Acta Universitatis Sapientiae
Electrical and Mechanical Engineering, 2 (2010) 63-72

Application Development in Database-Driven Information Systems

Marius MUJI

Department of Electrical Engineering, Faculty of Engineering, Petru Maior University of Tîrgu Mures, Tg. Mureş, e-mail: marius_muji@yahoo.com

Manuscript received October 1, 2010; revised October 18, 2010.

Abstract: The relational model provides extensive support for data integrity constraints (i.e. business rules) specification, as an integral part of the data model. Current Relational Database Management Systems (RDBMS), however, cover just partially the various categories of data integrity constraints, mostly those directly related with the database structure (e.g. entity integrity, referential integrity). The rest of them are delegated to the application languages. Consequently, they are usually defined in a function-oriented approach (e.g. the object-oriented technology), loosing their direct link with the data model – with all the negative consequences in terms of system scalability and logical data independence. The present paper proposes a data-oriented approach for the development of the external level of database systems. Under the proposed model, the external data is structured only by means of ordered sets of tuples (i.e. arrays of tuples), and the corresponding business rules (i.e. the presentation rules) are treated as external schema integrity constraints. Consequently, the application developer is able to define the user views of the system in a declarative fashion, similar to the relational database design. The immediate advantage is that he or she gains a data designer perspective, rather than one of a programmer. The essentiality (i.e. the unique data constructor) of the model facilitates a seamless integration with the relational model, an entity-relationship graphical representation, and the complete automation of the user interface development.

Keywords: Logical design– data models, schema and subschema.

1. Introduction

Database-driven information systems are developed around an *integrated* and *shared* source of data. The integration is important when somebody needs a general view of the system: for example, a manager who wants to track an item from the supplier to the end-client, spanning the procurement, production and sales activities of a company. This is why, regardless how many individual views we have about an organization's data, there is always needed an integrated, general view of the entire database. On the other hand, it is also important for initial system development *and* for long-term data management purposes to work with data representations which are not dependent on the physical storage equipment.

These requirements led to the ANSI/SPARC three levels architecture [2, 3] (see *Fig. 1*), which makes a clear distinction between the *physical* and the *conceptual* (i.e. logical) representation of the system, and between the general, integrated *community view* and the *individual views* of the system, respectively. The physical-logical separation provide *physical data independence*, which basically means that the applications would not be affected by changes at the physical data representations (for hardware upgrade purposes, for example); the community-individual views separation provides *logical data independence*, which means that the system could grow (through some new user views or modification of the existent ones) without affecting the applications corresponding to the user views that remain unchanged.

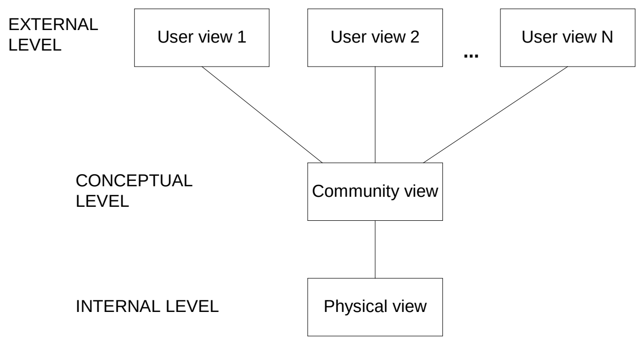

Figure 1: The three levels architecture.

The relational model provides the theoretical support for the development of information systems in accordance with the three levels architecture. Thus, Relational Database Management Systems are currently the technology of choice for the development of the physical and conceptual level, sharing with the application languages the development of the external level.

In this context, the database professionals are traditionally responsible for:

- the data structures at the conceptual and physical level;
- some of the integrity constraints at the conceptual level (i.e. the *database rules*), like: type constraints, entity integrity constraints, referential integrity constraints;
- some of the data structures of the external level (e.g. relational views, parameterized relation-valued operators [7]/stored procedures).

