A Modeling Tool for Multidimensional Data using the ADAPT Notation

Peter Gluchowski, Christian Kurze, Christian Schieder
Chemnitz University of Technology
{peter.gluchowski | christian.kurze | christian.schieder}@wirtschaft.tu-chemnitz.de

Abstract
Conceptual models are designed to express user requirements of information systems in a formal and self-explanatory manner from a business user’s point of view. Furthermore, they should contain enough information for IT staff to use them as a basis for technical deployment.

Within the domain of OLAP information systems, multidimensional data models are used to translate the specific user requirements into multidimensional data structures. Application Design for Analytical Processing Technologies (ADAPT) is one of the appropriate notations to express these models on a conceptual level. Up to the publication of this paper, ADAPT lacks any formal foundation. We will contribute to solving this issue by proposing a metamodel for ADAPT and a software prototype that transforms conceptual data models into logical ones in order to ease the development of data warehouse systems.

1. Introduction
Numerous steps must be taken during the development of data warehouse systems to accomplish a highly productive realization of user requirements. Although the development process is often not oriented to sequential procedure models, but passes various rebounds and iterations, the process stages requirements engineering and technical design prove to be of great importance. Regrettably, the transfer between the two phases turns out to be difficult, as the qualitative description of business requirements cannot immediately be converted into a formal technical specification.

Especially in the context of data modeling it is advisable to apply special techniques supporting the modeling of relevant structures on a conceptual level – independently of the database technology eventually used – to overcome this issue. These conceptual data models summarize the results of the information demand analysis and serve as a foundation for further developments [8, 9]. Usually, conceptual modeling relies on a graphical notation that allows writing, understanding and managing schemata by both designers and users [1].

The conceptual notation analyzed in this paper, called Application Design for Analytical Processing Technologies (ADAPT) [2], was first published in 1996 and offers a set of symbols which facilitate the visualization of multidimensional data structures in a mostly self-explanatory way. To the best of our knowledge, ADAPT lacks any formal foundation to date. The present article closes this gap by introducing a metamodel which contains precise specifications for admissible combinations and connections of model elements.

While conceptual models are to capture user requirements, logical models are to capture implementation aspects. They hide some details of physical data storage but could be implemented on a computer system directly. Modeling multidimensional data structures on a conceptual level reveals the navigation structures along the dimensions in a self-explanatory way, but the logical representation, for example by relational tables, does not; hierarchy levels and attributes are simple columns without further metadata.

The next step in implementing data models is their transformation into physical models. Their concepts describe the details of how data is stored in a specific system. Physical data models are meant for computer specialists and therefore not further covered within this paper.

Formally correct ADAPT models can be applied as a base for the automatic compilation of logical data models. However, the transformation into a logical model does not prove to be trivial as different database technologies and several realization options compete against each other [10]. This paper provides a solution which leads to the automatic transformation of ADAPT diagrams into logical data schemata for diverse target platforms.

The remainder of this paper is structured as follows: section two introduces basic multidimensional concepts; section three briefly overviews related work. An introduction to both ADAPT and the metamodel is
provided in section four. Part five focuses on the essential conversion steps from conceptual towards logical data models. The last section discusses the prototype and gives an outlook on future research work.

2. Multidimensional Data Modeling

The ontological discrimination between schema and instance layer is a fundamental aspect in modeling multidimensional data structures. This section elaborates on the conceptual elements used to describe multidimensional data schemata [3, 20]: dimensions, hierarchies, measures, and attributes. They are used to turn business data into actionable information. Indeed, a simple spreadsheet’s structure is a kind of multidimensional data structure; a two-dimensional one.

Dimensions represent the essential components of a multidimensional schema. They are a partially ordered set of dimension elements (also named as dimension members or dimension positions), which represent the dimension’s individual values; the dimension’s instance. For example, an analysis of a company’s turnover might be parameterized by the dimensions ‘product’, ‘organizational unit’ and ‘time’. Each single product sold by our company is an element of the product dimension.

