A Case Study of Recursive Data Modeling

Fazat Nur Azizah, Guido Bakema, Jan Pieter Zwart
Research and competence group Data Architectures and Metadata Management
Academy for Communication and Information Technology
HAN University of Applied Science, The Netherlands
June 2006

Summary

In the programming area much attention is given to recursion, but this technique is hardly considered in the data modeling area. This paper discusses a case study about storing project data of a knowledge-based application called Quaestor, in which the modeling of a recursive data structure is the central issue. Data modeling methods such as entity-relationship modeling or object oriented modeling are not yet equipped with concepts for dealing with recursive data structures. Some fact oriented modeling methods however can deal with recursion. In this case study, Fully Communication Oriented Information Modeling (FCO-IM) is used to capture the complex identification problem of the recursive structure of Quaestor project data.

1. Introduction

In the programming area, much attention is given to recursion, but this technique is hardly discussed in the data modeling area. A recursive structure is a structure defined in term of itself. For a procedure this means that it contains, within its body, a call to itself, whereas for a data structure[1] this means that its identifier is defined in term of itself. Many data modeling techniques, for example entity relationship modeling and object oriented modeling, are not yet equipped with concepts for dealing with recursive data structures. In entity-relationship modeling some people mistake a cyclic structure (i.e. a structure in which an entity type is involved in a relationship with itself) for a recursive structure[2].

This paper discusses a case study of modeling project data from a knowledge-based application called Quaestor, in which a recursive data structure is the central issue. This case study deals with a complex recursive identification structure. Previously, an attempt was made to model this structure, but this failed because the data modelers did not succeed in capturing the essence of the problem. In this case study, the fact oriented method Fully Communication Oriented Information Modeling (FCO-IM) was chosen since it has proved in the past to be able to describe complex recursive structures correctly.

[1] The term "data structure" is used here in the sense of the data modeling area, not in the sense of the programming area. For the definition of "data structure" in the programming area, see for example http://www.nist.gov/dads/HTML/ recursivstrc.html.
[2] See for example http://www.dbmsmag.com/9506d16.html and http://www.utexas.edu/its/windows/database/datamodeling/dm/schema.html.

2. Quaestor

Quaestor[3] is a knowledge-based application for assembling computational models, developed by Qnowledge Modeling Technologies, Wageningen, The Netherlands. The application helps engineers to design their computational models. Quaestor was developed based on the idea that computational models can be represented and manipulated as abstract structures that can be assembled from smaller elements (*parameters*) and/or model fragments [5].

When using Quaestor, the user starts by creating a knowledge base system (KBS for short). In a KBS, the user creates all parameters and other elements related to the domain to be modeled. Based on this KBS, the user can create projects. In a project, the user can construct computational models and store them. These are called *project data*.

With the increasing number of models built with Quaestor, a wish arose to be able to reuse former project data for building new computational models, to reduce the development time of new models. As a consequence, it was needed to store and maintain all project data in a single database that can be accessed by Quaestor and other applications. The project data are currently stored in a generic format called *telitab* format. This telitab format includes recursion, which must be properly covered by the database of Quaestor project data.

Telitab is the abbreviation of *text-list-table*, because the format consists of three main parts: text, list, and table. Telitab is a generic format that is used by Quaestor to communicate model parameters within itself (for data storage and retrieval functions) or with satellite applications. An example of a telitab is shown in figure 1.

