# A TOOL FOR USER-GUIDED DATABASE APPLICATION DEVELOPMENT
*Automatic Design of XML Models using CBD*

Carlos Rossi, Antonio Guevara, Manuel Enciso, José Luis Caro
*Dpto. Lenguajes y CC de la Computación, Universidad de Málaga, Málaga, Spain*

Angel Mora, Pablo Cordero
*Dpto. Matemática Aplicada, Universidad de Málaga, Málaga, Spain*

Keywords: Software development tools, Requirements elicitation and specification, Analysis and design, Functional dependencies, Logic.

Abstract: Beyond the database normalization process, much work has been done on the use of functional dependencies (FDs), their discovery using mining techniques, their use in query optimization and in the design of algorithms dealing with the implication problem etc. Nevertheless, although much research expounds the benefits of using functional dependencies, only a few modeling tools actually use them. In this work we present CBD, a new software development tool which allows end users to specify their requirements. CBD allows the user to design his/her own GUI for the application using forms and interface elements and it builds a meta-data dictionary with information on functional dependencies. This data dictionary will be used to generate the unified data model and a behavior model.

## 1 INTRODUCTION

Database design is not only a matter of dealing with the data that has to be stored. Since E.F. Codd introduced the Relational Model (Codd, 1970), experts have agreed on the importance of storing both the data and the semantics related to it.

Most relational database management systems (DBMS) and modeling tools use dependencies in a simple way and they rely on the specification of primary keys. The reason is that the number of keys are just equal to the number of tables in a database model, but the number of functional dependencies (FDs) are greater and they are more complex, because they may establish a relation among attributes that are in different tables (using the inclusion dependencies). Thus, the main obstacle is not the inclusion of FDs but their efficient treatment.

In (Evaluation, 1000) a new architecture which promotes the use of FDs was presented. The work focussed on the problem of view integration and proposed FDs as a key tool to discover knowledge and to facilitate the integration task. In addition to this architecture, a set of efficient methods to manage functional dependencies should be considered. In the literature, functional dependencies are dealt with using ad hoc methods designed for specific purposes which are difficult to extend. An alternative approach is to use a proper subset of classical logic to reason about FDs. There are many equivalent logics in the literature which follow this approach and all of them are very similar to that proposed by (Armstrong, 1974). As we will mention later however, these logics are not appropriate for automated reasoning. In (Cordero et al., 2002) a new kind of FD logic which allows the design of automated methods is presented (Cordero et al., 2002).

In this work we present CBD [1], a new modeling tool which allows the user to participate directly in the design process. Unlike the classical tools, CBD provides an interface to design directly each user view of the application. The information is captured using GUI elements (forms, buttons, etc) and they are stored in a meta-data dictionary. We have used an XML database instead of a relational database because we need flexible data storage, easy to share with other

[1]Spanish acronym for Cooperation in Databases.

applications. The FDs deduced from the GUI element connections provide some valuable information. CBD works by using the XML dictionary to generate the unified data model and the application code to cover user requirements. In (Nelson et al., 2005) the quality of data models is studied and redundancy elimination is cited as one of the important issues for increasing software quality. We will use an automated method based on FD Logic to reduce redundancy in the CBD dictionary.

The paper is organized as follows: in section 2 CBD is introduced and a comparison with other commercial tools is shown. Section 3 is devoted to presenting some efficient methods to reason with FDs and an explanation of how to use these methods to reduce redundancy in the CBD XML data dictionary. After the conclusions are drawn and the plans for future work are outlined in section 5, we include an appendix which illustrates how CBD works.

## 2 SOFTWARE DEVELOPMENT TOOLS

Nowadays, software development can be approached in two ways depending on who leads it:

1. Software development by professional teams, who gather user requirements by means of interviews, document inspection, etc. In this case, developers apply a direct engineering process, using a variety of CASE tools, such as (UML) modeling tools, integrated development environments, project management tools, etc.
2. Software development by the end user by means of application generators. These tools allow users to edit the application interface and to generate the end application (usually with a web architecture). This low-cost approach, although adequate for individuals and small businesses, is not optimal, since these tools are very limited in the management of non-trivial data relations and they have other drawbacks that reduce software quality.

