A/D CASE: A NEW HEART FOR $FD3^*$.

A. Mora, M. Enciso, P. Cordero, I. Pérez de Guzmán, J. Guerrero
E.T.S.I. Informática. Universidad de Málaga.
Campus de Teatinos. 29071 Málaga, Spain

Keywords: Conceptual data model; Functional dependencies, Logic and Information systems; Schema integration; Reverse engineering.

Abstract: In (Enciso and Mora, 2002) we introduce the *Functional Dependencies Data Dictionary ($FD3$)* as an architecture to facilitate the integration of database Systems. We propose the use of logics based on the notion of Functional Dependencies (FD) to allows formal specification of the objects of a data model and to conceive future automated treatment.
The existence of a FD logic provides a formal language suitable to carry out integration tasks and eases the design of an automatic integration process based in the axiomatic system of the FD logic. Besides that, $FD3$, provides a *High Level Functional Dependencies* (HLFD) Data Model which is used in a similar way as the Entity/Relationship Model.
In this paper, we develop a CASE tool named A/D CASE (Attribute/Dependence CASE) that illustrates the practical benefits of the $FD3$ architecture. In the development of A/D CASE we have taken into account other theoretical results which improve our original $FD3$ proposal (Enciso and Mora, 2002). Particularly:

- A new functional dependencies logic named SL_{FD} for removing redundancy in a database sub-model that we present in (Mora, 2002; Cordero et al., 2002a). The use of SL_{FD} add formalization to software engineering process.
- An efficient preprocessing transformation based on the substitution paradigm that we present in (Mora et al., 2003).

Unlike A/D CASE is independent from the Relational Model, it can be integrated into different database systems and it is compatible with relational DBMSs.

1 INTRODUCTION

An heterogeneous database system arises from several sub-systems described by local designers which may use different data models (relational, hierarchical, network , files system, etc.). All these data models have in common the existence of attributes (atomic data) and relationships between them. The data and most of their relationships can be stored in databases using functional dependencies.

As we show in (Enciso and Mora, 2002), database integration usually cover the following two steps (see also (Atzeni and Torlone, 1997)):

- The mapping between sub-models and a selected canonical model.
- The removing of data redundancies in the integrated model.

The important notion of *functional dependence* (FD) allows the integration of several data submodel in a new global data model having a formal basis.

We conceive a new data model based directly in FD logic. The FD data model will be considered as the integration canonical model. We describe the data and the relationship among them using the notion of functional dependence and we develop an axiomatic system to have deduction capabilities.

In our methodology the user participates directly in the design process. In (Enciso and Mora, 2002), we propose the use of the *Functional Dependencies Data Dictionary*, named $FD3$, as an architecture for

[*]This work has been partially supported by the Cicyt research project 1109/2000

Mora A., Enciso M., Cordero P., Pérez de Guzmán I. and Guerrero J. (2004).
A/D CASE: A NEW HEART FOR FD3.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 482-487
DOI: 10.5220/0002629004820487

assisting the integration of heterogeneous databases.

$FD3$ is based in a logic which allows the description of the FDs contained in the local database systems and the construction of a unified global system. To communicate the information collected in the global model, we introduce a *High Level Functional Dependencies* (HLFD) Data Model which is used in a similar way as the Entity/Relationship Model. The HLFD data model can be deduced automatically from a EFD logic theory. Furthermore, it is possible to translate automatically the HLFD data model into a relational database.

In (Cordero et al., 2002a), we introduce a new axiomatic system for FD logics, named SL_{FD} particularly designed to remove redundancy. We define two substitution operators and we illustrate their behaviour for removing redundancy. The new system improves the FD logic presented in (Enciso and Mora, 2002) and it is equivalent to Armstrong's axioms (Armstrong, 1974). We also showed that SL_{FD} is more adequate for the applications.

