A Metamodel for Functional Dependencies
Towards a Functional Dependency Model Transformation

Manuel Enciso, Carlos Rossi and Antonio Guevara
Dpto. Lenguajes y CC de la Computación, Universidad de Málaga, Málaga, Spain

Keywords: Model Driven Engineering, Metamodel, Model Transformation, Functional Dependencies, Logic.

Abstract: Model driven engineering has been shown to be a useful framework to enrich the quality of software. Metamodeling and model transformation have opened the door to specifying data models and manage it in a formal and solid way. These favourable features are particularly welcome in collaborative development, where we need a data model suitable for specifying information from different sources, and which can also facilitate the integration of this heterogeneous information to a global data model. In this paper we introduce a metamodel based on the notion of functional dependencies and we propose to use model driven engineering for the development of model transformation based on the SLFD logic.

1 INTRODUCTION

Our work aims to improve the quality of real-world software projects. More specifically, we optimize models using formal techniques, but decreasing the development cost by means of automatization. In this sense, we also reduce the usual rejection that software developers have to these techniques due to their complexity and high development time and cost. We achieve these advantages by applying Model Driven Engineering (MDE). Within this overall objective, this paper focuses on the optimization of data models.

The Unified Modeling Language (UML) (OMG, 2001) provides a standard for expressing data models. It has been used in databases to specify the two main approaches in data modeling (conceptual and logical modeling): the entity/relationship model (Gogolla, 2005) and the relational model (Akehurst et al., 2002), (Laleau and Polack, 2001). Nevertheless, the introduction of model driven engineering and the use of metamodels and model transformations has opened the door to the incorporation of several languages and techniques to software engineering. Particularly we are interested in the use of logic as a framework which provides formal specification and solid inference.

In this work we propose to incorporate the good result provided by the Simplification Logic for Functional Dependencies (SLFD Logic (Cordero et al., 2002)) and use this logic as the basis for model transformation. SLFD enables building specifications using the functional dependency concept as its main element. As usual, we consider a table as a subset of the cartesian product of a set of attributes. The functional dependency $X \to Y$ is defined between two sets of attributes (X and Y) and it is fulfilled in a relation if for every two tuples, if they agree in X, they also agree in Y.

Functional dependency is a main issue in the definition of the relational model. It is the basis for the Primary Key and Unique Constraint definitions. The functional dependency also allows the specification of a dependency relationship among attributes of a relational schema. This dependency information is used to optimize relational database models using the normalization theory.

Normalization was introduced to depurate relational schemas (Garcia-Molina et al., 2008). We will focus on the normalization based on functional dependencies. It provides a set of normal form definitions (second, third and Boyce-Codd normal form) and their corresponding decomposition processes. Each normal form is defined to avoid some update anomalies and its decomposition splits the table to two new tables preserving the functional dependencies. There is in the literature a great number of works which focus on the automatization of the normalization process and, more specifically, there are some proposals in the framework of the model-driven engineering (Akehurst et al., 2002). In these papers, the authors define a metamodel for the relational model which incorporates functional dependencies specifi-

Enciso M., Rossi C. and Guevara A..
A Metamodel for Functional Dependencies - Towards a Functional Dependency Model Transformation.
DOI: 10.5220/0004129002910296
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 291-296
ISBN: 978-989-8565-19-8

cation. The authors use the Object Constraint Language (OCL) (OMG, 2001) to express functional dependencies between attributes and the normal form definitions given above on classes at a metalevel.

In this work we propose the use of a metamodel which incorporates functional dependencies considered among the whole set of attributes, even if they appear in different relational tables. This feature allows us to use the metamodel as a framework for integration. Our main goal, as we will describe presently, is the incorporation of this metamodel into the collaborative tool CBD[1] (Cordero et al., 2010), in the context of a general redesign of CBD guided by the principles of MDE. The functional dependency metamodel and its model transformations will be a powerful tool to manage the heterogeneous information provided by the collaborative use of CBD.

For the implementation of the metamodel and model transformations we use well-known frameworks and languages such as EMF (Steinberg et al., 2009) and ATL (Jouault and Kurtev, 2006).

Therefore, the results of this work provides the following usage context: the user intuitively designs with CBD the forms he needs.Then CBD generates models that conform to a metamodel including the concept of functional dependency. By means of model transformations (based on logic SLFD) CBD obtains optimized data models and generates the scripts of the database of the final application. This usage context is depicted in figure 1.