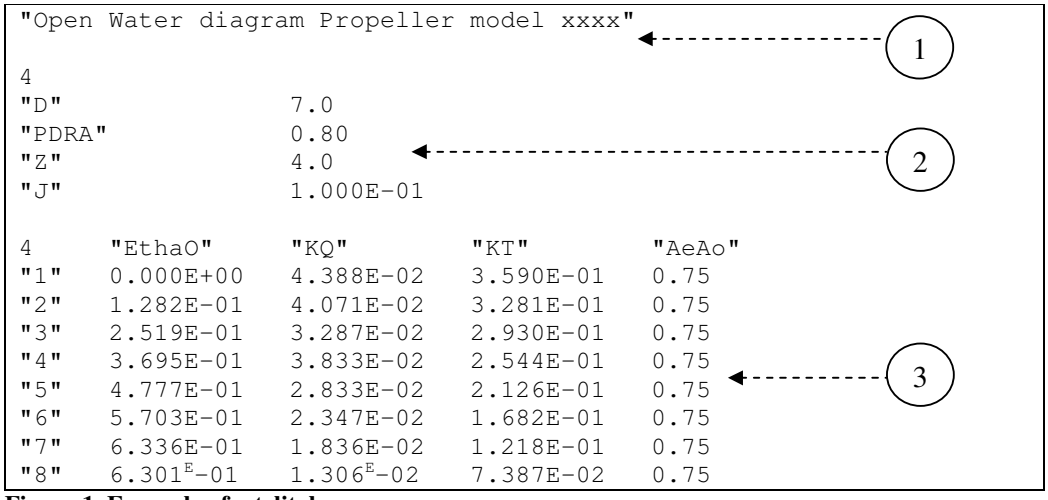

Figure 1. Example of a telitab

Text (no. 1 in figure 1) is unstructured information without parameters. The parameters involved in the model are described by means of a number of pairs of a

[3] Quaestor was initially developed for the naval company Marin in The Netherlands for assembling and executing computational models related to ship design. See http://www.qnowledge.nl/.

parameter name and its value, also called a *Parameter Value Combination* (PVC). The PVC's are written in a *list* (no. 2 in figure 1). Names of parameters are placed in quotation marks, followed by the value of the parameters. In a *table* (no. 3 in figure 1), the columns represent a set of parameters. Each entry in the table represents one PVC. Each row in the table represents one *case* of each set of PVC's. Each case is identified by a number (in the leftmost column).

3. The Project Data Analysis[4]

Along with the development of the database for storing Quaestor project data, a functionality to export telitab format to XML format has been added to the latest version of Quaestor. The general scheme of the XML transformation of a telitab structure of a Quaestor project data is shown in figure 2. Note that there is some additional information in the XML file that is not present in the original telitab structure shown in figure 1. This information is related to properties of parameters involved in the telitab, and of the telitab itself as well.

The XML file that stores the Questor project data is the source of input data for the database of Quaestor project data. Therefore, this file became the main basis for the analysis of the database, along with the study of the original telitab format, other additional documentations and interviews with domain experts.

Since all project data are created based on a KBS, information about this KBS must be stored as well. A KBS is identified by its name[5]. In a KBS, several parameters are defined. Each parameter has a name. Since the same parameter name can appear in different KBS's, a parameter is identified by the combination of its name and the KBS that contains it. Each parameter has properties: class, address, data, dimension, input, properties, and reference (see no. 2 in figure 2).

Based on the KBS, the user can create one or more projects, which are identified by the combination of the project name and the KBS name. Each project is stored in a telitab. A telitab that contains a whole project is the highest level telitab, also known as the primary telitab (see no. 1 in figure 2). A telitab is identified by its name and the project it belongs to. In the case of a primary telitab, this name is usually – not always – the same as the name of the project.

Each project can use the parameters defined in the respective KBS. A parameter that is applied in a project is addressed as a project parameter to distinguish it from the parameters of a KBS. A telitab consists of one or more project parameters. Therefore, a project parameter is identified by the combination of the identifier of the parameter it corresponds to and the identifier of the telitab that contains the parameter.

[4] This description is an adaptation from its original version in [1] chapter 4.
[5] The name of a KBS is not present in the XML document containing the project data. It is found in the file name of the XML file. This file name is designed to contain the name of the KBS as well as of the project in this format: *<KBS-name>.<project-name>.xml*.