In (Mora et al., 2003) we introduce a preprocessing transformation based on SL_{FD} which removes redundancy in a given set of Functional Dependencies and allows a more efficient further management by other well known algorithms (Atzeni and Torlone, 1997; Biskup and Convent, 1991; **?**). We have carried out an empirical study to prove the practical benefits of our approach.

In this paper, we present Attribute/Dependencies (A/D) CASE, a case tool which apply the result cited above in the area of heterogeneous database integration and in database cooperative design (Cordero et al., 2002b). A/D CASE includes a *High Level Functional Dependencies* (HLFD) Data Model which can be deduced from the global data dictionary, using automated reverse engineering.

This paper is organized as follows: in section 2 we summarize the $FD3$ architecture presented in (Enciso and Mora, 2002). Section 3 introduces SL_{FD} as the new heart of the $FD3$, and section 4 presents the pre-processing transformation which removes redundancy in a given set of Functional Dependencies. The AD/CASE tool is showed in section 5. Section 6 outlines the conclusions and future works.

2 FD3 ARCHITECTURE

As was argued in (Bertino et al., 2001), to facilitate user participation, we need new tools (easier to use and more powerful) and new techniques (including new data models). In this paper, we present Attribute/Dependencies (A/D) CASE, a case tool to design databases in a heterogeneous environment.

A/D CASE allows to put in practice the $FD3$ architecture presented in (Enciso and Mora, 2002). $FD3$ turns around a simple element: *The Functional Dependence (FD)*. The FD notion is inherent to most used database models (hierarchical, network, relational,etc.). Figure 1 shows briefly the $FD3$ architecture presented. The main characteristic of our architecture is the use of logic in all the stages: analysis, design, model transformation, integration, etc.

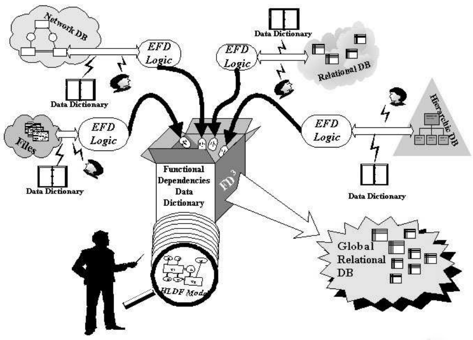

Figure 1: The Functional Dependencies Data Dictionary

$FD3$ allows the generation of a global data model as follows:

1. (i) We extracts some FD's automatically from the conceptual data model (*structural FDs*), (ii) the designer adds other FDs which corresponds to requirements of the information system (*environment FDs*). The local data dictionaries will be formally represented using an FD logic.

2. $FD3$ is the union (integration) of all the FD subtheories (local data dictionaries), rendering an integrated FD logic theory.

3. $FD3$ is depurated and we remove redundancy by applying the preprocessing transformation that we propose in (Mora et al., 2003). We obtain the Depurated FD logic theory which corresponds to the global schema of the heterogeneous database.

4. Finally, we may deduce a high level data model, named *High Level Functional Dependencies (HLFD) data model* from the integrated FD logic theory . The HLFD data model allows us to obtain a global vision of the whole system with a strong level of abstraction. The designer has a high level data model that will be use in a similar way as the Entity/Relationship model.

3 SUBSTITUTION LOGIC

In this paper we select Substitution Logic SL_{FD} (Cordero et al., 2002a) to be the heart of

$FD3$. SL_{FD}[2] is a formal system appropriate to be used in integration process.

In this section we summarize the new axiomatic system SL_{FD} (Cordero et al., 2002a). Their axiomatic system is guided by the idea of remove redundancy in an efficient way. This is one of the novelties of SL_{FD} because other well known FD logic system are guided by Armstrong Relations (Armstrong, 1974), which captures all the FD which can be deduced from a given set of FDs.

