quantifying attributes (e.g. price, costs, duration, etc.) that can be analyzed according to the dimensions (vehicle, customer, garage, seller and day). Dimensions have a hierarchy, consisting of dimension levels (e.g. vehicle, model, brand). Some dimensions are shared completely (customer and the time dimension) or partially (vehicle) by the two facts. A typical OLAP query against this schema could be: “show me the total costs of all vehicle repairs of make BMW in the last month”.

As the BabelFish methods requires all views to be represented as typed graphs, ME/R schemata can be represented in this form. A typed graph over a set of edge types  $\Sigma_E$  and a set of node types  $\Sigma_N$  is defined as a tuple  $(N, E, t_N, t_E, s, t)$  (see [6], [1] for details).

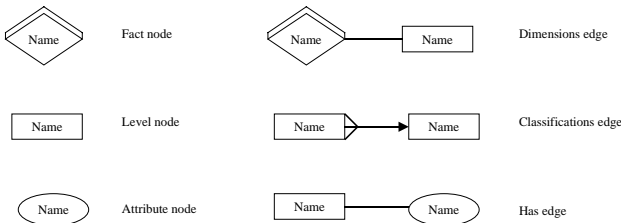


Figure 3. Representation of ME/R as a directed graph

The direction of the edges, i.e. the information which node is source and which node is target, is essential for *classifications edges* to represent the hierarchy (e.g. used for roll-up or drill-

down operations) inside the dimensions. The graphical representation of the ME/R graph elements is shown in Figure 3.

As schemata are represented as typed graphs, we use a graph grammar to describe the syntactical constraints of the modeling notation. A graph grammar over a set of edge types  $\Sigma_E$  and a set of node types  $\Sigma_N$  is defined as a tuple  $(\lambda, P)$  where  $\lambda$  is a nonempty initial typed graph over  $(\Sigma_E, \Sigma_N)$ , called the axiom.  $P$  is a finite set of productions. Each production  $p$  is of the form  $L \rightarrow R$  where  $L$  and  $R$  are typed graphs over  $(\Sigma_E, \Sigma_N)$  with  $L$  being the left hand side and  $R$  being the right hand side.

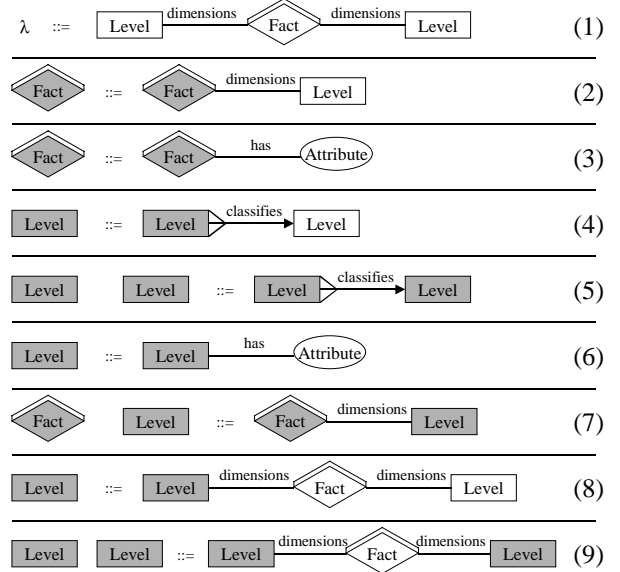


Figure 4. ME/R graph grammar

The replacement of nonterminals in graphs is far more complicated than in linear texts [9]. Therefore, different embedding strategies have been proposed to solve this problem. As outlined in [9] we use the concept of contexts. This means that both sides of the production contain a common context graph that allows for defining which part of the existing graph the new elements should be connected to.

Figure 4 gives an overview of the productions of the ME/R grammar. The nodes, which are marked gray on the left hand side represent nodes that must exist in the graph in order to apply the production. On the right hand side, nodes marked white are new nodes, created by the application of the production.

Details about the graph grammar can be found in [12].

