1. Introduction

The term data warehouse (DW) has been introduced by Inmon as a ‘... subject-oriented, integrated, time-oriented, non-volatile collection of data for management support’ [16]. Typically, the DW is a database held separately from operational systems. This is justified by the different requirements of functionality and performance in both types of databases: while a DW is gearing to a back end component of a decision support system with prime importance of reading access and complex queries, operative systems are optimized for multi-user access and high throughput of transactions.
Data in a DW are integrated from operational systems of an organization and probably supplemented by data from external sources. The main front end application is OLAP (Online Analytical Processing), an explorative and interactive process. It is advantageous to use the multidimensional data model for modeling DWs on a conceptual level. The most important characteristic of the multidimensional model is dividing data into facts (also called measures or quantifying data) and dimensions (also called qualifying data). To provide data on a suitable level of granularity hierarchies are built on the dimensions. The combination of facts and dimensions is often called (hyper)cube. Figure 1 explains these terms by means of an example.

Since operational and strategic decisions made on the basis of data warehouses are very important for an organization a comprehensible methodology for designing data warehouses is necessary. Within the scope of our research project ODAWA (OFFIS Tools For Data Warehousing) [15] we are developing a methodology for designing DWs. It is based on the approach of three–level–modeling well–known from building OLTP (Online Transaction Processing) databases. Moreover, we are distinguishing between language and (graphical) representation on the conceptual level and we want to offer continuous tool support for every phase of development. Figure 2 summarizes our framework.

For the conceptual level we have developed a multidimensional meta language, called MML (Multidimensional Modeling Language) [13], [14]. It has the following characteristics:

- MML is an object–oriented language and therefore provides a good basis for flexible, implementation–independent modeling.
- MML meets the needs of conceptual multidimensional models like e.g. sophisticated dimensional structures.
- MML enables schema evolution.

A fraction of the MML–class diagram being relevant later on in this paper is depicted in figure 3.

With the MML as basis different front end tools can be used. Exemplarily, we have developed \(m\)UML. \(m\)UML is an extension of UML (Unified Modeling Language) [20], [21], using stereotypes and tagged values as mechanisms of extensibility. Its implementation (see figure 4) has been done as an extension of the commercial CASE tool Rational Rose [19]. The MML is implemented as a class library in C++.

In the last decade patterns became a topic of interest in the field of software engineering. Unfortunately, 'there exists no standardization of the term design pattern in the realm of object–oriented software development' [18]. In the context of this paper patterns are characterized as follows:
• patterns are generic structures won by experience of modeling in the past.
• patterns can be of different sizes, e.g. whole reference models for a branch of industry or only part of a model.
• patterns can be abstract (describing the structure of a model) or concrete (describing one special model).
• patterns can have parameters and configuration steps; these configuration steps describe in which way and order the parameters are used.

By using approved patterns modeling errors can be avoided and the time of modeling can be reduced. In consequence of these reasons using patterns can implicate economical benefits like increasing productivity, increasing quality of software, reducing time-to-market, and reducing cost of software. Patterns can be classified by the following orthogonal characteristics:
• Level
Patterns can be defined on different levels, e.g. organization patterns or pieces of software.

• Magnitude
Patterns can be of different sizes, e.g. whole schemas (reference models for branches of industry or contexts) or simply part of a schema.

• Concreteness
Patterns can describe a concrete object or can be more abstract describing only its structure.

• Parametrizability
Patterns can either be fixed or they can have parameters determining their concrete extension.

Since MML is an object–oriented language and patterns are a powerful aspect in object–oriented software engineering we have applied patterns to our approach of conceptual data warehouse modeling. The remainder of the paper is organized as follows: in section 2 we enrich our framework for conceptual multidimensional modeling by adding the possibility to define, administrate and use parametrized patterns. The pattern language PL is introduced and demonstrated by examples in section 3. After reflecting related work, the paper concludes with a summary and an outlook.