CBD combines both types of development and also improves software quality using automated reasoning techniques. As we shall see in the following section, we apply an efficient transformation to the CBD data model and we reduce redundancy in the specification of the FDs. The simplification method produces a refined data model and consequently the corresponding relational database will be easier to manage.

### 2.1 CBD Overview.

CBD was developed in order to solve one of the main problems in professional software development: vagueness and incorrectness in gathering user requirements, which leads to serious flaws in the final product. This problem, widely detailed in Software Engineering literature (Pressman, 2006), has been traditionally approached by introducing new modeling techniques or applying process models that increase client interaction, such as iterative models (Jacobson et al., 2000) or agile processes (Martin, 2003). Nevertheless, in these approaches a human (the analyst) is still needed to translate the information provided by the client into requirements and models. This traslation usually causes mistakes generated by errors in analyst interpretation or by ambiguities and omissions in the client information.

CBD aims to solve this problem by avoiding the leading role of the analyst in requirement gathering: in CBD the end user records directly the input for requirements specification by means of collaborative techniques. More specifically, the user designs intuitively the GUI s/he wishes to have in their application, and then CBD that generates a catalogue of functional and information requirements (Guevara et al., 2007; Carrillo et al., 2008). The CBD engine processes this catalogue and it creates a relational database for the user application, as well as structural (classes), use case and behaviour (interaction) models in UML notation. These models are stored in XMI format. In this way, models can be imported in widely used CASE tools (MagicDraw, Enterprise Architect, Rational, etc), where an analyst could optimize the results automatically generated by CBD.

CBD manages semistructured data using eXist-db, a native XML database management system. In particular, requirements metadata, as well as user interface specifications are stored in XML documents. This architecture allows the application of reasoning methods for the treatment of dependencies in semistructured data and to optimize the relational database generated by CBD. At the moment, this task is carried out by a different application not integrated inside the CBD tool.

### 2.2 CBD Versus other Similar Tools

To explore the benefits of CDB, we believe it is necessary to compare it with other modeling tools. Because of the nature of CDB, we have chosen for this comparison those tools geared to the management, generation and creation of forms. Therefore, for this study we have considered the follow-

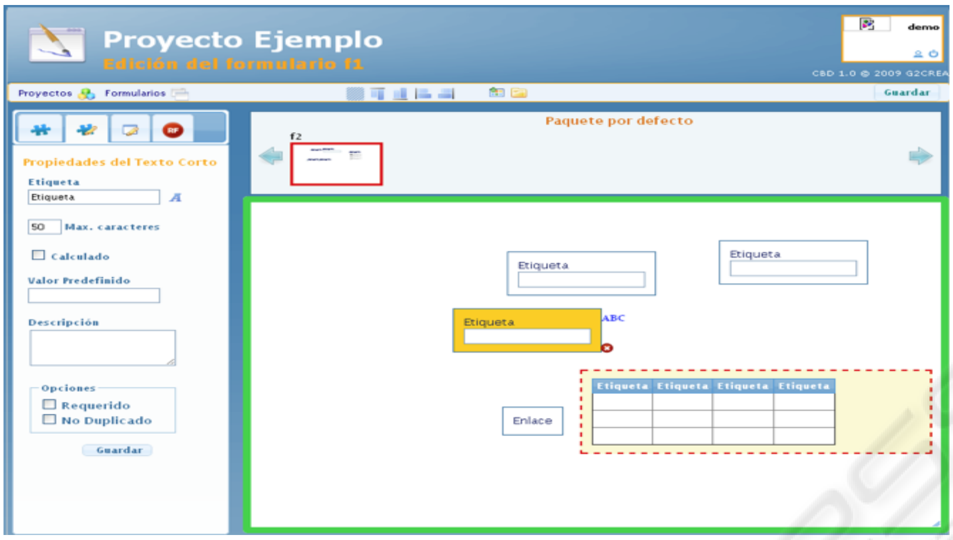


Figure 1: Definition of relations.

ing applications Wufoo (Infinity Box Inc., 2010), FormSpring (FormSpring LLC, 2010), FormAssembly (Veer West LLC, 2010), JotForm (Interlogy LLC, 2010), FormLogix (FormLogix.com, 2010) and Leonardi (Groupe W4 S.A, 2010), all of which have an analogical approach to CBD. The main feature common to all these tools is the construction of an automatic database model, using forms designed by users who may have no previous knowledge of databases.