## 5. THE TARGET PLATFORMS

There is a wide range of OLAP tools available but one can divide all of the products into two main design principles MOLAP (Multidimensional OLAP) and ROLAP (Relational OLAP). In order to show the overall applicability of our approach we implemented the generation process for two technology-leading representatives of these principles. We chose Cognos Powerplay representing MOLAP products and Informix Metacube as a typical ROLAP product. Nevertheless, our findings regarding data model expressiveness apply to all similar products.

The only precondition for an OLAP tool to be integrated as a target in the BabelFish environment is the existence of any kind of an accessible (not necessarily documented) interface for schema definitions. This can be a scripting language like MDL for Cognos Powerplay or database tables that contain the metadata (Informix) that can be accessed with SQL.

A comparison of the respective data models with ME/R revealed the following differences:

- *Different constructs, respectively different names for similar constructs.* Most of the OLAP tools have a different data model, i. e. different constructs to build the multidimensional schemata. As an example we could have a look at the *attribute node* construct that can specify a *fact node* or a *level node* in ME/R. Powerplay calls the fact attributes “measures” but has no possibility to describe a level with a specifying attribute. Metacube can handle attributes for level nodes and fact nodes, but calls the attribute for a fact a “measure”.
- *Richer semantics of ME/R compared to the OLAP tools.* Merging dimensions for instance, i. e. dimensions, that have parts of their hierarchy in common (e.g. geographic information is part of the dimensions customer and garage in Figure 2) cannot be modeled this way in any of the investigated OLAP tools.
- *Additional tool-specific information.* The OLAP tools can store a lot of semantic information, e.g. data source, description, data type, constraints, etc. This information is missing in current ME/R schemata but ME/R has an extension to store any kind of semantic details.

Most OLAP systems have limitations concerning the expressiveness of the multidimensional data model, compared to ME/R. Furthermore, there is a wide diversity in the data models of OLAP tools and a widely ambiguous terminology. These drawbacks strengthen our motivation to provide the user with a tool independent notation for multidimensional schema design, which does not impose tool-specific limitations. This complies with our design principle that all implementation-specific details should be hidden from the designer.

Typically, the OLAP tools have weaker semantics compared to ME/R. We have found four different restrictions, i. e. constructs of an ME/R schema that can not be adequately represented using the investigated OLAP tools (cf. Table 1). For all four cases we present a transformation that preserves as much of the original semantics as possible in the target tool’s data model.

1. *Specifying attributes for dimension levels.* In Figure 2 we can find the specifying attributes *age* and *income* for the dimension level customer which can not be represented in some MOLAP tools.
2. *Merging dimensions* are very common in multidimensional modeling. That’s why they can be very naturally expressed within ME/R. If a part of a hierarchy is used by more than one dimension, you have a *classifies* relationship from a dimension level to a second dimension level, that is part of another dimension. We have e.g. in our example the merging dimensions customer and garage, that share the dimension levels region and country.

3. *Alternative paths in the classification schema graph* occur if the modeled domain allows for alternative classifications (e.g. day-week and day-month) within the same dimensions which can be classified according to common criterion (e.g. year). Cf. Figure 2.
4. *Multiple facts.* An ME/R schema can investigate more than one subject of analysis. For each subject we can model a fact relationship in this schema. Fact relationships can share dimensions, e.g. the time dimension is essential in OLAP, so it is usually shared by all fact relationships. The example includes two fact relationships vehicle repair and vehicle sale, that have common dimensions, the time dimension and the dimension customer.

**Table 1. Comparison of the expressiveness of the data models of exemplary OLAP tools**

	<b>Cognos Powerplay 6.0</b>	<b>Informix Metacube 4.02</b>
Attribute for dim. levels	Not provided	Supported
Alternative paths	Not provided	Supported
Merging dimensions	Not provided	Not provided <sup>1</sup>
More than one fact	Not provided	Supported

## 6. THE GENERATION PROCESS

The generation of the OLAP schema(ta) and configuration from the conceptual ME/R schema is divided into four phases (Fig. 5):

1. Loading the static data model view from the repository system (Softlab Enabler for our prototype). The schema is represented as a typed graph in the view (see above).
2. The graph has to be parsed using the graph grammar of Fig. 4 to ensure syntactical correctness. This parsing can avoid errors during the generating process and can give feedback to the user about the type and the location of the syntactical error.
3. The (syntactically correct) graph has now to be adapted to the data model of the target system. As outlined above typical OLAP systems have limitations in their multidimensional data model (compared to ME/R).
4. Creating the output for the target system. As a result of the adaptation of the graph in phase 3, we can now translate every ME/R construct to a corresponding construct of the target OLAP system. The logical schema is finally written back into the repository (logical layer of the metamodel).