2. Parametrized Patterns for Conceptual Multidimensional Modeling

2.1 Pattern Model
The model for parametrized patterns to support multidimensional modeling is depicted in figure 5 (all associations without multiplicities in the diagram are many–to–many).

![Diagram](image)

Figure 5. Model for Parametrized Patterns

The most important class Pattern is described by the following attributes:

• Name: name of the pattern.
• **Synonyms:** pattern is also known as these.

• **Purpose:** why is this pattern defined.

• **Examples:** contexts of use of the pattern.

While these attributes describe the functional and structural aspects of a pattern the classes `Parameter`, `Step` and `MML-Script` specify the technical facet of the pattern. Each pattern has got a list of parameters. Each parameter has got a data type (class `Parameter–Data–Type`) and a domain (class `Parameter–Domain`). A domain is a collection of intervals of the specified type (class `Parameter–Domain–Part`). The model considers the data types string and cardinal and arrays of these basic types. Beside the standard operations (e.g. addition on cardinals or concatenation on strings) for arrays of cardinal there are defined the functions `max` and `min` returning the maximal resp. minimal entry of the array.

For specifying the configuration process of a pattern an ordered list of steps is assigned to each pattern. The association between the classes `Parameter` and `Step` defines which parameter can be determined by the user at which step. Instantiating a pattern is traversing these steps in the defined order. After the last step of this configuration process the MML-Script associated to the pattern is executed with the chosen parameters. A MML-Script is a sequence of statements. Statements can be either simple constructor calls or assignments to variables or they can be more complex in form of while–loops or if–then–else–constructs. Both last–named statements use expressions over the parameters as arguments. When the execution of a MML–script ends all created objects are made implicitly persistent in the MML–repository.

For better handling of large sets of patterns the model offers the following:

• It is possible to organize sets of patterns in libraries (class `Pattern–Library` and association to `Pattern`).

• The self–referencing association `Pattern–Relationship` contains pairs of patterns and describes their similarities and differences in verbal form. There are three kinds of relationships between two patterns: simple relationship, part–of and specialization.

• The classes `Pattern–Category` and `Pattern–Category–Extension` enable the classification of patterns according to the classification given in the introduction. Moreover, this classification schema can be extended by the user by defining new categories respectively adding extensions to any category.

• Assigning headwords to patterns is similar to categorizing but the designer has all the freedom possible because entries can be free text and are not limited to any categories. By using headwords the user can build a subject catalogue on the patterns.

It is also possible to create users (class `User`), combine them to groups (class `Group`) and restrict their access to selected patterns only (class `Access–Right`). In case of conflict the rule *lex specialis ante lex generalis* is in force, e.g. more special rights to a user overlies the rights of his group.

### 2.2 Implementation

In this section our architecture depicted in figure 4 is extended by the model introduced in the previous section. This extended architecture is depicted in figure 6. We distinguish between two kinds of users: a designer who defines the patterns and a modeler who defines multidimensional schemas by using the patterns. Both of them can communicate with the system via a GUI or a script language. Analogous to
distinguishing users we differentiate between DPL (designer’s pattern language) and MPL (modeler’s pattern language). Both languages use a class library realizing the model from the previous section. The object persistence management layer and the database driver can be reused. The patterns are stored in a repository persistently. The pattern and MML repositories can coincide on the physical level.

![Diagram showing the environment for pattern use](image)

**Figure 6. Environment for Pattern Use**

### 3. Pattern Language PL

The pattern language PL is composed of the DPL for the pattern designer and the MPL for the modeler.

#### 3.1 Designer’s Pattern Language

DPL provides the following statements:

- CREATE, UPDATE and DELETE PATTERN `<pattern>` for creating and manipulating patterns.

- CREATE USER, DELETE USER, ADD `<user>` TO GROUP `<group>` and GRANT/REVOKE `<access–right>` ON `<pattern>` TO `<user>` | `<group>` for user and right administration.

- CREATE LIBRARY `<library>` and ADD `<pattern>` TO LIBRARY `<library>` for organizing patterns in libraries.