We have organized the comparison according to the double use which may be made of these tools: either by users as a tool to create their own application or as a tool which captures requirements. First we will present the characteristics necessary for each of these uses, indicating which of the tools of the comparison have these particular features. Then, to close the section, we will present a comprehensive table of all the tools and their respective properties.

Firstly we compare the design and execution of the forms designed by the users. In this comparison, the following characteristics should be observed:

1. Most of the applications studied, exclusively implement systems with only one form, or, if several are possible, they are unrelated. Only Wufoo and Leonardi have multi-form mode possible and in the latter case it is extremely complicated to use. CBD addresses this issue and implements a solution based on the creation of a multi-form project, with the possibility of establishing navigation flows.

2. Secondly, it is important to study the expressiveness of the data models that can be designed. As can be observed in the comparative table in Figure 2, most of the tools only allow relations 1:n. Furthermore, the approach of the tools studied is to consider these relations as simple sets which can be selected and displayed, they are not tables. In this respect, both CBD and Leonardi both allow relations 1:n and n:m including parameterized grids (see Figure 1) . It should be noted that CBD can define reflexive relations 1:m, a characteristic which distinguishes it from the other tools.

3. One characteristic which is common to all the tools is the management of end users in the application generated. However, what makes CBD different is that it contemplates the participation of various users in the design, allowing user collaboration to take place. Furthermore, in keeping with this approach, CBD also includes a system that allows users to control and validate different versions.

As previously mentioned, as well as generating the application, it is possible to use these tools to extract requirements and to elaborate models from these requirements. In this second part of the comparison, we wish to highlight the following characteristics:

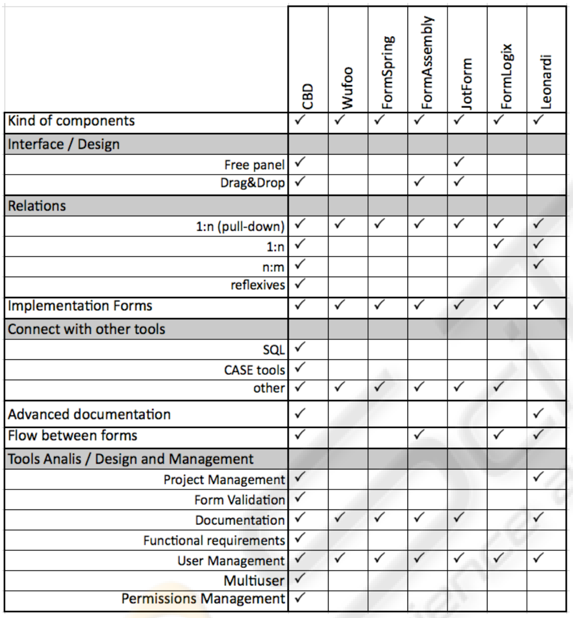
1. CBD allows data model generation in SQL. Other tools such as FormSpring or FormAssembly also allow the data model to be exported, but in this case it is only possible to do so using CSV files, which must be processed manually in order to be able to create the relational schema on a database management system.

2. Another important feature is that CBD offers the analyst a control panel for the forms and the relations that the users have designed, thereby allow-

ing analysts to obtain better knowledge of the system requirements. This characteristic is exclusive to CBD.

3. As we mentioned previously, our intention is for CBD to be useful also to professional developers. For this reason, CBD allows knowledge acquired in requirements gathering to be exported to other modeling environments (MagicDraw or Enterprise Architect for example). This is done in the form of UML class models, use cases and sequence diagrams, using standard XMI format, in such a way that this information can be integrated with other information obtained with these professional modeling tools.