The application professionals are, in turn, responsible for:

- the remaining part of the integrity constraints for the conceptual/community data (i.e. the *application rules*);
- all the data structures of the external views – even when the DBMS provides a layer of data at the external level (e.g. relational views), the application languages need to redefine the entire external view using their own data constructs;
- the *presentation rules* [6] implementation, i.e. the end-user interface, including CRUD (create, retrieve, update, and delete) operations, and display customization (e.g. field labels, field alignment, background and foreground colors, etc.).

The current trend in application development is determined by a significant pressure coming from the programming community, which promotes an object oriented approach for the entire architecture of the information system. Consequently, the data structures and the business rules are usually defined in *a function-oriented approach* [12], loosing their direct link with the *data model* [6] – with all the negative consequences in terms of *system flexibility* and *logical data independence* [7].

By contrast, we propose a data-oriented approach for the development of the information systems, including the external level of the ANSI/SPARC architecture. Thus, we defined a *presentation model*, which preserves the *essentiality* of the relational model [4], i.e. the existence of a *unique data constructor* (in our case, the array of tuples), and prescribes a declarative solution for the presentation rules specification, perceived as *external view integrity constraints*.

The model introduces a clear separation between the display-related presentation rules (e.g. field labels, field alignment, background and foreground colors, etc.), and data-related presentation rules (e.g. data filtering, master-detail navigation, data ordering). CRUD operations are accomplished through the standard behavior of the array constructor. For any CRUD operation initiated by the end user, the system initiates automatically the invocation of some operators from the underlying levels, which actually realize the *mapping* between the presentation level and the lower levels of the system (i.e. the lower external sub-levels and/or the conceptual level – see *Fig. 2*).

Section 2 provides a discussion about the external level of a database-driven system. Section 3 presents our presentation level modeling approach, followed by an example in Section 4. We conclude with the advantages of the proposed approach, and some possible applications.

2. The external level - a closer look

While the external level of the three levels architecture is usually split in multiple sub-levels [3] (see *Fig.2*), *the presentation level* of the system is actually the outermost sublevel, which contains the external data as seen by the end user (i.e. the *external views* of the system).

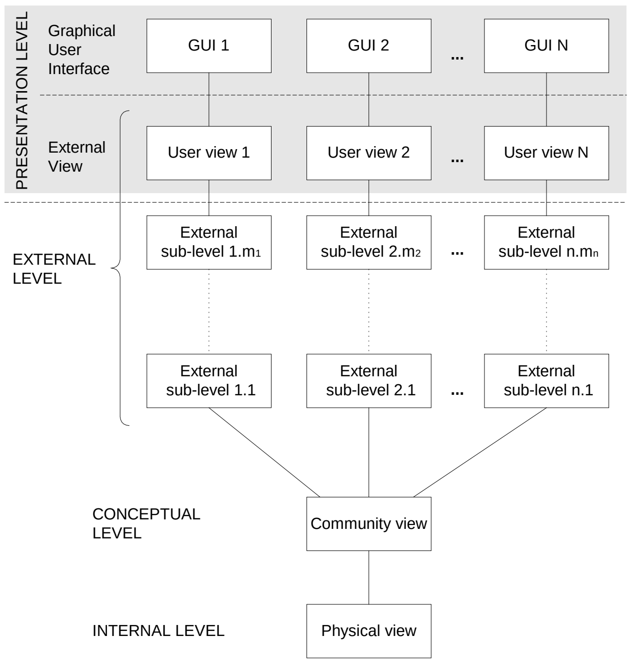

Figure 2: The external sub-levels of the system.

Some of the external sub-levels, i.e. those closer to the conceptual level, are usually implemented under the relational model, through relational views and/or relational operators (e.g. stored procedures). The external sub-levels closer to the end user are built under the theoretical model employed by the application languages (in most cases, object-oriented). The well-known *impedance mismatch* issue is in fact a measure for the lack of compatibility between the two theoretical models. The major difference is determined by the switch of focus from data to function: the data constructs defined by the database designers are spread among multiple function-oriented software constructs by the application developers [13].

The majority of the mapping solutions employed today to overcome the impedance mismatch have the aim to provide the application developer with the means for accessing the lower (relational) levels of the system transparently, using only the concepts and tools specific to the application languages. Even when some specific concepts of the conceptual level models are introduced (e.g. data entities and relationships) [1], the main purpose is to ‘push’ the mapping layer as ‘low’ as possible.