Other important novelty of SL_{FD} is the definition of two substitution operators which have not been defined up to now in other FD logic. Their application do not imply the incorporation of *wff*, but the substitution of new *wffs* by simpler ones, with an efficiency improvement

Definition 3.1 *Given the alphabet* $\Omega \cup \{\mapsto\}$ *where* Ω *is an infinite numerable set, we define the language* $\mathbf{L}_{FD} = \{X{\mapsto}Y \mid X, Y \in 2^{\Omega}$ *and* $X \neq \varnothing\}$. *In the literature, attributes must be non-empty. Notice that in* $\mathbf{L}_{FD}$ *the right hand side of a wff may be the empty set, named* $\top$.

We define an axiomatic system, $\mathcal{S}_{FDS}$, for $\mathbf{L}_{FD}$ with a substitution rule as primitive rule. The main novelty of the axiomatic system is that, for first time (Atzeni and Antonellis, 1993; Fagin, 1977a; Ibaraki et al., 1999; Paredaens et al., 1989), transitive rule is not a primitive rule, with the consequently efficiency benefits.

Definition 3.2 *The system* $\mathcal{S}_{FDS}$ *defined on* $\mathbf{L}_{FD}$ *has one axiom scheme:*

Ax_{FDS} : $\vdash X{\mapsto}Y$, *where* $Y \subseteq X$. *Particulary,* $X{\mapsto}\top$ *is an axiom scheme.*

The inference rules are the following:

Fragmentation rule

$\lfloor Frag \rfloor$: $X{\mapsto}Y \vdash_{\mathcal{S}_{FDS}} X{\mapsto}Y'$, *where* $Y' \subseteq Y$

Composition rule

$\lfloor Comp \rfloor$: $X{\mapsto}Y,\ U{\mapsto}V \vdash_{\mathcal{S}_{FDS}} XU{\mapsto}YV$

Substitution rule

$\lfloor Subst \rfloor$: $X{\mapsto}Y,\ U{\mapsto}V \vdash_{\mathcal{S}_{FDS}} (U\text{-}Y){\mapsto}(V\text{-}Y)$, *where* $X \subseteq U$, $X \cap Y = \varnothing$

This axiomatic system is equivalent to other well known FD axiomatic system (Atzeni and Antonellis, 1993; Fagin, 1977a; Ibaraki et al., 1999; Paredaens et al., 1989) and, thus, we have the usual derived rules in SL_{FD}. Particularly we may derive the *Reduction Rule* and the *Union Rule* that will be used later:

Reduction Rule

$\lfloor Reduc \rfloor$: $X{\mapsto}Y \vdash X{\mapsto}Y\text{-}X$, where $Y\text{-}X \neq \varnothing$

This rule allows the construction, in linear time, of an equivalent FD set with less redundancy.

Union Rule

$\lfloor Comp \rfloor$: $X{\mapsto}Y,\ X{\mapsto}V \vdash_{\mathcal{S}_{FDS}} X{\mapsto}YV$

This rule allows a reduction in the number of *wff* contained in the FD set. Nevertheless, all automated deduction FD systems uses fragmentation rule instead of union rule. Fragmentation ensures the minimun size in left hand side of the *wff*, but enlarge the size of the FD set.

Furthermore, we have the following derived rule, a novelty in the literature: **r-Substitut.Rule**

$\lfloor rSust \rfloor$: $X{\mapsto}Y, U{\mapsto}V \vdash U{\mapsto}(V\text{-}Y)$, if $X \subseteq UV, X \cap Y = \varnothing$

4 A PRE-PROCESSING TRANSFORMATION BASED ON THE SUBSTITUTION PARADIGM

In (Mora et al., 2003) we present an efficient preprocessing transformation, based on the substitution paradigm, which removes redundancy from a given set of functional dependencies. Furthermore, we use Prolog to build an empirical study which illustrates the practical benefits of our approach.