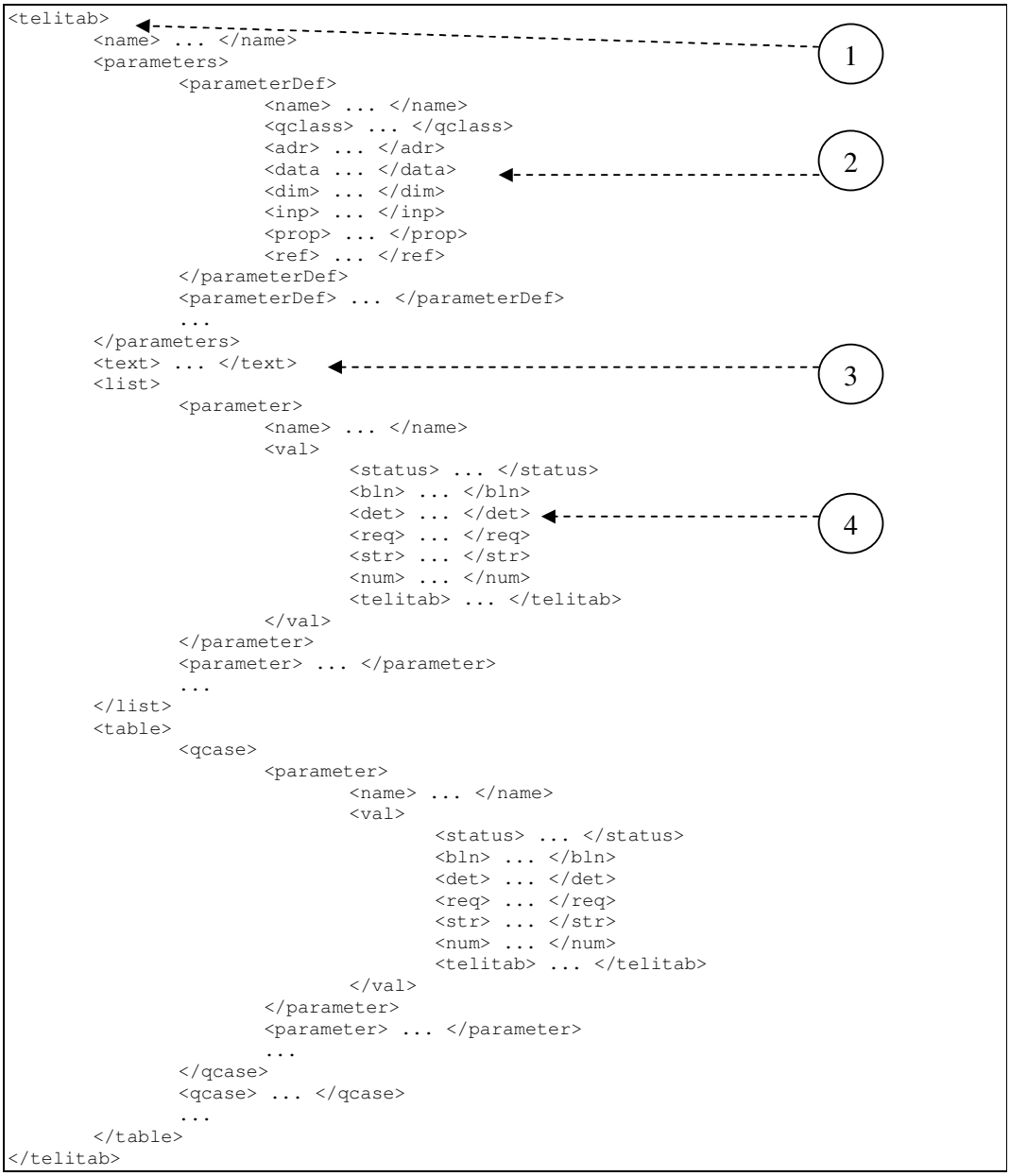

Figure 2. General scheme of XML file containing telitab structure

A project parameter must have a value assigned to it. This value can either be simple or compound. In a telitab, simple values are stored in a list, whereas compound values are stored in an n-x-m table (n the number of project parameters involved in the table and m is the maximum number of cases). Therefore, the case number is a part of the identifier of an individual value of a project parameter in a table. Each project parameter value has several properties (see no. 4 in figure 3): status, boolean value, address of parameter which determines the value, and address of parameter that requires the value. The value itself can be either a string, a numeric value, or a telitab. If the value of a project parameter is a telitab, then a lower level telitab is present, which has exactly the same structure as the higher level one,. This lower level telitab can contain project parameters applied in other telitabs of the same project.

A lower level telitab is identified by its name and the project parameter that contains it as a value. The name of such a lower level telitab is usually the same as the name of the parameter.

A telitab also has a text part which describes any additional information related to it (see no. 3 in figure 2).

Modeling information about a KBS and its parameters is not difficult, since the above description provides clear information. The same holds for a project. The main problem in modeling Quaestor project data lies in the modeling of telitabs and their project parameters. At least two approaches came up to solve the problem:

1. **Emphasis on the structure of the telitab**