Dimension elements might be grouped by means of hierarchically ordered levels, so-called dimension hierarchies. Within the schema description dimension elements are condensed into generalized abstract dimension levels (or hierarchy levels). To refer to the product dimension’s example, this leads to an aggregation of single products into product subcategories and/or product categories. The examined levels’ elements are connected by parent-child-relationships. Usually this results in tree structures with a root node (top level element), several leaf nodes (leaf level elements) and multiple nodes in between on the particular levels. Deviations from this ideal structure will occur in practical use such as parallel hierarchies or unbalanced tree structures.

The semantics of measures are determined by the semantics of their descriptive dimensions. By spanning a spatial structure of orthogonal dimensions and defining cells at the dimension element intersections, a multidimensional matrix is created. This is often referred to as a data cube or hypercube. The contents of these cells, the so-called facts, are precise numerical values of the modeled business measures. Within research community there is neither a distinct nor a generally accepted definition of the terms measure, measurement and fact. According to our understanding, measures describe the schema of facts. Within the given example, the measure ‘turnover’ is determined by the dimensions ‘product’, ‘organizational unit’ and ‘time’. Its data type should be ‘currency’. Picking out individual dimension members such as product ‘x’, organizational unit ‘y’ and time period ‘z’ allows indexing a single turnover value, the fact ‘t’.

Two different alternatives exist to model measures. On the one hand it is possible to create one data cube for each measure. On the other hand, one cube can contain several measures by introducing a measure dimension; with each member representing one measure.

For the remaining part of the paper it is essential that measures are not isolated from each other. Often, they are linked via calculation rules. These rules are used for the dynamic calculation of dependent facts from the externally given independent facts. Complex dependencies within systems of financial control can be created in this way.

Attributes are an integral part: business attributes in particular give a deeper insight into multidimensional data structures. There are two distinct options. Attaching attributes to a dimension implies that every single element within this dimension inherits these attributes, for example each product and each product group. Applying an attribute to a certain hierarchy level opens up the possibility of attributes that are valid for this individual level only; each product might have a weight whereas a product category will not have a weight.

3. Related Work

This section categorizes some of the related work. Due to numerous studies in this area this can be a short overview only and makes no claim to be complete.

3.1. Modeling Frameworks

Several methodologies are intended to represent multidimensional data models on a conceptual level. They can be categorized into three different types [1]: extensions to the Entity-Relationship model, extensions to the UML, and ad hoc models. Each one is appropriate to represent basic multidimensional concepts but they differ significantly in their ability to represent more sophisticated concepts such as irregular hierarchies.

Entity-Relationship based models extend the basic ER notation by means of multidimensional concepts. The ME/R model [4] extends the ER model by adding three elements: a fact relationship set, a dimension
level set and a rolls-up relationship set. Using this approach, only static data structures can be described; dynamic and functional aspects are not covered. [5] uses dimensions, fact relationships, cardinalities in Martin notation, hierarchies and analysis criteria as well as a ‘xor’ operator to express hierarchy anomalies.

Adaptations of UML are exemplified in [6] and [7] as well as the customizations of the model driven architecture (MDA) such as [8]. The approach in [6] uses stereotypes to typify classes as cubes, dimension levels, measures and so on; they are displayed with the same icons as in the ADAPT notation. [7] proposes a UML profile for multidimensional data modeling. The MDA approach presented in [8] tries to offer a framework for all the relevant data warehouse components, for example ETL processes, data sources and repositories. The authors extend the UML as well as the Common Warehouse Metamodel (CWM) and use the Query/View/Transform (QVT) language for establishing transformations between different models.