Figure 2 illustrates all the aspects considered in this study. The table shows in great detail each functionality for all the tools included in this work:



| | CBD | Wufoo | FormSpring | FormAssembly | JotForm | FormLogix | Leonardi |
|---|---|---|---|---|---|---|---|
| Kind of components | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Interface / Design | | | | | | | |
| Free panel | ✓ | | | | ✓ | | |
| Drag&Drop | ✓ | | | ✓ | ✓ | | |
| Relations | | | | | | | |
| 1:n (pull-down) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 1:n | ✓ | | | | | ✓ | ✓ |
| n:m | ✓ | | | | | | ✓ |
| reflexives | ✓ | | | | | | |
| Implementation Forms | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Connect with other tools | | | | | | | |
| SQL | ✓ | | | | | | |
| CASE tools | ✓ | | | | | | |
| other | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Advanced documentation | ✓ | | | | | | ✓ |
| Flow between forms | ✓ | | | ✓ | | ✓ | ✓ |
| Tools Analis / Design and Management | | | | | | | |
| Project Management | ✓ | | | | | | ✓ |
| Form Validation | ✓ | | | | | | |
| Documentation | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Functional requirements | ✓ | | | | | | |
| User Management | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Multiuser | ✓ | | | | | | |
| Permissions Management | ✓ | | | | | | |

Figure 2: Tools comparison.

From this study, it can be concluded that CBD performs much better than the other tools, and its main advantages are the following:

- Collaborative design of forms and project management.
- Expressiveness in the models: it contemplates the relations 1:n, n:m and the reflexive relations 1:n
- Exportation of the the data model in SQL.
- Advanced control panel for expert users (system analysts).
- Possibility of modeling the navigation between forms.
- Exportation of models in XMI format for the integration with team-based modeling environments.

These characteristics illustrate how CBD is a tool that allows both form design and the generation of applications directly by non-expert users. However, unlike the other tools in this field, CBD is not intended only for this use and it can also be used in professional environments, in the same way as commercial modeling tools, and integrated with these tools. In the appendix at the end of this work, the CBD work interface and its most important functions are presented in more detail.

## 3 FUNCTIONAL DEPENDENCIES MANAGEMENT

The normalization theory highlights the importance of the semantic information collected in the FDs of a schema. In (Armstrong, 1974) the author presents the well-known Armstrong's Axioms, a sound and complete inference system for FDs which can be considered the pioneer for other equivalent FD logics(Fagin, 1977; Paredaens, 1982). All of them have a similar pattern, their axiomatic systems are strongly based on the transitivity rule which avoids the development of automated methods directly based on logics.

In (Cordero et al., 2002) a novel inference rule for FDs was presented. This rule is the core of a new sound and complete FD logic named Simplification logic for FDs ($\mathbf{SL}_{FD}$). The new inference system does not include the transitivity rule as a primitive rule. This property allows us to consider Simplification logic as an executable logic and to develop efficient deduction methods directly based on the FD inference system. In the following we summarize $\mathbf{SL}_{FD}$:

**Definition 1.** *We define the* $\mathbf{SL}_{FD}$ *logic as the pair* $(\mathcal{L}_{FD}, \mathcal{S}_{FDS})$ *where* $\mathcal{L}_{FD}$ *is the language*

$$\{X \to Y \mid X, Y \in 2^{\Omega} \textit{ with } \Omega \textit{ being the attributes set}\}\ ^{2}$$

*and* $\mathcal{S}_{FDS}$ *is the following axiomatic system:*
$\lfloor \mathbf{Ax} \rfloor$ *Axiom scheme: if* $Y \subseteq X$

$$\vdash X \to Y$$

$\lfloor \mathbf{FR} \rfloor$ *Fragmentation rule: if* $Y' \subseteq Y$

$$X \to Y \vdash X \to Y'$$

$\lfloor \mathbf{CR} \rfloor$ *Composition rule:*

$$X \to Y,\ U \to V \vdash XU \to YV$$

[2] As usual, $XY$ is used as the union of sets $X, Y$; $X \subseteq Y$ as $X$ included in $Y$; $Y - X$ are the elements in $Y$ that are not in $X$ (difference); and $\to$ is the FD.