   The first idea is to model the structure of the telitab as it is described. So, a telitab contains several project parameters and each project parameter has values which can be a numeric value, a string, or another telitab.

2. **Emphasis on the structure of the project parameters**
   If the telitab format is seen just as a kind of container for the project parameters, it is clear that the structure of the hierarchy of the project parameters is a *tree*. A telitab can then be understood as an encapsulation of the value of a project or a project parameter. The root of the tree is the project, followed by nodes which are the project parameters (see figure 3).

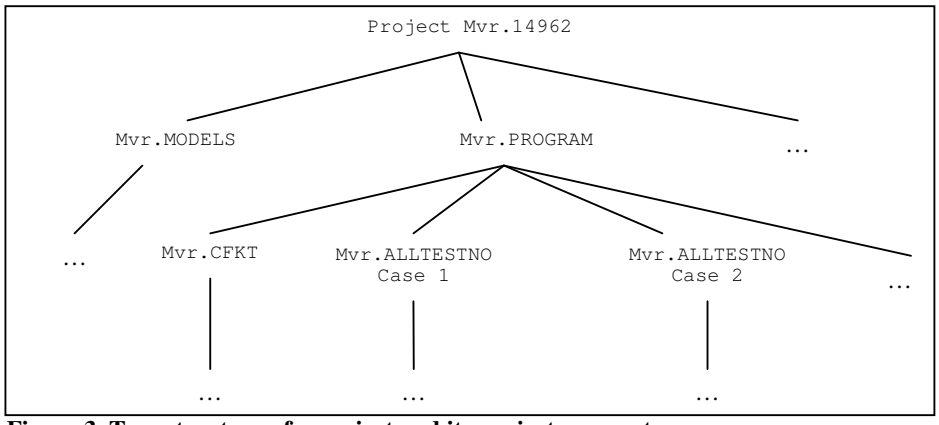

Figure 3. Tree structure of a project and its project parameters

The second idea seems preferable because there is no need to store the telitabs explicitly. The telitab format is just considered as a means of storing the hierarchical relationships between the project parameters. If in future telitabs would be replaced by another way of storing this hierarchy, then the data model would be independent of such a change. However, this approach was rejected for three reasons. Firstly: there are at least two properties related to telitabs only: name and text. Secondly: we wanted to respect the preference of domain experts who are used to deal with their models in a telitab way of thinking. Finally, it is not likely in the near future that the telitab format will be replaced by another format. Therefore, we chose the first approach, and our further discussion focuses on this.

Both approaches however involve recursive structures. In the first approach, a telitab contains several project parameters and a project parameter can have a telitab as its value. In the second approach, the recursion appears in the structure of the hierarchy of the project parameters.