We follow the opposite approach, which considers that the relational model is better suited not only to design the persistent data structures of the conceptual level, but also to build *and* manipulate the data structures of the external level.

However, the end user’s perception of data often implies the existence of a *current element* and a certain *inspection order* for a given set of data. It follows that, at least for the presentation level, there is a need for some non-relational features. At the same time, we consider that the essentiality of the relational model (i.e. the existence of a unique, *essential*, data constructor [4]) would provide, also, at the presentation level important advantages related to impedance mismatch and interface automation. Consequently, our model considers the array of tuples as its *unique* data constructor. It could have been a list, or any other collection type, as well – to cite from reference [7], any preference is just “a purely psychological decision – there is no logical reason for preferring (say) an array over a list”.

3. The user view from a data oriented perspective

From the end user’s point of view, the general behavior of a typical application consists on a limited set of actions related to data entities. In fact, there are just four basic actions, or data-function interfaces [12], classically known as CRUD operations: create, retrieve, update, and delete.

Since it requires a more complex analysis, we’ll discuss first the issues related to *data retrieval*. In this regard, the end user can take the following typical actions:

1. **to identify one element in a set**, by means of some unique property or set of properties which distinguishes that particular element from all the rest in the set;
2. **to determine a subset of a set**, based on some *filtering criteria*, namely some common properties of the subset elements;
3. given one element in a set A, and an existing relationship defined from A to B, **to identify all the related elements** in B, under the rule that defines the relationship (e.g. master-detail navigation);

4. **to display the elements of a set in a particular order.**

Let us consider that *all* the data ‘seen’ by the end user through one particular user view, is composed at any given time by ordered sets of tuples (i.e. arrays of tuples). The user may also be aware about some existing relationships between two sets, under the definition provided on the reference [5]: “Let A and B be sets, not necessarily distinct. Then the relationship from A to B is a rule pairing elements of A with elements of B.” Note that we discuss about *directed* relationships, so a relationship defined from A to B will be different from another relationship defined from B to A.

If we consider that any filtering value, which is to be applied to the set X, is seen as an element of another set Y, when a relationship was defined from Y to X, then *the rule which defines the relationship is the filter itself*. Similarly, a change of the display order of a set A may be also accomplished by changing the current element of another set B (which contains the ordering sequences of choice), when a relationship is defined from B to A. Based on the relationship definition, and on the current element of B, the system will reorder the set A accordingly (more accurate: the ordered collection representing A is *(re)created* based on the relationship definition).

Under this approach, we are able to design the entire presentation level only by means of (ordered) sets and relationships between sets. The processing of all the data requests at the presentation level is hidden inside the defining rules for set relationships. The only functionality kept at the presentation level is related to the automatic enforcement of the relationship rules, i.e. the automatic recall of the defining operator attached to the dependent array, when the current element of the parent array changed its value.

Considering the **update operations** (i.e. insert, update, and delete), our presentation model does not require special features, other than the existing data access solutions employed by the application languages. However, in order to preserve the uniformity of the model, and to increase the level of logical data independence, the recommended solution implies the existence of a level of update operators (e.g. stored procedures, or any other application procedures), at the interface with the underlying levels of the system. At the presentation level, we’ll have to declare the procedure’s name, and the name and type of its parameters.

4. An example

The following example is inspired from the chapter about presentation rules in reference [6]. Some details were added to enable a better presentation of our approach (see *Fig. 3*).

Suppose that we have a user view that exposes to the end user data about customers, orders, and order details. Suppose that the user will have to be able to see at any time all the customers which simultaneously satisfy the following conditions:

- they have a credit limit less than a certain value;
- they are located in a specific region;
- they can be ordered by name, by credit limit, or by the total value of their orders;
- customers whose accounts are overdue must be displayed in red.

Likewise, the user should be able to see, also, at any time, the orders which simultaneously satisfy the following conditions:

- they belong to the current customer;
- their issuing date is in a certain period, say after a start_date and before an end_date, specified by the user;
- they can be ordered by date, value-ascending, or value-descending;
- rush orders must be displayed before regular orders.