The preprocessing transformation establishes an efficient pruning based mainly on the substitution rules. In some cases, our preprocessing transformation captures the redundancy of the original FD set entirely, with the corresponding benefits for the efficiency. The transformation applies the following steps [3]:

- In step 1, the rule $\lfloor Reduc \rfloor$ transforms FDs into reduced FDs.
- In step 2, the rule $\lfloor Union \rfloor$ renders FDs with disjoint determinants.
- In step 3, we exhaustively apply the substitution rules. After each application of substitution if the result requires it, the union rule will be applied before the following substitution.

As we remark in the previous section, before starting step 3 the size of the FDs set has been reduced with limited linear cost. We will achieve an important improvement with respect the rest of FDs algorithms, because all of them apply the rule $\lfloor Frag \rfloor$ as their first transformation, which increases the number of FDs.

This preprocessing transformation is applied to an input FD set, rendering a new FD set with less redundancy. In some cases, the new set has been treated

[2]We replace EFD logic presented in (Enciso and Mora, 2002) by Substitution Logic presented in (Cordero et al., 2002a) because of their practical benefits.

[3]The transformation has quadratic complexity.

completely and it does not have any redundant FD. In other cases, the new set has less size (considering both, attributes and FDs) than the original one and, consequently, can be treated more efficiently by other well known algorithms (Atzeni and Torlone, 1997; Biskup and Convent, 1991; Coulondre, 2003).

5 A/D CASE

In (Enciso and Mora, 2002) we propose as a future work to develop a case tool to implement all the techniques involved in the $FD3$ architecture. In this section we show A/D CASE v1.0, which covers this ambitious goal.

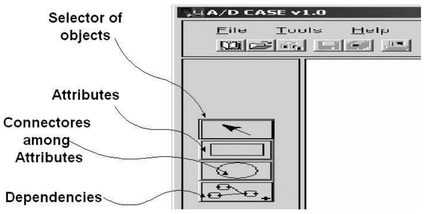

Figure 2: A/D CASE

In the Figure 2 we show the buttons that appear in A/D CASE. Figure 3 shows the environment of A/D CASE. In the following subsections, we will show

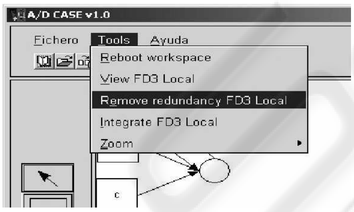

Figure 3: A/D CASE tools.

how A/D CASE helps the users to built a global and unified schema following the $FD3$ architecture.

5.1 Build the local FD3s

To prevent the users from managing a formal system, they will generate a *Functional Dependencies Diagram* (FD Diagram). This is a natural way to communicate their knowledge about the Information System. The FDs included in the FD diagram are translated into SL_{FD} well-formed formulas automatically. Thus, we have the benefits of logic formalisms (soundness and automated treatment) without *suffering* their disadvantages (user-unfriendlyness).

Notice that the direct specification of functional dependencies by the user is a novelty in the literature. In fact, only a few data models manage FDs, and most of them consider the FD in a hidden mode, because there seems to be a concept difficult to learn. Our opinion is that FD Diagram eases the comprehension of FD, which is more natural than it appears. Thus, A/D CASE simply asks the user for his *data* (his attributes) and let him to establish a connection, named *the left han side determines the right hand side*.

The user inserts the *wffs* that represent his local data model using A/D CASE. The user draws the attributes in a rectangle box and uses the *connect* button to specify the existence of a FD among the attributes. Each connection symbol (each circle) represents a FD of the local schema. Notice that the box representing an attribute appears only once in the local FD diagram.

A/D CASE allows to manipulate the graphical representation of attributes and dependencies in the usual way: add, modify, delete, move, resize, zoom, etc.

Figure 4 shows an example of a local schema representing an airport subsystem. The user specify his subsystem *drawing* the functional dependencies.

In (Enciso and Mora, 2002),(Cordero et al., 2002b) we describe exhaustively the equivalence between a set of *wffs* of the FD logic and a FD diagram. A/D CASE helps the user to specify FDs and to translate diagrams into *wffs*.