- CREATE CATEGORY `<category>`, INSERT `<extension>` INTO CATEGORY `<category>` and ASSIGN `<category>..<extension>` TO `<pattern>` for handling the classification of patterns within the schema of categories.
• ASSIGN HEADWORD <headword> TO <pattern> for handling classifications of patterns with free text.

With the CREATE PATTERN–statement as the most complex one the designer can create new patterns. The following example should demonstrate its use:

**CREATE PATTERN** 'Multiple Hierarchies'
**SYNONYMS** 'Simple Hierarchy with many roll–up paths'
**PURPOSE** 'A typical dimensional structure able in many cases'
**EXAMPLES** 'Hierarchy of products in a sales cube; rolled up by product groups and brands'
**PARAMETERS**

- ('Name of the dimension', Dim-Name, STRING),
- ('Number of hierarchies', N, CARDINAL, [1..100]),
- ('Number of levels in hierarchy $I$, M[I]', ARRAY OF CARDINAL, [1..100], I=[1..N]),
- ('Name of level $I$ in hierarchy $J$, Lev-Name[I][J]', ARRAY OF STRING, I=[1..max(M)], J=[1..N]),
- ('Name of top–level', Top-Lev-Name, STRING),
- ('Fact to connect', Fac-Name, STRING)

**STEPS**

1. `'Defining name of dimension','What is the name of the new dimension?', Dim–Name),
2. `'Defining number of hierarchies','How many hierarchies should be defined?', N),
3. `'Defining number of levels','How many levels does hierarchy $M[I]$ have?', M[I]),
4. `'Naming the levels','What is the name of level $I$ in hierarchy $J$?', Lev–Name[I][J]),
5. `'Naming the top-level','What is the name of top-level', Top-Lev-Name),
6. `'Defining corresponding fact','To which fact will the dimension be connected ?', Fac-Name),

**MML-Script** <Path-Of-File>;

In this example a pattern called 'Multiple Hierarchy’ is defined. The clauses SYNONYMS, PURPOSE and EXAMPLES are descriptions of the pattern. With these attributes the designer characterizes the pattern and later the modeler identifies the patterns appropriate for his use by means of these descriptions. Thus these attributes serve as a communication interface between designer and modeler. After the keyword PARAMETERS the parameters of the pattern are listed: their name, data type, domain of the parameter and in case of arrays the range of the index variables.

The components of each step are sequence number, name, dialogue question to the modeler and the parameter to be determined in this step. A dialogue question can contain parameters. They are included by '$'. The MML–script clause references a file. In our example the MML–script is defined as follows:

```mml
Dimensional-Class(Dim-Name);
Dimension(Fac-Name,Dim-Name,Dim-Name);
I=1;
while (I<=N)
J=2;
while (J<=M[I]+1)
Dimensional-Class(Lev-Name[I][J-1]);
if J=2 then
Roll-Up(Dim-Name,Lev-Name[I][J-1],Lev-Name[I][J-1]);
else
Roll-Up(Lev-Name[I][J-2],Lev-Name[I][J-1],Lev-Name[I][J-1]);
end
J=J+1;
end;
```
In the first two lines the root of the dimension is defined and connected to the fact. The outer while–loop runs over the different hierarchies and the inner one over the levels of each hierarchy. For every level a dimensional class is created. To get a connection between these classes a roll–up path is created.

To provide more ergonomy, on top of DPL a GUI (graphical user interface) should be realized. Figure 7 shows the specification of a screen for manipulating a pattern definition.