When the user inspects a specific order, the system should provide all the order_details that belong to that particular order. Those details should be displayed in their part number order.

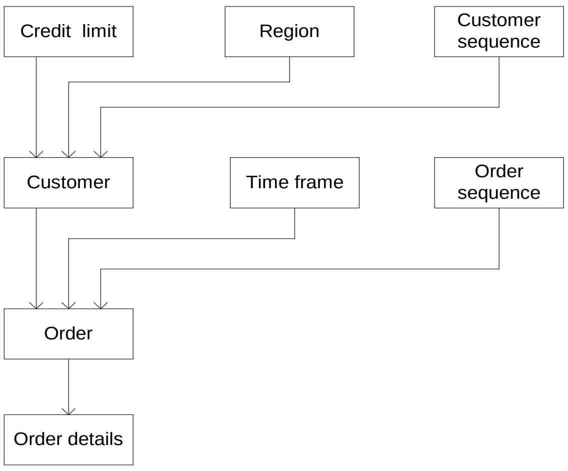

Figure 3: A *user view* example.

In Figure 3, *all* data structures are arrays. Some of them represent application data (i.e. filtering and/or ordering conditions), like *credit limit*, *customer sequence*, *time frame, order sequence.* They are not dependent on other data, so their defining functions don't have parameters.

The array named *region* takes its values from the conceptual level (possibly through a relational view), but its content doesn’t depend on any other data structure from the user view.

The customer data contained by the *customer* array depends on the current region chosen by the user from the *region* array, on the current customer sequence chosen by the user in the *customer sequence* array, and also on the value provided by the user in the *credit limit* array (the credit limit array will be a special case of an array with one tuple and one attribute, but still an array and not a simple scalar variable, in order to preserve the *essentiality* of the external view model). This is why the defining operator of the customer array should have three parameters, which will automatically take their values at run time from the current tuples in the *region, customer sequence*, and *credit limit* arrays, respectively, at any refresh of the customer data.

The list of customer orders exposed to the user at a given moment, contained by the *order* array, depends on the current elements of the *customer* array, the *time frame* array, and the *order sequence* array. Consequently, the defining operator of the order array should have at least three parameters, one for every parent array. In fact, for the present example, we may consider four parameters: one for the link with the customer array (e.g. customer_id), two for the link with the time frame array (e.g. start date, and end date), and one for the link with the order sequence array (e.g. order_sequence_no).

As required, the *order details* array will contain at any moment all the details of the current order from the *order* array. The rule that the details should always be ordered by their part number is specified *inside* the defining function of the order details array, and will remain transparent at the user view design level.

We should also be able to provide solutions for the presentation rules that are not related with *relationship definitions*:

- “customers whose accounts are overdue must be displayed in red” – for this rule, we need to introduce an attribute in the *customer* array, which would allow the distinction of the ‘red’ customers, so that, at the display level, while defining the graphical object (e.g. the grid, or the list) which displays the customers data, we’ll be able to incorporate this presentation rule in a straightforward manner (i.e. declaratively, if possible);
- “rush orders must be displayed before regular orders” – this rule is implemented *inside* the defining function of the *order* array (which is completely transparent for our model) .

So, under the proposed model, the developer is able to design the presentation level declaratively, just specifying:

- the declaration of all the array structures: array name, attribute names, data types;

- the defining operator of every array;
- the link between every parameter of any defining operator and its corresponding attribute from the parent array;
- the update procedures, their triggering events, and the links of their parameters with the corresponding attributes.

Our user view's dependency graph was represented graphically in *Fig. 3* using arrows, oriented from parent to child, but it could have been used any other entity relationship graphical notation (e.g. crow foot, IDEF1X, IE, etc.). Thus, the user views will have the same (E-R like) graphical representation as the conceptual level. The only difference is that instead of *foreign key relationships*, we have pairs of defining operator *parameters* and *attributes* of the parent array(s).