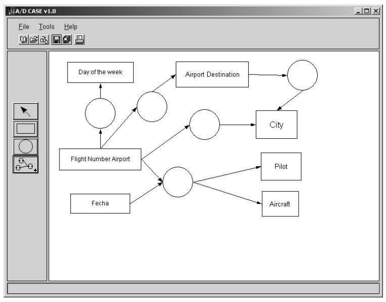

Figure 4: An FD Subscheme.

5.2 Remove redundancy from all local schemas

SL_{FD} is the core of this process. It is possible to remove redundancy from a FD diagram using the new *Remove Redundancy* pre-processing transformation.

We would like to remark that the success of our approach is due to the existence of an axiomatic system which follows a different direction in the FDs treatment. Figure 3 shows the *remove redundancy FD3 Local* in the *Tools* submenu.

5.3 Obtain the global FD3

The above step improves the efficiency of the joining process: the local views that should be integrated have been depurated separately. Now, we integrate all the sub-schema in a global schema containing all the information in a unified mode.

In our tool, it is a trivial task, because integration is defined with the union set operator. The integrated model will be depurated again to avoid redundancy.

5.4 Obtain HLFD data model

The FD data model is apropiat to integrate and manipulate data knowledge. Nevertheless, it is not a good approach to communicate this information to the users. This task must be done using another model with a higher level of abstraction. By the other side, we would like to get an automated process which enable us to get this high level data model directly from the FD data model, using automated reverse engineering techniques.

A/D CASE construct in a automatic way a new data model, named *High Level Functional Dependencies* (HLFD) Data Model (see (Enciso and Mora, 2002; Cordero et al., 2002b)) which can be used to communicate information in a more natural way. Figure 5 presents the HLFD Data Model of the Figure 4.

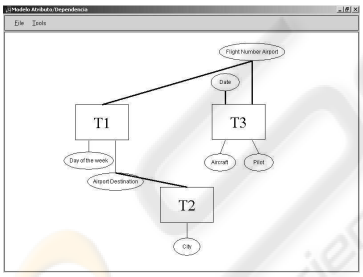

Figure 5: The HLFD Data Submodel.

A/D CASE tool generates automatically a HLFD data model. As we introduce in (Enciso and Mora, 2002), the HLFD presents the attributes grouped in objects which can be considered as entities. A/D CASE label each object with a generic name (T_1, T_2, etc. in figure 5. Later, the user label these objects with names which provides semantic information to the global HLFD model.

Finally, A/D CASE may translate the HLFD data model to a relational model (Enciso and Mora, 2002; Cordero et al., 2002b). Thus, A/D CASE generates both, the relational database itself and the Entity/Relational model which correspond with the information contained in the global FD data model.

6 CONCLUSIONS AND FUTURE WORK

In this work we present A/D CASE, a case tool for integrating database local schemes in a heterogeneous framework. A/D CASE follows the *Functional Dependencies Data Dictionary ($FD3$)* Architecture presented in (Enciso and Mora, 2002) and it has automated deduction capabilities. The engine of A/D CASE is based on the new logic SL_{FD} and on the preprocessing transformation presented in (Mora et al., 2003). The heart of A/D CASE allows to remove redundancy in a set of functional dependencies and facilitates the integration process. Beyond this technical results, this work shows that functional dependencies logics may be used successfully in practice.

Furthermore, A/D CASE generates automatically, using reverse engineering techniques, a *High Level Functional Dependencies* (HLFD) Data Model which may be used in a similar way as the Entity/Relationship Model.

In short term, we will use A/D CASE to make an empirical study about the use of the FD data model versus the use of the Entity/Relationship model. We will propose several information systems to different designers, some of them will use A/D CASE and the FD data model and the others will use an Entity/Relationship case tool. We will compare the model obtained by these users using different data models and different tools.