Ad hoc approaches raise the level of abstraction by directly using domain concepts. Therefore, ad hoc approaches can be seen as Domain Specific Languages (DSLs) [18]. They are especially useful if the modelers themselves are not software developers; language visualization and ease of use are emphasized. Modelers do not have to cope with stereotyped classes or variations of entity types; they can intuitively use multidimensional modeling concepts like cubes and dimensions.

Therefore ad hoc approaches differ from ER or UML approaches by not adapting a certain notation to new fields of application, but developing a new visual language in order to support modelers on a higher level of abstraction.

ADAPT ranks among these ad hoc approaches. There are several other methodologies with a formal foundation such as the Dimensional Fact model [9]. The model consists of some fact schemes whose basic elements are facts, measures, attributes, dimensions and hierarchies. They are accompanied by several other features such as attributes or the additivity of attributes along dimensions.

We have chosen ADAPT for the purpose of this paper because it allows the creation of semantically rich models due to several different model elements. Furthermore, ADAPT is relatively easy to learn and mostly self-explanatory. In contrast to other ad hoc modeling techniques a stencil for Microsoft Visio has been made available free of charge, thus facilitating the application of the modeling language in practice.

3.2. Model Transformation

Some of the above-named papers already include possibilities to transform conceptual models into logical representations. Several approaches concentrate explicitly on transforming conceptual models into logical ones like [5]; and sometimes the other way round, which is actually more relevant, as seen in [10]. These approaches are often discussed under the term schema evolution. It has gained much interest in both research and practice. Therefore, an online bibliography concentrating on this area is available [17]. Currently 418 papers on schema evolution are listed.

In most cases, the methodologies concentrate on a star schema or one of its variants. The prototype presented in section 5 tries to offer an open architecture which can be extended in order to support more than one target schema.

A new level of abstraction has been introduced by the concepts of model management, as commented in section 6. Currently, the cited online bibliography [17] contains 61 papers on this approach.

4. ADAPT foundations

The ADAPT notation emerged during the 1990s in the course of an attempt to create a graphical, business oriented representation of OLAP data models [2]. Due to its pragmatic roots, the notation lacks any formal foundation. Further, by the time of the publication of this paper, the modeling language had largely been ignored in scientific publishing on conceptual modeling. To overcome these deficiencies we will introduce the basic building blocks of the modeling language, demonstrate their usage and subsequently provide a formal foundation by presenting a UML-based metamodel.

4.1. Modeling with ADAPT

The notation provides a variety of symbols which are depicted in figure 1. Each one of them represents a conceptual object of an OLAP application.

A common issue of modeling in analytical contexts is the necessity to not just model the schema of a multidimensional problem (such as hierarchy levels), but to model specific instances within this schema where appropriate (such as individual dimension members). Dynamic aspects, especially calculation models of derived measures, are of great importance. ADAPT closes both gaps by offering special symbols, as described within the following paragraphs.
The basic elements of this notation are *Hypercube* (or *Cube* for short) and *Dimension*. A Hypercube is the basic unit of storage for business data in a multidimensional database, physically, an n-dimensional array. A dimension is the representation of an axis of such a cube. Detailed modeling of dimensions is done via the symbols *Hierarchy*, *Level*, *Member*, *Attribute*, *Scope*, and *Model*.

![Figure 1. Basic symbols](image)

We show the elements’ utilization and interaction by applying the notation to model the business case of a company which wants to analyze its product sales (e.g. units and turnover) by different organizational units (e.g. stores) to customers (e.g. in different geographic locations) over time (e.g. months). This simple example can be sufficiently modeled by using one cube (Sales) with five dimensions (Organizational Unit, Customer, Product, Time, Measures). In a first step we refrain from displaying the dimensions in detail. The resulting diagram is shown in figure 2.

![Figure 2. Sample cube with five dimensions](image)

By taking a detailed look at the product dimension we can demonstrate the other design elements. Our example company sells products from different suppliers in several product subcategories which aggregate into product categories. For the abstract representation of these objects the *Level* element is used. The sales department wants to analyze the data by supplier as well as product categorization. Therefore, we need to model two parallel hierarchies (Category and Supplier) with different depths or number of levels.