![Figure 7. Designer’s GUI for manipulating patterns](image)

### 3.2 Modeler’s Pattern Language

The most important statement of MPL is INSTANTIATE PATTERN. In the script language the parameters must be indicated, e. g.

```
INSTANTIATE PATTERN 'Multiple Hierarchies'
PARAMETERS ((Dim-Name='Product'),
            (N=2),
            (M[1]=1,M[2]=2),
            (Lev-Name[1][1]='Product Groups'),
            (Lev-Name[2][1]='Brand'),
            (Lev-Name[2][2]='Producer'),
            (Top-Lev-Name='All Products'),
            (Fac-Name='Sales'));
```

Much more user–friendly is configuring a pattern with the GUI of MPL. This GUI is dynamically in the way that the dialogue windows are created during the process of instantiation by using the parameters and the configuration logic defined by the steps in the CREATE PATTERN statement. The process of instantiation of our example with the dynamic GUI is shown in figure 8.

After executing the INSTANTIATE PATTERN–statement or finishing the GUI–dialogue the MML–script of the pattern *Multiple Hierarchy* is executed with the desired parameters and finally all
Figure 8. Part of dialogue sequence of modeler’s GUI

instantiated objects are made persistent in the repository. The modeler can observe the result after refreshing in Rational Rose. Individual modifications and additions are possible. The result of our example can be viewed in figure 9 in notation of the mUML. The result is a multiple hierarchy on dimension 'Product' depicted in the example in figure 1.

3.3 Browsing the pattern library

In this subsection browsing large pattern libraries is illustrated. After determining his search criteria the user should be presented a result like that one depicted in figure 10. In the right part the search criteria are repeated and the resulting patterns are listed. If the user selects one of it this pattern and its relationships to other patterns are represented in the left part. Furthermore, the transitive closure of pattern relationships is shown. In this manner the modeler can explore the collection of patterns. By dragging the mouse pointer over a pattern additional information about this pattern should be displayed.

4. Related Work

In the area of data warehouses some approaches for conceptual multidimensional modeling have been developed, namely MERM (Multidimensional E/R Model) [22], ADAPT (Application Design for Analytical Processing Technologies) [3] and DFM (Dimensional Fact Model) [12]. But they have some deficits:

- No adequate expressiveness, especially for modeling sophisticated dimensional structures.
• No compatibility among each other.
• No support by tools.
• No strict distinction between design levels.

On the other hand some commercial tools for designing DWs are available but they are most often proprietary. Hence conceptual modeling is often left out in practice or the conceptual and logical levels coincide, e. g. applying Kimball’s dimensional modeling [17].

Patterns have their origin in architecture: the aim to build high–quality homes and to make urban planning with reusing patterns was described in [1] and [2]. With the emergence of object–oriented design of software and object–oriented programming languages in the late eighties and early nineties patterns also became relevant in this field [5]. In [11] a team of authors called ‘Gang of Four’ published a collection of patterns and leveraged the idea in the community. Since then patterns have been used for many different domains ranging from organizations and processes to teaching and software architecture. Another approach in the area of patterns is made by [4]. In opposite to the ‘Gang of Four’ the authors wanted to provide a system of patterns instead of just a pattern catalog. This aim should be reached by having an unified description of patterns, by describing relationships between patterns, by classifying patterns and last but not least by providing search strategies to support the user in finding the right pattern. As a conclusion we can say that at present the software community is using patterns largely for software architecture, design and software development processes.

Important references concerning relevant topics about patterns are [6], [7], [8], [9], [10] and [23]. The idea of designing parametrized patterns by an interactive dialogue with the user can also be found in [24].

5. Summary and Outlook

In this paper we have suggested a concept for using parametrized patterns to support conceptual multidimensional modeling. After giving a brief introduction we proposed a model for parametrized patterns. Then we showed how our current architecture can be extended by implementing this model. Afterwards, we distinguished between two kinds of the users: designers (who are defining patterns)
and modelers (who are using patterns to create multidimensional schemas). For both groups we have introduced both a script language and a GUI.

As future tasks we want to do the following:

- evaluating our approach in a real world project,
- constructing libraries of patterns,
- defining patterns by a graphical editor,
- supporting the modeler in the way that he can trace his work graphically when using the dynamic GUI,
- expanding the library by so-called anti-patterns which describe a bad solution from the past.

6. References