In medium term, we will extend A/D CASE in two directions:

- To consider the manipulation of another data dependencies (Fagin, 1977b; Lakshmanan and Veni Madhavan, 1987; Lopes et al., 2002).
- To investigates how the proof procedure for the implication problems, called *chase* (Biskup and Convent, 1991) and a new top-down proof procedure for generalized data dependencies (Coulondre, 2003) can be improved with the substitution paradigm.

In long term, we intend to apply our extended theoretical result to a set of current problems that have been face on with dependencies, like the following:

- The elimination of replication in XML (Lee et al., 2002).
- The elimination of redundancy in the relations between data discovered using Data Mining techniques (Lopes et al., 2000).
- The elimination of redundancy in associations rules discovered using Data Mining techniques (Calders and Paredaens, 2003).

REFERENCES

Armstrong, W. W. (1974). Dependency structures of data base references. *IFIP Cont. Poc.*

Atzeni, P. and Antonellis, V. D. (1993). Relational Database Theory. *Benjamin/Cummings*.

Atzeni, P. and Torlone, R. (1997). Mdm: a multiple-data-model tool for the management of heterogeneous database schemes. *ACM-SIGMOD*.

Bertino, E., Catania, B., and Zarri, G. P. (2001). Intelligent database systems. *ACM Press. Addison-Wesley*.

Biskup, J. and Convent, B. (1991). Relational chase procedures interpreted as resolution with paramodulation. *Fundamenta Informaticae*, 15 (8):123–138.

Calders, T. and Paredaens, J. (2003). Axiomatization of frequent itemsets. *TCS*, 290 (1):669–693.

Cordero, P., Enciso, M., Guzmán, I. P. d., and Mora, A. (2002). SLfd logic: Elimination of data redundancy in knowledge representation. *LNAI 2527*, pages 141–150.

Cordero, P., Enciso, M., Mora, A., and Guzmán, I. P. d. (2002). Modelo de datos de dependencias funcionales para un entorno turístico cooperativo. *Proceedings TURITEC 2002*, pages 61–76.

Coulondre, S. (2003). A top-down proof procedure for generalized data dependencies. *Acta Informática*, 39:1–29.

Enciso, M. and Mora, A. (2002). FD3: A functional dependencies data dictionary. *Proceedings of ICEIS*, 2:807–811.

Fagin, R. (1977). Functional dependencies in a relational database and propositional logic. *IBM. Journal of research and development*, 21 (6):534–544.

Fagin, R. (1977). Multivalued dependencies and a new normal form for relational databases. *ACM TODS 2*.

Ibaraki, T., Kogan, A., and Makino, K. (1999). Functional dependencies in horn theories. *Artificial Intelligence*, 108 1-2:1–30.

Lakshmanan, V. S. and Veni Madhavan, C. E. (1987). An algebraic theory of functional and multivalued dependencies in relational databases. *TCS*, 54, 1:103–128.

Lee, M. L., Ling, T. W., and Low, W. L. (2002). Designing functional dependencies for XML. *LNCS. EDTB 2002 Proceedings.*, 2287:124–141.

Lopes, S., Petit, J.-M., and Lakhal, L. (2000). Efficient discovery of functional dependencies and armstrong relations. *EDBT 2000, LNCS*, 1777:350–364.

Lopes, S., Petit, J.-M., and Toumani, F. (2002). Discovering interesting inclusion dependencies: application to logical database tuning. *Information Systems*, 27 1:1–19.

Mora, A. (2002). Dependencias funcionales, ideal-operadores no deterministas y operadores de sustitución. *PhD Thesis. UMA*.

Mora, A., Enciso, M., Cordero, P., and Guzmán, I. P. d. (2003). An efficient preprocessing transformation based on the substitution paradigm. *CAEPIA 2003. LNAI. Springer Verlag*.

Paredaens, J., De Bra, P., Gyssens, M., and Van Gucht, D. V. (1989). The structure of the relational database model. *EATCS Monographs on TCS*.