The *Member* element is particularly useful when modeling dimensions with few elements such as *budget*, *actual* and *variance* in a scenario dimension or a small number of key performance indicators in a measure dimension. Dependencies between measures or scenarios can be indicated by using the *Model* element. Models help to represent calculation rules within systems of financial control, for example the DuPont-System. To leverage diagram clearness it can be useful to depict one cube per measure and use the model element to show calculations between cubes.

![Figure 3. Connector symbols](image)

Among the conceptual representation of hierarchy levels various types of relations may exist. The connection operators *Loose Precedence* (one arrow) and *Strict Precedence* (two arrows) connecting the hierarchy levels allow the modeling of unbalanced *tree* and *forest structures*. The usage of *Loose Precedence* between product and product subcategory suggests that not all products can be assigned to a specific product line. In other words, there is not necessarily a parent object for each instance on the upper next level. *Strict Precedence* on the other hand indicates the mandatory aggregation of product subcategory to product category; for each instance there is always a corresponding parent element. Figure 3 displays the provided connector symbols representing the different relationships. *Self Precedence* qualifies recursive relationships within a hierarchy. *Used By* is a specialized connector for characterizing input parameters for and dependencies between models. The simple *Connector* denotes all other relationships between any two elements.

*Scopes* or subsets summarize related dimension elements. They are suitable for modeling categories within dimensions. The elements of a scope are either explicitly defined or can be derived from other database objects by calculations. Our example uses the results from an ABC analysis to classify products into three different classes (A, B and C class products).

![Figure 4. Subset operators](image)

Four *Subset* operators visualize relationships among different scopes. *Fully Exclusive* indicates disjoint subsets which contain the complete original set of dimension members. *Fully Overlapping* separates the total dimension into non-disjoint parts. *Partially Exclusive* connects disjoint subsets, which do not cover the complete population. *Partially Overlapping*
indicates partial division of the population into subsets; dimension members may be found in more than one subset. In our example each product exclusively appears in one class. Therefore we apply the Fully Exclusive operator.

Figure 5 shows the entire representation of the product dimension in ADAPT as described in our example case above. Our exposition thereby is limited to the application of the most important elements. For a more detailed explanation see [2].

4.2. ADAPT Metamodel

This section introduces the ADAPT metamodel, which is depicted in figure 6. It has been created following the design science approach [19]. Furthermore, ADAPT can be seen as a Domain Specific Languages and the development of metamodels is a typical approach in implementing DSLs [18]. We make use of the UML notation whereby classes represent modeling objects as well as operators; stereotyped associations represent the different connector types.

Within the first step of the development process, we selected the most important elements of ADAPT which support the basic ideas of multidimensional modeling presented above. These elements form the classes within the metamodel. The second step consisted of establishing relationships between the modeling elements according to [2].

The navigation directions of the metamodel’s associations indicate the direction in which the arrow heads of each connection symbol should lead. For example, the association between Cube and Dimension is navigable from Cube towards Dimension and stereotyped with connector. This implies the usage of a connector (see figure 2) between Cube and Dimension with the arrow pointing towards the dimension. Please notice that the arrows of Strict Precedence and Loose Precedence are heading to the parent level when connecting hierarchy levels with each other.

Figure 6. ADAPT metamodel
In order to increase clarity, the classes do not contain any attributes. Every class should have at least a name as well as containers for storing the connected elements. In case of Cube this would include a collection of corresponding dimensions. A name can be omitted only at Subset Operator and its child classes.

A dimension is linked with one or more members, hierarchies or attributes via connectors. The XOR-Constraint expresses that either individual dimension members or hierarchies exist.