This paper is organized as follows: first we present the background about CBD and the SLFD Logic. In Section 3 we present the metamodel for functional dependencies. The model transformation induced by the functional dependency algorithms based on SLFD will be presented in Section 4, where we will provide a specification of the main inference rule of SLFD using ATL (Jouault and Kurtev, 2006). The paper ends with a conclusion and future work section.

2 BACKGROUND

In this section we present some results that may be considered as preliminaries for the development we present in this paper. We introduce CBD (Cordero et al., 2010) and the Simplificaction Logic for Functional Dependencies (Cordero et al., 2002).

2.1 SLFD Logic

The use of logic to manage functional dependencies is not new. Since the introduction of the so called Armstrong's Axioms (Armstrong, 1974) in the mid 70's, there has been a lot of works which have followed this logic approach (see (Cordero et al., 2002) for further details). Nevertheless, none of this work has been incorporated to software engineering tools. As presented in (Enciso et al., 2011), the use of functional dependencies is only a matter of research and no commercial development tool or DBMS make much use of them. As the authors cited in that paper "*Classical FDs logics, based on Armstrong's axioms (Armstrong, 1974) provide a good formal basis but it cannot be considered a real approach, because it does not have an executable orientation*".

This situation has induced the use of indirect methods to solve the main problems related with functional dependencies. Thus, there is in the literature a lot of work which introduced *ad hoc* algorithms focussed on different functional dependencies problems. We would like to address these problems in an uniform framework, without sacrificing a good performance.

SLFD logic provides a formal basis to deal with functional dependencies with an extra benefit in the efficiency. As (Mora et al., 2012) shows, algorithms based on SLFD operators are even faster than those proposed in previous work in the literature. The reason is that the inference system of the SLFD logic is not based on the transitivity rule (as is the other classical functional dependencies logic). The intrinsic behavior of transitivity avoids its use in real problems, but it is the core of all these logics. SLFD provides a new simplification rule as the kernel of its novel inference system (Cordero et al., 2002).

The core of the classical axiomatic system is the transitivity rule:

$$\lfloor Trans \rfloor \quad \frac{X \to Y \quad Y \to Z}{X \to Z}$$

SLFD avoids this rule and it is substituted by a new simplification rule:

$$\lfloor Simp \rfloor \quad \frac{X \to Y,\ U \to V, X \subseteq U}{(U - Y) \to (V - Y)}$$

$\lfloor Simp \rfloor$ allows the use of logic to manage functional dependencies. It does not consider two functional dependencies and produces a new one (like $\lfloor Trans \rfloor$ does). It reduces an existing functional dependency ($U \to V$) by simplifying some of their redundant attributes. Thus, the axiomatic system works by reducing the original problem to a simpler one with no redundancy. This new orientation has a direct impact in the efficiency of the methods designed on the new logic.

[1] Spanish acronym for Cooperation in Databases.

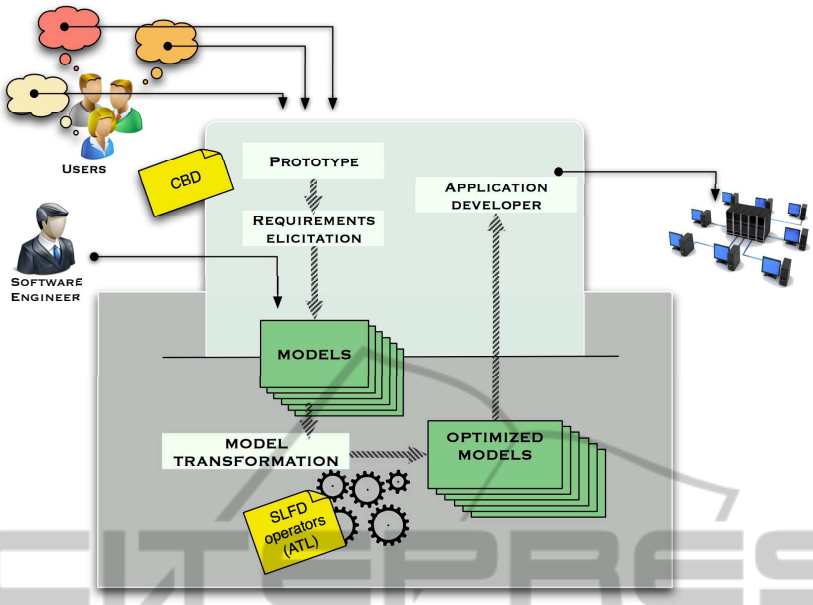

Figure 1: FD-based model driven architecture of CBD.