In the context of data modeling, recursion can be understood as a special case of generalization (see [3]). A typical problem related to generalization is different ways of identifying different objects of the same type. As discussed before, every telitab is identified by one of two following ways:

- in the case of a primary telitab, it is identified by its name and the project it belongs to (a project is identified by a project name and a KBS name).
- in the case of lower level telitabs, it is identified by its name and the project parameter it is a part of (a project parameter is identified by the parameter it corresponds with and the telitab the project parameter belongs to).

Things are even more complex because there are two types of project parameters: the ones that are stored in a list and the ones that are stored in a table. Only a value stored in a table needs a case number as part of its identifier. So there are two subtypes of project parameter, and each subtype can contribute in the identifier of a telitab. Therefore, apart from recursion and generalization, also specialization (or subtyping, see [3]) is involved.

This complex identification problem cannot be modeled using an entity relationship or object oriented approach, because these approaches do not support recursion and generalization. UML (an object oriented method) uses the term generalization where specialization is actually meant. At the moment, only the fact oriented modeling methods FCO-IM [2][3] and PSM [6] support recursive identification structures and distinguish properly between generalization and specialization. For this case study FCO-IM (Fully Communication Oriented Information Modeling) was chosen because FCO-IM is supported by a modeling tool[6], whereas PSM yet is not. Moreover FCO-IM has repeatedly proved to be able to model complex recursive structures in operational practice [2].

FCO-IM modeling starts with verbalizing facts. These verbalizations are then further analyzed, resulting in a conceptual data model. This model can then be automatically converted to a logical model (entity-relationship model or class diagram of UML) and/or to a physical model (relational model) [3].

In the Quaestor case study, verbalizing all types of facts helped to survey all problems with respect to identification. Table 1 provides some examples of verbalizations of a KBS, parameters, a project, telitabs, and project parameters.

In the examples of verbalizations of telitabs and project parameters, the recursion is clearly visible. Take for example verbalizations 8 and 9 in table 1. These examples refer to telitabs that are identified by their names and the project parameters that have them as their values.

[6] See for the FCO-IM modeling tool CaseTalk: www.CaseTalk.com.

Table 1. Examples of verbalizations

	[KBS]
1	"There is a knowledge-based system Mvr."
	[Parameter]
2	"There is parameter ALF in knowledge-based system Mvr."
3	"There is parameter MODELS in knowledge-based system Mvr."
4	"There is parameter X in knowledge-based system Mvr."
5	"There is parameter Y in knowledge-based system Mvr."
	[Project]
6	"Project 14962 in knowledge-based system Mvr exists."
	[Telitab]
7	"Telitab 14962 of project Mvr.14962 exists."
8	"Telitab MODELS of parameter Mvr.MODELS in telitab 14962 of project Mvr.14962 exists."
9	"Telitab X of case 1 of parameter Mvr.X in telitab MODELS of parameter Mvr.MODELS in telitab 14962 of project Mvr.14962 exists."
	[Project Parameter]
10	"Parameter Mvr.MODELS in telitab 14962 of project Mvr.14962 exists."
11	"Parameter Mvr.ALF in telitab MODELS of parameter Mvr.MODELS in telitab 14962 of project Mvr.14962 exists."
12	"Parameter Mvr.Y in telitab X of case 1 of parameter Mvr.X in telitab MODELS of parameter Mvr.MODELS in telitab 14962 of project Mvr.14962 exists."

The project parameter in example 8 is stored in a list since it has no case number as a part of its identification.

"Telitab MODELS of parameter Mvr.MODELS in telitab 14962 in project ...."
Telitab Name *Project Parameter*

The Project Parameter part of this example contains another telitab, making its identifier recursive:

"... parameter Mvr.MODELS in telitab 14962 in project Mvr.14962 .... "
Parameter *Telitab*

The Project Parameter part in example no. 9 is stored in a table since its identification requires a case number.

"Telitab X of case 1 of parameter Mvr. X of telitab MODELS of ...."
Telitab Name *Project Parameter Case*

The Project Parameter Case part of this example contains a project parameter that is a part of another telitab, making its identifier recursive.