There might be a connector or a strict precedence between hierarchies and the uppermost hierarchy level. Using strict precedence claims that there should be an artificial overall hierarchy level. Strict precedence and loose precedence as well as self precedence connect single hierarchy levels. Strict or loose coupling allows more than one parent level; in case of a recursive relationship exactly one parent and one child level exist. Subset operators divide a hierarchy level into different dimension views. A connector establishes the connection between level and subset operator; more than one subset operator can be connected to one hierarchy level. The subset itself consists of one or more scopes.

Attributes are, as already stated, attached to dimensions. Alternatively, they might be linked to hierarchy levels and scopes via connectors.

Calculation models have at least two inbound members and create one calculated value. A member is allowed to take part in more than one calculation model. Calculation rules between different cubes should not be considered here.

The proposed metamodel does not support all aspects of the ADAPT notation. For example, scopes are bound to hierarchy levels via subset operators. The authors of [2] see the possibility to link scopes directly with dimensions in order to categorize individually modeled members. It is also not intended to limit the range of attributes’ possible values.

The use of identifiers should be limited. Ideally, each identifier is distinct within its object type. For instance, each dimension and each hierarchy level require different names. The same does not apply to elements with different semantics, for example both a dimension and a hierarchy can be identically named.

5. Modeling Tool Prototype

The presented prototype should be easily extendable by means of multiple database systems that can be accessed via the modeling tool. The main problem behind this thought is the existence of various target platforms which differ in the way they store multidimensional data: ROLAP, MOLAP und HOLAP. Therefore, a way of accessing multiple transformation algorithms must be found, and each algorithm has to be parameterized. The right-hand side of figure 7 summarizes this approach. In order to concentrate on this requirement we use the concept of reutilization by not inventing a new modeling language or a new editor but using ADAPT and an existing ADAPT stencil for Microsoft Visio.

5.1. Architecture concept

The prototype is based on a three-layer architecture as shown in figure 7. An editor for ADAPT is situated within the graphical or presentation layer. In our case this will be Microsoft Visio because of an already well-proven stencil. Starting with Visio 2003, it is possible to save diagrams in an XML format, DatadiagramML (formerly known as XML for Visio).

The abstraction layer acts as an intermediary between graphical modeling and transformations.
ADAPT diagrams are represented as directed typed graphs in order to reduce the loss of information between diagrams and internal storage. A typed graph $G$ can be defined as the following tuple: $G = (N, E, t_N, t_E, s, t)$ with a finite set of nodes $N$, a finite set of edges $E$, an assignment of a type to each node $t_N: N \rightarrow \Sigma_N$, as well as to each edge $t_E: E \rightarrow \Sigma_E$, and the assignment of source and target for each edge $s, t: E \rightarrow N$ [10]. The instances of $N, E, s$ and $t$ arise from the ADAPT diagram whereas $\Sigma_E$ and $\Sigma_N$ reflect the modeling elements of the notation itself.

The third layer provides transformational functionalities. Therefore, one of the most important requirements is the extensibility by means of adding new transformation algorithms without intervening into the source code of the modeling toolset.

In version 6 of the Java Standard Edition (SE), the ServiceLoader API has been made public [16]. It helps to find, load and use so-called service providers. Within the context of this API a Service is a collection of interfaces and classes that provide access to specific program functionality. A service provider reflects the actual implementation. In the case of the prototype each transformation algorithm represents a service provider. They are defined by the service provider interface (SPI), which consists of a set of public interfaces and abstract classes. Only the classes and methods contained within the SPI are visible to the actual application.

To create an SPI it is necessary to find a generic interface which offers capabilities for creating multidimensional modeling constructs. The following operations are suitable for our purpose:

- **create database <database>**: This operation creates a new database as a container for all other structures and quantitative values.
- **create dimension <dimension>**: Dimensions are essential for multidimensional data structures. Each dimension has to receive a distinct name.
- **create measure <measure>**: As a next step, measures have to be instantiated within the database.
- **create cube <name> <var1>, <var2>, ..., <varm> <dim1>, <dim2>, ..., <dimn>**: The fourth method assembles data cubes from the given dimensions and variables.