$\lfloor$**SR**$\rfloor$ *Simplification rule: if* $X \subseteq U$ *and* $X \cap Y = \varnothing$

$$X \to Y,\ U \to V \vdash U\text{-}Y \to V\text{-}Y$$

Moreover, we have the following derived rule:
$\lfloor$**rSR**$\rfloor$ r-Simplification rule: if $X \subseteq UV$, $X \cap Y = \varnothing$

$$X \to Y,\ U \to V \vdash U \to V\text{-}Y$$

The main characteristic of $\mathbf{SL}_{FD}$logic is the simplification rule, which was originally developed to be applied to the redundancy elimination problem. The inference rules can be rewritten as equivalences that allow reducing the specification

$\{X \to Y\} \equiv \{X \to Y\text{-}X\}$

$\{X \to Y, X \to U\} \equiv \{X \to YU\}$

$\{X \to Y, U \to V\} \equiv \{X \to Y, U\text{-}Y \to V\text{-}Y\}$ if $X \subseteq U$ and $X \cap Y = \varnothing$

$\{X \to Y, U \to V\} \equiv \{X \to Y, U \to V\text{-}Y\}$ if $X \subseteq UV$ and $X \cap Y = \varnothing$

Therefore, applying the $\mathbf{SL}_{FD}$rules, the redundancy (Cordero et al., 2002) can be eliminated, the closures (Mora et al., 2006) can be calculated and the implication problem (Mora et al., 2004) can be solved. This means that by simply applying the inference rules of the logic, these problems can be solved with a similar cost to that of the best algorithms existing in the literature. However in this case using the logic directly, allows us to analyze the reasoning process.

## 4 FUNCTIONAL DEPENDENCIES IN ACTION

From the point of view of the practical implementation of FDs in relational database management systems, it is clear that FD technology has begun to be incorporated in commercial tools. Currently, almost all of them implement the concept of primary key and candidate key, which is stronger than that of FDs.

At present, it is assumed that relational database designers should normalize their tables during the debugging process on the original design, which limits the automation of the design process. If the management systems incorporated directly the concept of FDs, then these systems could be automatically handled and in this way intelligent debugging of the DB could be carried out.

The technology for performing this process has already been developed, but as yet it has not been included in the management systems. In our opinion the reasons for this are two-fold: firstly there is no formal framework for the validation and manipulation of FDs using the logic (Cordero et al., 2002) and secondly there are no tools existing to gather requirements which allow the user or designer to include their knowledge on FDs.

In this work we present an approach to this problem, using the formal framework provided by the $\mathbf{SL}_{FD}$logic and using the information found in the XML dictionary of CBD In this way, we use CBD as a tool to capture the requirements with the form being the main element for the user. We wish to point out that in CBD each form is not moved directly to a table, as is the case for the rest of the form design tools. As we have mentioned, CBD allows a connection between various forms belonging to the same user, allowing greater information control and more possibilities as regards restrictions than if there were only one key per table. This change in the approach allows the existence of FDs among the attributes input by the users to be deduced.

Our approach has been to use $\mathbf{SL}_{FD}$-based methods for FDs about the information contained in the data dictionary of the CBD. Specifically, we carried out the application of the algorithm presented in (Cordero et al., 2002) to eliminate redundancy in the information contained in the XML database. Currently this debugging process of the restrictions stored in the CBD is done externally of the CBD, and it is executed before the CBD generates the unified relational database model. The algorithm uses the CBD database, consulting the metadata specification and extracting the information on the FDs. This information is debugged and returned to the CBD so that a model with less redundancy is generated. This debugging process is efficient, and as presented in (Cordero et al., 2002), it has a lower cost than the other algorithms dealing with the same problem in the bibliography.

## 5 CONCLUSIONS AND FUTURE WORK

In this work, we present CBD, a tool which allows the direct participation of the users. The tool itself has been presented and a comparison made with other tools created for the design of forms and the generation of applications based on this design. In this comparison, it can be seen that CBD is a more powerful tool than the others presented in every sense. Nevertheless our intention is not to bind the use of CBD to the problem of form design, but rather to open it up to a greater use in the area of modeling.