Scanning and parsing relate to the ME/R schema and are independent of the used OLAP tool. The transformation step handles the restrictions of the target system. When integrating a new OLAP target system in the BabelFish tool the tools data model has to be inspected and already known limitations can be resolved by using the respective transformation method. The generation phase has to translate the ME/R constructs to

<sup>1</sup> Please note that we intend to create a star schema as Informix does not provide a real snowflake schema. In fact merging dimensions are possible with a snowflake schema.

constructs of the target system using the interface of the OLAP tool and therefore has to be newly programmed.

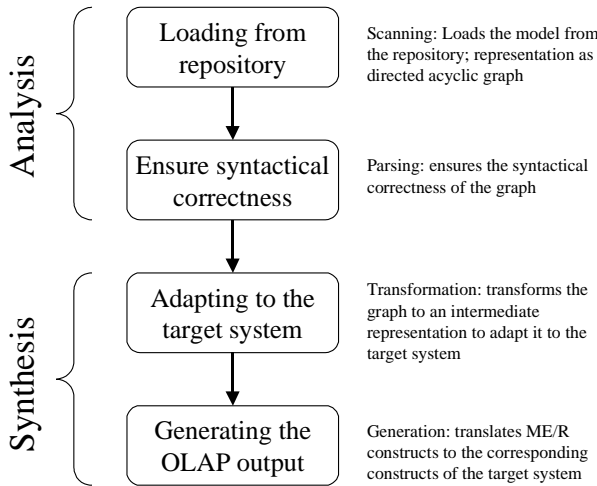


Figure 5. Four phases of the generation process

### 6.1 Building a parser for a graphical notation

To ensure the syntactical correctness of a schema we built a parser referring to the graph grammar of ME/R. This gives the modeler feedback about syntactical errors, avoids errors during the generation process and the creation of faulty OLAP schemata.

The parser, based on the graph grammar described in section 4, is a bottom-up parser specific to our grammar. We define an order in which the productions have to be applied and traverse the graph to find such a production. This method avoids a complex backtracking algorithm like outlined in [9].

### 6.2 Adapting the schema for the target system

As described in chapter 5, there are four characteristic limitations in OLAP tools compared to ME/R. For all four cases we found an adaptation of the corresponding construct to preserve as much as possible of the semantics.

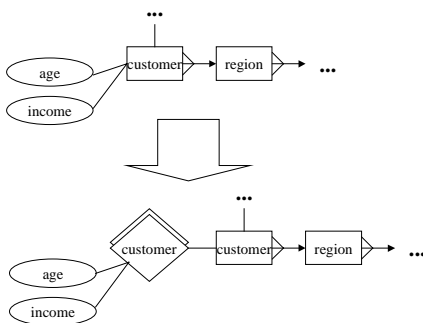


Figure 6. Attributes of a dimension level

The first case is a dimension level with a describing attribute, which can not be modeled in some MOLAP systems (see Figure 6). Just skipping the attributes solves the modeling problem but the extra information about the dimension level would be lost. To avoid this, we create a new fact node with all the attributes

(because fact nodes can possess attributes), which is the source of a dimensions edge to the original dimension level. In other words the new fact node is one-dimensional but inherits the full hierarchy of this dimension.

You might notice that the schema now has (at least) two fact nodes, which conflicts with limitation of some OLAP tools. For this reason the transformation rules have to be applied in the order as presented here.