2.2 CBD

CBD was created to provide an improvement in one of the critical points in software development: user requirements elicitation. This process has several well-known problems associated with it, as described in most Software Engineering books (Pressman, 2010; Sommerville, 2011). These problems have traditionally been tackled by means of new modeling techniques or new software process models (e.g. iterative processes (Jacobson et al., 1999) or agile methodologies (Martin, 2003)). These process models increase user participation, but finally an analyst must "translate" the information gathered to requirements and models. In most real-word projects, analysts introduce "noise" and a deviation from real user needs. So, this translation usually introduces inaccuracies that spread throughout the development process and reduce software quality.

CBD's approach to this problem tries to avoid the main role of the analyst in requirements elicitation. In CBD the final user specifies its requirements directly by means of collaborative techniques. More concretely, the user intuitively designs in CBD the user interface (forms containing UI components and controls) that he wishes. Then CBD executes an engineering process (Cordero et al., 2010) that generates a model (that conforms to a CBD metamodel) of functional and information requirements. This model is processed by the CBD engine generating use case, structure (class) and behavioral (sequence) UML models (in XMI format). Furthermore, CBD generates a relational database (to be used in the user application). It is in this last step where the results of this work will be applied.

CBD is designed with a web application architecture. At http://www.cbd.uma.es the reader can see a video illustrating the workflow of CBD.

The metamodel of CBD is expressed in XSD and all the information managed by CBD is stored in XML format, using the XML native database management system eXist-db. So, the metadata related to the information and functional requirements, as well as the specifications of user interface are expressed in XML documents. The metamodel includes the concept of functional dependency.

In an evolution still in development, CBD will have a web-application generation functionality. From the requirements given by the user and the generated models, it will be possible to generate a final application for the user. In the current version generation of applications is done with *ad hoc* techniques, which will be redesigned to use MDE techniques, particularly model-to-text transfomations.

Though generation of applications by the user is a service offered by other solutions (Zoho or Google Forms, for example), as far as we know, none of them offer the power of CBD in the generation of databases with a great wealth of tables and relations, or the generation of UML models in XMI format (Cordero et al., 2010).

3 METAMODEL FOR FUNCTIONAL DEPENDENCIES

In this work we propose to combine our theoretical results about functional dependencies with CBD, under the MDE paradigm. In this sense we use the modeling framework EMF (Steinberg et al., 2009), to adjust to the standard. CBD generates models that conform to a XSD metamodel. So, our first step is to transform this metamodel to Ecore. We use the XSD2Ecore transformation of EMF.

There exists in the literature some work which introduces a metamodel for the relational data model (Akehurst et al., 2002; Laleau and Polack, 2001). They introduce a class to define relational tables and it also allows the encoding of constraints (primary key, unique and functional dependency). In (Akehurst et al., 2002) the authors provide an OCL specification of the functional dependencies normal forms and they design a tool which supports the normalization process. This solution may be considered a top-down approach, because we consider a rough relational model with (probably) some great degree of redundancy, and the normalization model transformation renders, by table decomposition, a depurated relational model with no redundancy and a greater number of smaller tables. This work provides a very interesting framework in the model-driven engineering area to deal with functional dependencies. Nevertheless, this approach is not suitable for use in our collaborative scenario.

CBD collects the information from different users and integrates it into a unified model. All the information stored in the CBD model needs to be flexible in order to allow further integration. So, it does not use tables until the design process is concluded and CBD generates a database and the code to manage it.

In this paper we present a metamodel oriented to the specification of functional dependencies, without any table reference. This metamodel considers the whole set of attributes involved in a system and relates them using functional dependencies. This new approach is suitable for bottom-up processes, like that used by CBD. It considers the atomic information about attributes and dependencies, builds a model that conforms to a functional dependency metamodel and transforms it (using SLFD model transformations) rendering a set of functional dependencies. This functional dependency model may be transformed into a depurated relational model.

The use of our functional dependency metamodel provides both a bottom-up approach suitable for a collaborative environment and a greater capability of redundancy removal. In the normalization theory, only the interaction among the functional dependencies of the same table are considered. Thus, the information used to remove redundancy in a table is limited to the intra-table functional dependencies. In our metamodel, the functional dependencies do not belong to any table and they are part of a global metamodel which represents all the information from different users. The Simplification rule may be used in the whole set of functional dependencies, providing a greater level of interaction among functional dependencies.