From these examples, it is clear that verbalizations 8 and 9 are recursive. The same holds for verbalizations of project parameters in examples 10-12.

4. The Conceptual Data Model

The relevant part of the FCO-IM conceptual model for the Quaestor project data is shown in figure 4. It contains the central object types *KBS*, *Project*, *Telitab*, *Parameter*, *Project Parameter Case* and *Project Parameter*, which has two subtypes: *Project Parameter in List* and *Project Parameter in Table*.

To highlight only the essential part, many constraints are not shown, apart from uniqueness constraints (depicted by horizontal double pointed arrows), which are vital for identification. One totality constraint (depicted by a large dot between the two subtypes) is included as well to make clear that every instance of *Project Parameter* must be either a *Project Parameter in List* or a *Project Parameter in Table.*

The three uniqueness constraints of object type *Telitab* in figure 4 indicate the generalized identification structure: there are three different ways to identify a telitab. In other words: the primary key of object type *Telitab* is either (role 6 + role 7) or (role 6 + role 8) or (role 6 + role 11). Three of the four roles that form the object type are optional (marked by OP).

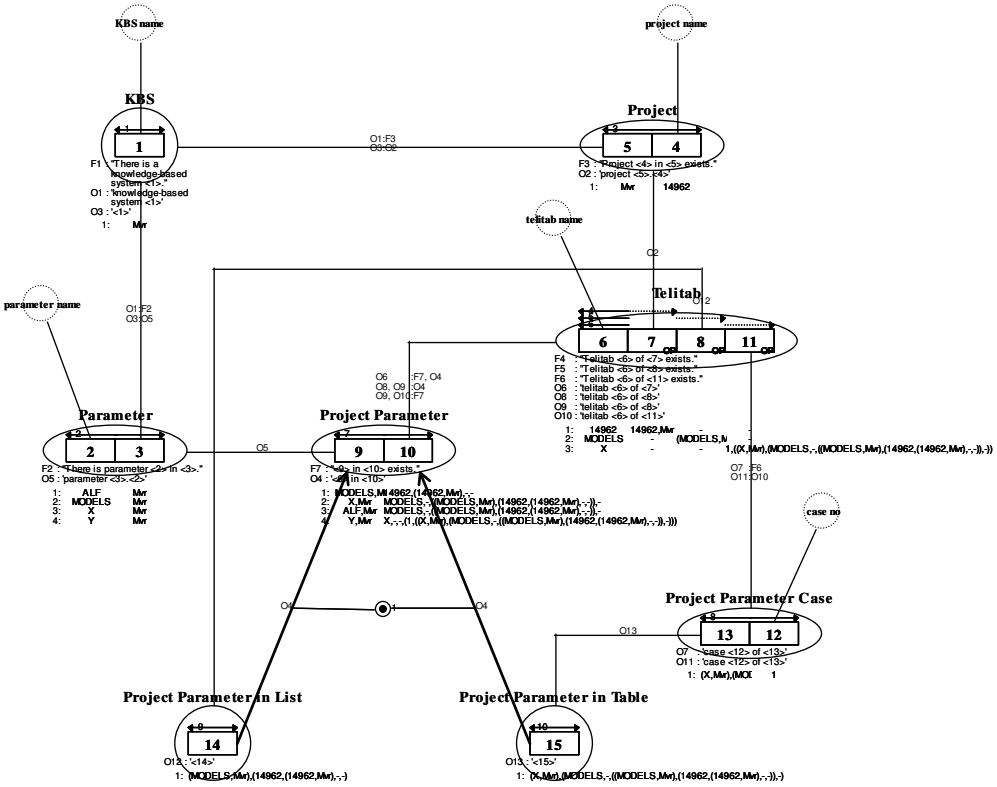

Figure 4. Conceptual FCO-IM data model

The recursions in figure 4 may not be very clear on first sight, because they are indirect:

- Role 8 of object type *Telitab* is played by *Project Parameter in List*, which is a subtype of *Project Parameter* which is itself partly identified by *Telitab*.
- Role 11 of object type *Telitab* is played by *Project Parameter Case*, which is partly identified by *Project Parameter in Table,* which is a subtype of *Project Parameter,* which is partly identified by *Telitab*.