A reference to a partial graph is passed to each method.

### 5.2. Transformation Examples

The prototype is evaluated by a case study, which has been simplified for the purpose of this paper. A second production cube containing four dimensions (Product, Plant, Time and Measures) extends the example given in section 4.1. The dimensions product and time are shared by both cubes. Further research will include evaluation in real-world business applications. At the moment there are negotiations on how to integrate our software in a large project within the telecommunication industry sector.

5.2.1. Galaxy Schema. Within a galaxy schema, cubes and dimensions correspond to fact and dimension relations, respectively. They are connected via foreign key relationships [3].

<table>
<thead>
<tr>
<th>Model element</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact table</td>
<td>fact_</td>
</tr>
<tr>
<td>Dimension table</td>
<td>dim_</td>
</tr>
<tr>
<td>Primary key</td>
<td>pk_</td>
</tr>
<tr>
<td>Foreign key</td>
<td>fk_</td>
</tr>
<tr>
<td>Variable / Measure</td>
<td>m_</td>
</tr>
<tr>
<td>Hierarchy level</td>
<td>level_</td>
</tr>
<tr>
<td>Parent level of recursive</td>
<td>parent_level_</td>
</tr>
<tr>
<td>relationships (self precedence)</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>attr_</td>
</tr>
<tr>
<td>Dimension scope (connected via</td>
<td>subset_</td>
</tr>
<tr>
<td>Exclusive operator)</td>
<td></td>
</tr>
<tr>
<td>Dimension scope (connected via</td>
<td>scope_</td>
</tr>
<tr>
<td>Partially operator)</td>
<td></td>
</tr>
</tbody>
</table>

In order to identify the modeling constructs in the generated schema, prefixes which precede each identifier (such as relations or columns) are introduced. A summary is given in table 1.

A second assumption concerns the data types of each individual modeling element. Table 2 outlines them. Our future task is to use transformation parameters to implement a solution which allows a flexible setting of data types.

### 5.2.1. Galaxy Schema

<table>
<thead>
<tr>
<th>Model element</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary key</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Foreign key (has to be the same as primary key)</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Fact</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>Hierarchy level</td>
<td>VARCHAR(255)</td>
</tr>
<tr>
<td>Attribute</td>
<td>VARCHAR(255)</td>
</tr>
<tr>
<td>Dimension Scope</td>
<td>VARCHAR(255)</td>
</tr>
</tbody>
</table>

Each dimension corresponds to a relation $dim_<dimension name>$ with a serially-numbered primary key. Attributes of the dimension itself correspond to a separate column $attr_<attribute$
If there are dimension members, only one single column is generated `<dimension name>`. The members are added later on via `INSERT` statements.

According to the metamodel, hierarchies are permitted only if there are no dimension members. Hierarchy levels are represented in a column `level_<level name>`; the same holds for attributes of levels: `attr_<attribute name>`. Levels in parallel hierarchies are represented by one column for each level. Subsets connected via `Fully Exclusive` or `Partially Exclusive` result in one column per subset operator. The column’s name corresponds to the names of each scope: `subset_<name scope 1_name scope 2_ ... _name scope n>`. An alternative would be the creation of a column with a `SET` data type containing one entry for each scope. Overlapping operators (`Fully Overlapping` and `Partially Overlapping`) result in one column for each scope. This allows an element to be contained in different scopes. Recursive connections between hierarchy levels are represented by a column `parent_level_<level name>`.

Fact tables `fact_<cube name>` are connected via foreign key relationships to dimension tables (`fk_<dimension name>`). For each measure a column `fact_<fact name>` is created. An alternative would be the creation of a measure dimension; each measure would map onto one dimension element.