5. Conclusions

There is a clear need for a data-oriented approach in application engineering. The software engineering field is now dominated by the new trend introduced by the OMG's Model Driven Architecture [14], which has a strong object oriented bias. The position sustained by this paper is that the application development should be *not only model-driven, but data-model-driven* [10, 11]. The paper introduces a data-oriented model for the development of the external level of database systems, which considers the *presentation level* as the only required data layer above the relational data model. Moreover, this should be a thin layer, with the unique purpose of data presentation, which doesn't need to address any business logic other than the *presentation rules* [6].

The *standard behavior* and the *essentiality* of our model enable the automation of the presentation level development. At the same time, the *mapping operators* (defined at the lower levels and called at the presentation level to promote the CRUD operations to the conceptual level) are the key for the provision of logical data independence at the presentation level. This constitutes the major step forward from the previous attempts to automate the interface, which failed to provide an appropriate degree of logical data independence at the external level of the system. Trying to generate the interface based on various entity-relationship patterns existent at the conceptual level, and assuming that the user views are just sub-schemas of the conceptual level [15, 16, 18], they become useless as soon the external level has multiple sublevels, i.e. the presentation data is obtained from the conceptual data through a series of complex operations – which is always the case for large, integrated information systems.

The foreseen applications of the presentation model are related primarily to the application development for database-centric systems (e.g. enterprise resource planning systems, e-commerce systems, etc.). CASE tools which support entity-relationship diagrams represent, also, an important area for our model implementation.

Future work will concentrate primarily on the development of interface automation tools, designed in an object-oriented approach, and implemented with general purpose third-generation languages (e.g. Java, C#). In a long term vision, the presented model could be used in data-model driven methodologies for declarative development of database-centric applications.

References

[1] Adya, A., Blakeley, J. A., Melnik, S., and Muralidhar, S., "Anatomy of the ADO.NET entity framework", *ACM SIGMOD International Conference On Management Of Data. Beijing, China*, 2007, pp. 877-888.

[2] ANSI/X3/SPARC Study Group on Data Base Management Systems. "Interim Report", *ACM SIGMOD Bulletin*, no. 2, 1975.

[3] Date, C. J., "An Introduction to Database Systems (8th edition)", Addison-Wesley, 2003.

[4] Date, C. J., "Date on Database: Writings 2000-2006", Apress, 2006.

[5] Date, C. J., "Logic and Databases: The Roots of Relational Theory", Trafford Publishing, 2007.

[6] Date, C. J., "What Not How: The Business Rules Approach to Application Development", Addison-Wesley, 2000.

[7] Date, C. J., and Darwen, H., "Foundation for Future Database Systems: The Third Manifesto (2nd Edition)", Addison-Wesley, 2000.

[8] Halle, B., "Business Rules Applied: Building Better Systems Using the Business Rules Approach", Wiley, 2001.

[9] Hay, D. C., "Data Model Views", *The Data Administration Newsletter - TDAN.com*, Apr. 2000.

[10] Lewis, B., "Data Lineage: The Next Generation", *The Data Administration Newsletter - TDAN.com*, Aug. 2008.

[11] Lewis, B., "Data-Oriented Application Engineering: An Idea Whose Time Has Returned", *The Data Administration Newsletter - TDAN.com*, Jan. 2007.

[12] Lewis, W. J., "Data Warehousing and E-Commerce", Prentice Hall PTR, 2001.

[13] Lewis, W. J., "E-Commerce Vs. Data Management", *The Data Administration Newsletter – TDAN.com*, Jan. 2002.

[14] Model Driven Architecture. http://www.omg.org/mda/

[15] Pizano, A., Yukari, S., and Atsushi, I., "Automatic generation of graphical user interfaces for interactive database applications", *Conference on Information and Knowledge Management, Washington, D.C.*, 1993, pp. 344-355.

[16] Rollinson, S. R., and Roberts, S. A., "A mechanism for automating database interface design, based on extended E-R modelling", *Advances in Databases. s.l. : Springer Berlin / Heidelberg*, 1997, pp. 133-134.

[17] Ross, R. G., "Principles of the Business Rule Approach", Addison-Wesley Professional, 2003.

[18] Rowe, L. A., and Shoens, K. A., "A form application development system", *ACM SIGMOD International Conference On Management Of Data., Orlando, Florida*, 1982, pp. 28-38.