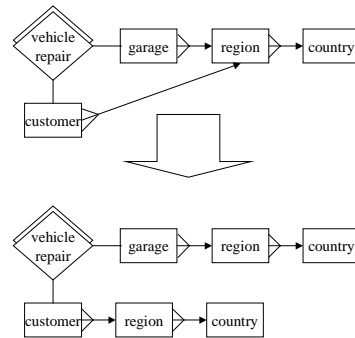


Figure 7. Merging dimensions

A different approach to the problem, modeling the level attributes as dimension levels, has been abandoned to avoid adding extra semantics to the schema that is not useful, e.g. a new classification edge between customer and age.

Merging dimensions are the second problem we have to solve. None of the evaluated tools could deal with merging dimensions (please note that a snowflake schema in a ROLAP tool generally could easily handle the problem. However, Informix Metacube does not support a normalized schema). In order to obtain the full hierarchy for all dimensions, we have to duplicate the hierarchy starting at the dimension level, where the merging occurs. As an example, in Figure 7 the hierarchy *region-country* is duplicated for the dimension *customer* as it is already part of the dimension *garage*. The structure now reflects the original hierarchy, but what gets lost is the information that the data of *region* and *country* has the same origin for both dimensions.

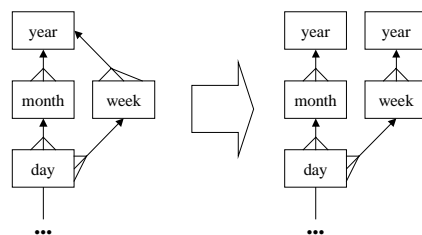


Figure 8. Alternative paths inside a dimension

The next step is to resolve alternative paths inside a dimension (see Figure 8). The data model of Powerplay can not describe this, because it can only handle dimension schemata with a tree structure (different hierarchies can not merge in an endpoint). Similar to the problem of merging dimensions, we duplicate the

dimension beginning at the merging point, obtaining the full hierarchy.

Finally, a Powerplay schema can only describe one multi-dimensional cube. To analyze different subjects you have to create a multidimensional cube in a single Powerplay schema for each.

Referring to our graph, we have to create disjoint sub graphs with only one fact in each sub graph to generate the schemata for Powerplay. After deploying step 1 to 3, the only connecting points between the various facts in our graph can be shared dimensions. These shared dimensions have to be duplicated for each of the facts.

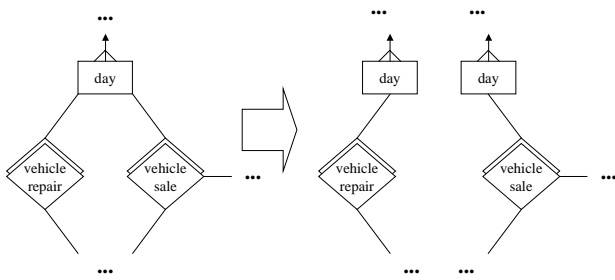


Figure 9. More than one facts (with shared dimensions)

In order to summarize, the conducted transformation operations preserved a large part of the original semantics such that all the queries that can be answered by the ME/R schema can be answered by the OLAP schema as well. This is achieved by generating redundancies whenever it is necessary. What we have lost are the integrity constraints, due to the fact, that we have duplicated elements.

### 6.3 Generating the OLAP output

After preparing the graph for the generation, i. e. the graph has been transformed to adapt it to the limitations of the data models of the target systems and extra information has been added to the graph, it can now be used to create a target-dependent OLAP schema. Table 2 presents the mapping of elements of the ME/R data model to the corresponding elements of the target data models. We found some redundancies in the target system’s data model definitions, presumably for performance reasons (e.g. the first level of a dimension has to be defined in Powerplay, although this information is already inherent in the definition of the dimension levels).

In order to avoid interference with a running warehouse system, the generation process must work without connecting to the system. Therefore, we have to use a language that can create the schemata of the OLAP products to write scripts containing the full schema. In the case of Powerplay there is MDL (Model Definition Language) that can be seen as a data definition language to create Powerplay schemata.