### 5.2.2. Common Warehouse Metamodel (CWM)

Another worthwhile approach is mapping structural information onto the CWM. The OLAP package within the analysis layer appears appropriate for this task. The assumptions in Table 3 apply to data types.

The Netbeans Metadata Repository (MDR) is an implementation of a Meta Object Facility (MOF) compliant repository. It is appropriate for generating interfaces according to the Java Metadata Interface (JMI) specification. These interfaces are used to access an extent within the repository which contains the metamodel and its instance.

#### Table 3. Data types for modeling elements within the OLAP package of CWM

<table>
<thead>
<tr>
<th>Model element</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>String</td>
</tr>
<tr>
<td>Identifying Attribute</td>
<td>Integer</td>
</tr>
<tr>
<td>Fact</td>
<td>Float</td>
</tr>
</tbody>
</table>

A schema has to be instantiated as a container for storing the model elements of the whole ADAPT diagram. Each cube and each dimension map onto the corresponding classes of the CWM. A `CubeDimensionAssociation` connects cubes and dimensions. The mapping of a dimension organizes as follows: for each dimension level an object has to be generated and has to be published to the according dimension. Hierarchies are represented as instances of the class `LevelBasedHierarchy`. Additional semantics can be given to hierarchies by using the class `ValueBasedHierarchy`. It is useful for hierarchies that have some kind of topological order within the individual levels. Because of complexity reasons, the current implementation only supports instances of `LevelBasedHierarchy`.

Despite the relatively low depth of CWM, some arguments can be found to build a conceptual metadata
schema on the basis of CWM. Some parts of the metadata can be stored according to the standard which leads to an easier exchange of metadata via XMI. Standard conformant extensions of the reference model offer the possibility to adapt it to enterprise specific needs. These extensions can be easily published and used by other CWM users. The usage of XMI offers the advantage of checking files against a document type definition or an XML schema in order to discover transmission errors.

Figure 9. Transformation steps from conceptual ADAPT towards XMI

6. Discussion and Future Work

The prototype shows the possibility of finding a generic interface for algorithmic transformation of conceptual data models based on the ADAPT notation into different logical representations. A suitable way of parameterization has to be implemented.

For an aggregation of the Java tool and Visio into one implementation the features of Eclipse Graphical Editing Framework (GEF) have to be evaluated. The research prototype GraMMi [12] offers interesting thoughts about a repository-based graphical modeling toolset. Another attractive proposition to develop multidimensional data models collaboratively within a web browser also needs to be investigated.

In this context some consideration has to be given to a quality evaluation of the data models. [11] proposes a two-dimensional approach: the first dimension is the classification concept (syntax, semantics and pragmatics), the second dimension includes measures (absolute and relative measures). Further research work needs be checked and adapted to ADAPT diagrams.

Within the transformation layer an optimization of the generated schemata has to be implemented. Much improvement can be made within a star or galaxy schema. Some suggestions are given in [5].

The prototype makes use of object-at-a-time operations. This can be understood as the transformation of the original problem into an object-oriented one and manipulating the models and mappings in that representation. To raise the level of abstraction it would be better to use model-at-a-time and mapping-at-a-time operators. They are expected to improve programmers’ productivity for metadata applications [13].

Systems supporting such functionality are discussed under the term model management systems (MMS). They support “the creation, compilation, reuse, evolution, and execution of mappings between schemata represented in a wide range of metamodels” [14]. Such an MMS is not a user-oriented tool. In fact it is a reusable component that can be integrated into specific applications relating to data programmability problems.

The most relevant operators for our problem at the current state of research are generation as well as execution of mappings. New transformations could be generated quickly, in the best case even without writing any Java code; a graphical definition would be ideal.

An important topic on our research agenda is a reverse engineering approach of existing data warehouses which is not currently implemented and will require further research. In this context, visualization of data models is not restricted to the ADAPT notation, sometimes different representations are more useful. Fact sheets that can be viewed with Excel showing a description or the calculation of a measure are one example.
7. References