As mentioned, the main purpose of this case study is to build a working relational database. However, presently there is no support from RDBMS vendors for such complex structures. Therefore the conceptual model cannot be directly implemented.

By using a method called nicknaming, recursive and generalized structures can be transformed algorithmically into another conceptual model without generalization (and recursion), which can be converted to a relational database schema. This method involves changes in the verbalizations of the facts. The underlying idea is to give a nickname (meaningless key or index) to each element in a recursive structure or generalized object type, so that each can be identified independently. In this case, each telitab is given a positive integer number (1, 2, 3, etc.) as an index. It is then the task of the user interface to hide these artificial numbers and to present the real recursive structure to the user. See [2] and [3] for some notes on this subject.

5. Conclusions and Future Works

Recursion and generalized data structures are encountered many times in practice. For another example of recursion and generalization in The Dutch Railway Company, see [2]. As in the Quaestor case, the real nature of the conceptual structure could not be easily comprehended. In both cases, modeling in FCO-IM immediately gave insight because this method requires the data modeler to observe the case from a semantic and elementary point of view.

It seems that the use of inadequate data modeling methods causes many data modelers to have difficulty in understanding generalized and recursive identification structures at the type level, even though they can recognize such structures at the instance level. The Quaestor case study showed this once again because data modelers did not succeed to grasp the essence of the Quaestor project data, using modeling methods that do not include generalization and recursion.

Data model patterns is an emerging field of research in the data modeling area. The idea of finding patterns in data models comes from the idea of reusability in the information system development area. Data model patterns are found by studying similarities among existing models and/or by finding the best solution from two, three, even hundreds of correct data models of a particular problem. This case study shows that it is also interesting to study whether structure patterns such as generalization and recursion can be applied.

There are a number of data model patterns researches that give interesting results (see [4] for example), but it seems that most of these attempts seek patterns from the perspective of the content (terminology used, the semantics of objects defined and their relationships, type of area involved, etc.) or in other words: at the instance level. To use such patterns a designer must find which of these patterns has the same (or mostly similar) content as the system he/she is modeling. Once such a pattern is found, the modeling process can presumably be done very quickly. It is more interesting however, to seek patterns from the point of view of the structure of the patterns themselves. An instance level approach might not be very useful if the new case cannot be matched to any existing pattern (or perhaps uses only small portions of these patterns). The Quaestor case study discussed in this article is an example of this.

Perhaps a meta level of patterns abstracted from existing patterns (and other cases which are not included in current patterns) can be found. This hypothesis will be studied further.

6. References

[1] Azizah, F.N.: "Development of Database Server for Quaestor Project Data", Master Thesis of Study Program Information Systems Development, HAN University, Arnhem/Nijmegen The Netherlands, 2005

[2] Bakema, G., Zwart, J. P., van der Lek, H.: "Fully Communication Oriented NIAM", NIAM-ISDM 1994 Conference, Working Papers, Albuquerque, New Mexico USA, 1994

[3] Bakema, G., Zwart, J. P., van der Lek, H.: "Fully Communication Oriented Information Modeling (FCO-IM)", HAN University, Arnhem/Nijmegen The Netherlands, 1999

[4] Hay, D.C.,: "Data Model Patterns", Dorset House Publishing, New York, 1996

[5] Hees, M. Th. Van: "Knowledge-based Computational Model Assembling", paper on SCSC (Summer Computer Simulation Conference) 2003, http://www.qnowledge.nl, access on June 2005

[6] Hofstede A.H.M. ter, "Information Modelling in data intensive domains", Radboud University Nijmegen The Netherlands, PhD thesis, 1993.