To illustrate the advantage of the inclusion of the functional dependencies among all the attributes of the system, we present the following example:

Example 1. *We consider that in our company we have the following table to store the details of each employee* `EMPLOYEE(Emp#,Teleph#, Room, Employee_Name)`. *Its primary key is* `Emp#`. *We also have another table to store the location of each IP:* `NETWORK(IP, Room)`. *Its primary key is* `IP`. *Note that these tables are fully normalized up to* 3^{rd} *Normal Form. Suppose the company decides to save money and it adopts the VoIP protocol. Thus, we have to store the following information* `VOIP(Teleph#,IP)`, *where* `Teleph#` *is the primary key.*

In the relational model, the three tables are fully normalized. But if you consider a set of all the functional dependencies fulfilled in the whole set of attributes (apart from the table they belongs to), we obtain the following set: {`Emp#` → `Teleph# Room Employee_Name`, `IP`→ `Room`, `Teleph#`→ `IP`}. *This set may be simplified to the following set of functional dependencies having less redundancy* {`Emp#` → `Teleph# Employee_Name`, `IP`→ `Room`, `Teleph#`→ `IP`}.

We will focus on the subset of the CBD metamodel relative to functional dependencies. In figure 2 we show that fragment. The metamodel includes the concepts of Schema, Attribute and Functional Dependency, with their usual interpretations. We would remark that in this metamodel, a functional dependency may have more than one attribute both on its left and right sides. So, two classes are required to model the functional dependency "ends".

4 MODEL TRANSFORMATION BASED ON SLFD

As we have mentioned in the preceding sections, we propose the use of SLFD logic as a basis for the development of some model transformations. There can be found in the literature a set of works which show that

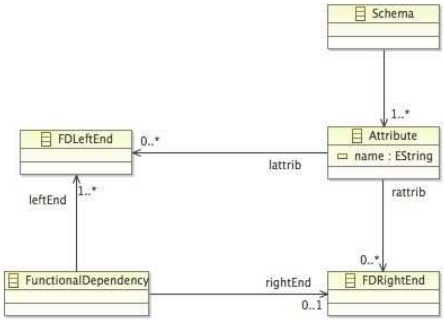

Figure 2: FD model-driven architecture.

the SLFD inference system may be used to produce efficient algorithms which solve several of the problems of functional dependencies. In all these methods the simplification rule plays an outstanding role.

To tackle each functional dependencies problem, in each approach the authors have designed a specific operator which transforms the set of functional dependencies and renders a proper solution. In all these logic operators the simplification rule is the atomic component in the reasoning and inference. We present here some of these algorithms, referring the reader to the original work for a detailed explanation [2]:

- In (Cordero et al., 2002; Mora et al., 2003) the simplification rule is directly used to design a couple of transformations which reduce the redundancy in the functional dependencies set looking for a more optimal specification. These transformations are similar to the normalization process.
- In (Mora et al., 2012) a closure method is presented. It introduces a SLFD operator which carries out the simplification of each element in the set of functional dependencies up to getting the closure for the input set of attributes. It introduces the use of a seed, built as a functional dependency with the empty set on the left side. This seed will play the role of the first functional dependency of the Simplification rule. In the paper the authors show that this approach is even more efficient than the rest in the literature.
- In (Cordero et al., 2012) the authors introduce another SLFD operator heavily based on the simplification rule. The operator is used to introduce a depurated method which finds all the minimal keys. The operator is used in the branching process to reduce the search space, providing a powerful pruning and a great benefit in the efficiency of the method.

[2]On the site http://www.slfd.uma.es/WebFDTools/ the reader will find an executable web prototype of some of these algorithms.

We propose to use these algorithms to define a set of model transformation the objective of which is to produce a depurated set of functional dependencies with no redundancy. This set of transformations will render a new functional dependencies model which will be transformed into an optimal relational model. Then, the use of model-to-text transformations allows the generation of DDL scripts.

The transformations will be defined in ATL (Jouault and Kurtev, 2006), a language properly integrated into the EMF framework.