In fact, MDL is not designed to be directly written by the schema designer, which is mirrored in the fact that the MDL parser has no error handling. To build the multidimensional data definition statements in MDL the ME/R graph has to be traversed multiple times, because the creation of the constructs has to be done in a special order in MDL. Furthermore, the construction needs (sometimes redundant) information of other corresponding parts

of the graph (e.g. a dimension level requires information of all the hierarchies in its dimension). The example of MDL below shows the definition of the dimension “vehicle” and its first dimension level:

```
DimMake "Vehicle" DimType Regular
RootCatMake "Vehicle" Dimension "Vehicle"
DrillCatMake "By Vehicle" Dimension "Vehicle"
Root "Vehicle"
LevelMake "Brand" Dimension "Vehicle" Drill "By
Vehicle" Parent "" Source "Brand"
(other dimension levels analogously)
```

Table 2. Mapping of ME/R elements and the elements of the target data models

ME/R data model	Powerplay data model	Metacube data model
<b>Schema</b>	Multiple models.	DSS System.
<b>Fact node</b>	Single model.	Fact table.
<b>Dimension level node</b>	Level.	Default attribute of a dim. element.
<b>Attribute of a level node</b>	Not provided <sup>2</sup> .	Attribute of a dim. element.
<b>Attribute of a fact node</b>	Measure.	Measure (internal terminology: fact).
<b>Dimensions edge</b>	Not necessary due to the limitation of one fact per model.	Modeling element <i>dimension</i> . Relationship between a dim. and fact is stored in a separate table.
<b>Classifications edge</b>	Stored with the target node <sup>3</sup> .	Hierarchy (internal terminology: rolls up).
<b>Has edge</b>	No corresponding element.	No corresponding element, stored with a measure.

Metacube as a ROLAP product stores the data and the metadata, e.g. the names of the dimension levels, the hierarchical structure etc., in a relational database system (in our case Informix Dynamic Server 7.30). In order to create the OLAP schema for Metacube we use SQL DDL statements. As we decided to use a star schema we first have to define the tables for the data, i.e. a table for each fact node (and its corresponding attributes) and each dimension (with its dimension levels and attributes). Furthermore, the meta information of the schema has to be inserted in the tables where Metacube stores the metamodel (SQL DML statements). The (simplified) example below gives an expression of the SQL scripts automatically generated by our generation tool. The mnemonics (e.g. l\_1 for the first dimension level of the dimension) simplifies the interaction with other BabelFish views working on the generated warehouse system.

<sup>2</sup> The adaptation of the schema for Powerplay has eliminated these level attributes

<sup>3</sup> This modeling is responsible for the restriction that alternative paths in dimensions are not possible.

These scripts, generated by the prototype generator MERTGEN defines fully functional OLAP schemata for both target systems.

```
-- create dimension table for the data
create table "informix".dt_4 (id integer
primary key, l_1 varchar(30), l_2 varchar(30),
l_3 varchar(30));

-- create the entries in the (existing)
Metacube system tables
-- create a dimension
insert into "metacube".dim (dim_id,
dss_system_id, dim_desc, dim_type,
dim_schema_name, dim_table_name,
dim_to_fact_key) values (4, 1, 'Vehicle', 0,
'informix', 'dt_4', 'id');
-- define user interface for the dimension
insert into "metacube".ui_dim (dim_id,
dss_system_id, icon_bitmap, nofilter_label)
values (4, 1, 0, 'All Values');
-- correlation between facts and dimensions
insert into "metacube".fact_dim_mapping
(dss_system_id, fact_table_id, dim_id,
fact_to_dim_key) values (1, 1, 4, 'l_4_id');
-- create user interface for this element
insert into "metacube".ui_fact_dim
(fact_table_id, dim_id, dss_system_id,
screen_order) values (1, 4, 1, 1);
```

## 7. CONCLUSIONS AND FUTURE WORK

The central assumption of our BabelFish approach is that the data warehouse designer models the universe of discourse on a conceptual level using graphical notations. He is being supported by a computer aided warehouse engineering (CAWE) environment. This environment generates the implementation of the conceptual design models, thus hiding the implementation details (e.g. limitations of the target system) from the modeler. In this paper, we discussed the issues of this automatic generation process for both the OLAP database schema and the frontend configuration.

The different expressiveness of the conceptual notation compared with the data models of the commercial OLAP tools turned out to be the main challenge for the generation process. However, we managed to find transformations of the conceptual multidimensional schema (described as graph transformations) that preserved a large part of the semantics. Experiments showed that the resulting OLAP application matched the designers view of the application domain's universe of discourse. Additionally, an analysis of the data model of other OLAP tools showed that in general our list of limitations presented in section 5 is a good classification of OLAP tool limitations. Thus, implementing handlers for these limitations, it is possible to keep large parts of the generation component independent of the actual target system (of course, the code generation is always tool specific).

These results show that a generation is generally feasible and useful, encouraging further research work in this direction. For example, generation of schemata is only useful during the initial design phase when no data has been loaded into the system yet. During later iterations of the design cycle, changes to the schema also have take the existing data into account. This schema evolution issue for OLAP systems has been extensively discussed in [1].

OLAP tools offer different tuning parameters for the implementation of a logical multidimensional schema on the

physical layer (e.g. snowflake schema vs. starschema, sparsity handling mechanisms). In order to exploit these mechanisms it is necessary to have information exceeding the information contained in the static schema (e.g. concerning the typical workload). Therefore, it seems promising to research the possibility of using larger parts of the warehouse metamodel (e.g. the dynamic view) as input to the generation process.

## 8. REFERENCES

- [1] M. Blaschka: FIESTA – *A Framework for Schema Evolution in Multidimensional Databases*, PhD thesis, Technische Universität München, Germany, 2000.
- [2] Dan Bulos. *OLAP database design: A new dimension*. Database Programming&Design, Vol. 9(6), June 1996.
- [3] L. Cabibbo, R. Torlone: *From a Procedural to a Visual Query Language for OLAP*. In 10<sup>th</sup> IEEE Int. Conference on Scientific and Statistical Database Management (SSDBM 98), Capri, Italy, 1998.
- [4] M. Golfarelli, D. Maio, S. Rizzi: *Conceptual design of data warehouses from E/R schemes*. In Proc. of the 31<sup>st</sup> Hawaii Int. Conference on System Sciences (HICSS'98), Hawaii, USA January 1998.
- [5] M. Golfarelli, S. Rizzi: *A Methodological Framework for Data Warehouse Design*, In Proc. of the 1<sup>st</sup> International Workshop on Data Warehousing and OLAP (DOLAP), Washington, DC, USA, 1998.
- [6] K. Hahn, C. Sapia, M. Blaschka: *Automatically Generating OLAP Schemata from Conceptual Graphical Models*, Technical Report, FORWISS, Munich, Germany, October 2000, [www.forwiss.tu-muenchen.de/~system42/publications](http://www.forwiss.tu-muenchen.de/~system42/publications)
- [7] A. Harren, O. Herden: *Conceptual Modeling of Data Warehouses*, Poster, ER 1999.
- [8] R. Müller, Th. Stöhr, E. Rahm: *An Integrative and Uniform Model for Metadata Management in Data Warehousing Environments*, In Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW), Heidelberg, Germany, 1999.
- [9] J. Rekers, A. Schürr: *Defining and Parsing Visual Languages with Layered Graph Grammars*, Journal of Visual Languages and Computing 8(1): 27-55, 1997
- [10] C. Sapia: *On Modeling and Predicting User Behavior in OLAP Systems*, In Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, 1999.
- [11] C. Sapia, M. Blaschka, G. Höfling, B. Dinter: *Extending the E/R Model for the Multidimensional Paradigm*, In Int. Workshop on Data Warehouse and Data Mining (DWDM'98), Singapore, 1998.
- [12] C. Sapia, M. Blaschka, G. Höfling: *GraMMi -Using a Standard Repository Management System to build a Generic Modeling Tool*, In Proc. Hawaii Int. Conference on System Sciences (HICSS'00), Maui, Hawaii, USA, 2000.
- [13] J. Trujillo, M. Palomar, J. Gómez: *The GOLD Definition Language (GDL): An Object Oriented Formal Specification Language For Multidimensional Databases*, In Proc. ACM Symposium on Applied Computing (SAC'00), Como, Italy, 2000