The prototyping and generation of applications from an interface design created by the user is a functionality offered by some solutions. Nevertheless, as far as we know none of them manages FDs nor do they generate UML models. In this way, the user can generate better quality applications and we overcome some of the limitations of other automated application generators. This differentiates CBD from the other tools available, as by using the forms designed by the user, systems analysts can obtain requirements in the initial development phases of an information system. Therefore, CBD is extremely useful as a tool for gathering requirements as it allows extracting knowledge and exporting it for analysis and design processes.

The second contribution of this work is the use of $\mathbf{SL}_{FD}$logic to eliminate redundancy in the XML database, which works as a data dictionary of CBD. This efficient use is possible using an inference system, which provides not only soundness, but also allows us to explain the reasoning which has been followed in the execution of the application. Our goal when designing the $\mathbf{SL}_{FD}$logic was to clear the way for future construction of an automatic technique that systematizes the use of rules of the axiomatic system. In our opinion, it is not enough to know simply whether a FD is redundant or not, we also need to be able to report on which FDs allow that deduction and the $\mathbf{SL}_{FD}$ rules used, something that is not possible if we use the indirect methods that are commonplace in the literature.

As regards future work, we have begun work in two areas:

- We are working to produce a second version of CBD that will generate web applications from its model. This application can be deployed in the user servers, or hosted in CBD servers in a combination of PaaS (Platform as a Service) and SaaS (Software as a Service) models.
- We wish to incorporate the debugging algorithms of the $\mathbf{SL}_{FD}$ logic into the CBD. The objective of this integration is not only to make the tool easier to use, but to be able to use the information the algorithms provide on the reasoning to help the analyst explain the overlaps in the different views of the model.

## REFERENCES

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

Carrillo, A. L., Falgueras, J., Dianes, J. A., and Guevara, A. (2008). A guided interface for web interaction. In *Proceedings of ICEIS 2008 - 10TH International Conference On Enterprise Information Systems.*, pages 70–77.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387.

Cordero, P., Enciso, M., Guzmán, I. P., and Mora, A. (2002). Slfd logic: Elimination of data redundancy in knowledge representation. *Lecture Notes in Artificial Intelligence*, 2527:141–150.

Evaluation, B. (1000). This is a paper written by one of the authors of this paper. In *We have remove this data following the submission guidelines for authors*. Published.

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

FormLogix.com (2010). `http://www.formlogix.com`.

FormSpring LLC (2010). www.formspring.com.

Groupe W4 S.A (2010). `http://www.lyria.com`.

Guevara, A., Caro, J. L., Leiva, J. L., and Gmez, J. L. (2007). I. comis: Cooperative methodology for information systems. In *Proceedings of ENC'2007 Advances in Computer Science*, pages 81–87.

Infinity Box Inc. (2010). `http://wufoo.com/`.

Interlogy LLC (2010). `http://www.jotform.com/`.

Jacobson, I., Booch, G., and J., R. (2000). *El proceso unificado de desarrollo de software*. Addison Wesley.

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

Mora, A., Aguilera, G., Enciso, M., Cordero, P., and Guzmán, I. P. (2006). A new closure algorithm based in logic: Slfd-closure versus classical closures. *Inteligencia Artificial. Revista Iberoamericana de IA*, 31:31–40.

Mora, A., Enciso, M., Cordero, P., and Guzmán, I. P. (2004). The functional dependence implication problem: optimality and minimality. An efficient preprocessing transformation based on the substitution paradigm. *Lect. Notes in Artificial Intelligence*, 3040:136–146.

Nelson, H. J., Poels, G., Genero, M., and Piattini, M. (2005). Quality in conceptual modeling: five examples of the state of the art. *Data Knowl. Eng.*, 55(3):237–242.

Paredaens, J. (1982). A universal formalism to express decompositions, functional dependencies and other constraints in a relational database. *Theoretical Computer Science*, 19 (2):143–160.

Pressman, R. (2006). *Ingenieria del Software*. McGraw Hill.

Veer West LLC (2010). www.formassembly.com.