In this paper we only show the key piece of all the transformations we propose, which is the simplification rule. Since this rule will be applied in the implementation of the operators mentioned above, it is defined as a ATL helper.

```
helper context
MMFunctionalDependency!FunctionalDependency def :
simplifRule(pivot:FunctionalDependency) :
FunctionalDependency =

leftEnd <- leftEnd - pivot.rightEnd,
rightEnd <- rightEnd - pivot.rightEnd;
```

This helper will be included in a transformation controlled by an iterator that will only apply it to a functional dependency `fd` when the expression

```
fd.leftEnd.contains(pivot.leftEnd)
```

evaluates to true.

5 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a metamodel that allows us to optimize data models through the application of MDE techniques. The metamodel is based on the concept of functional dependency, and considers the full set of attributes in the schema, with no relational table structure.

This fact, together with the definition of a set of model transformations with a solid formal basis (the SLFD logic) provides an appropriate mechanism for the elimination of redundancy. In addition, our approach allows a process of bottom-up integration suitable for collaborative developments.

The functional dependency metamodel has already been incorporated into the collaborative tool CBD. This tool allows end users to design the forms and UI components they need in their application. This information is collected in models that conform to the metamodel we have presented. From all these models CBD generates a unified database for the end application, as well as UML models in XMI format.

Our work in progress is the implementation of model transformations based on the operators of the SLFD logic. These operators are based on the simplification rule, that provides better results than the usual classical algorithms based on the transitivity rule. We use EMF framework and ATL to define the metamodel and the model transformations.

Regarding future work, one of our goals is to complete the redesign of CBD by applying MDE techniques. In this sense, we will use model transformations to generate UML models, and model-to-text transformations for code generation.

ACKNOWLEDGEMENTS

Supported by grant TIN2011-28084 of the Science and Innovation Ministry of Spain, co-funded by the European Regional Development Fund (ERDF).

REFERENCES

Akehurst, D., Bordbar, B., Rodgers, P., and Dalgliesh, N. (2002). Automatic Normalisation via Metamodelling. In *ASE 2002 Workshop on Declarative Meta Programming to Support Software Development*.

Armstrong, W. W. (1974). Dependency structures of data base relationships. In *IFIP Congress*, pages 580–583.

Cordero, P., Enciso, M., Guevara, A., Caro, J. L., Mora, A., and Rossi, C. (2010). A tool for user-guided database application development. Automatic design of XML models using CBD. In *5th International Conference on Software and Data Technologies. ICSOFT 2010*, pages 195–200.

Cordero, P., Enciso, M., and Mora, A. (Apr, 2012). Automated reasoning to infer all minimal keys. *Submitted to Information Processing Letters*.

Cordero, P., Enciso, M., Mora, A., and de Guzmán, I. P. (2002). SL_{fd} logic: Elimination of data redundancy in knowledge representation. In *8th Ibero-American Conference on Artificial Intelligence, IBERAMIA 2002*, pages 141–150.

Enciso, M., Mora, A., Cordero, P., and Baena, R. (2011). A claim to incorporate functional dependencies in development tools. Benchmarking and checking functional dependencies algorithms. In *6th International Conference on Software and Data Technologies. ICSOFT 2011*, pages 313–316.

Garcia-Molina, H., Ullman, J., and Widom, J. (2008). *Database Systems: The Complete Book*. Pearson.

Gogolla, M. (2005). Exploring ER and RE syntax and semantics with metamodel object diagrams. In *Metainformatics 2005*, pages 61–72.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison Wesley.

Jouault, F. and Kurtev, I. (2006). Transforming models with ATL. In *Satellite Events at the MoDELS 2005 Conference*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138, Berlin. Springer Verlag.

Laleau, R. and Polack, F. (2001). A rigorous metamodel for UML static conceptual modelling of information systems. In *13th International Conference on Advanced Information Systems Engineering, CAiSE'01*, pages 402–416.

Martin, R. (2003). *Agile software development : principles, patterns, and practices*. Prentice Hall.

Mora, A., Cordero, P., Enciso, M., Fortes, I., and Aguilera, G. (2012). Closure via functional dependence simplification. *International Journal of Computer Mathematics*, 89(4):510–526.

Mora, A., Enciso, M., Cordero, P., and de Guzmán, I. P. (2003). An efficient preprocessing transformation for functional dependencies sets based on the substitution paradigm. In *10th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2003*, pages 136–146.

OMG (2001). *The Unified Modeling Language Version 1.4*. Object Management Group formal//01-09-67.

Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach, 7/e*. McGraw-Hill.

Sommerville, I. (2011). *Software Engineering, 9/e*. Pearson.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition.