Scientific Bulletin of the "Petru Maior" University of Tîrgu Mureş
Vol. 12 (XXIX) no. 2, 2015
ISSN-L 1841-9267 (Print), ISSN 2285-438X (Online), ISSN 2286-3184 (CD-ROM)

INTEGRITY CONSTRAINTS FOR THE DATA-ORIENTED DESIGN OF THE USER INTERFACES

Marius MUJI
"Petru Maior" University of Tîrgu Mureş
Nicolae Iorga Street, no. 1, 540088 Tîrgu Mureş, Romania
marius_muji@yahoo.com

Abstract

Declarative development of the presentation level of database-driven information systems was always a main goal for the development technologies. However, it is just partially attained, for a limited set of design patterns, while the general approaches still rely on procedural code. This paper formally defines a set of integrity constraints, as part of a presentation-purpose logical data model. The automatic enforcement of the respective integrity constraints constitutes the declarative support for any implementation technology of the proposed presentation model.

Key words: Logical design, data models, declarative specifications

1. Introduction

Database-driven information systems architecture has traditionally two different logical layers [1] [2]: one that holds all the specific user perspectives (*user views* – UV) on the information system, the other that integrates all those particular perceptions about the "universe of discourse" [3] into one *community view* (CV).

Since it holds the unified representation of the system's persistent data, the community view was historically developed by database professionals, being based on strong theoretical grounds. In order to gain physical data independence, the community view was split into a conceptual level and a physical/internal level. The most important theoretical contribution that facilitated the development of a class of technologies capable to implement these architectural levels is the relational model [4] [5] [6]. The fact that the great majority of the current database management systems (DBMSs) rely on its mathematical definition ensures a high degree of physical data independence, facilitating data portability, and, not least, a common mindset for the database professionals. Moreover, an important part of the community view can be expressed declaratively, through the essential data structure of the relational model (i.e., the relation), a set of integrity constraints (e.g., primary keys, foreign keys), and a set of mathematical operators known as the "relational algebra" [2] [7].

On the other hand, the user views of the system are built using numerous development technologies, from general-purpose languages (e.g., C, Java) to domain specific languages (DSLs) [8] [9] and front end development frameworks, like Microsoft *.Net* [10], Oracle Application Development Framework [11], Eclipse [12], and many others. Although they strive to provide a high level of abstraction, through declarative facilities and object oriented features, these technologies are not able to ensure the same level of data independence as the relational DBMSs, since they are not built around a logical data model with a mathematical definition. Consequently, at some point of the system's development, the programmers still have to write procedural code, in order to implement all the "presentation rules" [13] expected by the business user.

The main purpose of our paper is to extend the formal definition of a presentation-purpose logical data model [14] [15], through a set of integrity constraints. The proposed constraints will provide the theoretical ground for the declarative specification of the presentation component of the system.

Section 2 discusses similar approaches; Section 3 contains a brief presentation of the considered data model, in terms of data structures and logical operators; Section 4 provides the formal definitions of the proposed integrity constraints, and Section 5 concludes with considerations about the scope of the presentation model.

2. Current technologies from a conceptual perspective

There are many technologies available today for the development of the user views in database-driven information systems. At the implementation level,

they have to comply with various technical requirements, usually related to the compatibility with some specific software platforms. However, from a conceptual point of view, they all strive to provide similar features for the automation of the development process. Considering the facilities for declarative development, there are two main approaches followed by all these technologies.

The first approach is related to the attempt to generate the user view metadata (i.e., the presentation level data structures) from the conceptual schema of the community view. Thus, the entities from the database level are (automatically) transformed in data collections specific to a particular development technology. The tools employed for this task are called object-relational mappers [16] [17] [18] [19], and their main purpose is to synchronize the metadata of the community view with the metadata of the user view.

In many cases, the semantic complexity of the community data, which have to integrate a large number of user views, requires a high level of generalization for the data entities of the database. Consequently, the transformations/mappings between the database entities and the data structures of the user views will have to incorporate the whole semantic complexity of the generalization/ specialization process. For this reason, the automation facilities provided for simple one-to-one mappings cannot be used in systems with a higher level of sematic complexity.

The second approach that facilitates the declarative development of the information system is represented by the usage of a data model as theoretical support for the conceptual representation of the user views. The typical example in this regard is the ADO.NET Entity Framework [20] [21], which implements a version of the entity-relationship model at the application level. The main objectives of this approach are:

- To minimize the impedance mismatch, based on the natural match between the application 'entities' and the relational data structures;
- To raise the level of abstraction in application design, through integrity constraints (declarative) specification.

While the first objective is attained by native metadata compatibility, the second objective is just partially achieved, due to the fact that the Entity Data Model comes with a limited set of integrity constraints (like *keys* and *relationships* [21]), conceived mostly to express static properties of (persistent) data, but not as much appropriate for the specification of all the dynamic aspects related to data presentation.

Our Presentation Model follows the same objectives, but it is more limited in scope (the logical specification of the presentation level) and introduces a set of (presentation specific) integrity constraints, as key ingredients for the declarative specification of the most common presentation rules.

An important aspect which has to be emphasized, related to the introduction of a formal data model at the presentation level of the system, is the fact that it usually enables graphical representations, like those expressed by the entity-relationship diagrams used in database design. This can bring multiple advantages related to the computer aided design of the system, and, not at least, can shift the programmer's mindset from "code writing" to "system design".

3. Data Structures and Logical Operators

The presentation model defines a unique, essential, data constructor: the "array of tuples" [7]. The formal definition of the array is based in the mathematical formalism introduced by Bert de Brock and Frans Remmen [22] [23], and described by Lex de Haan and Toon Koppelaars [24].

In terms of the considered mathematical formalism, the array is defined as follows [15].

If T is considered to be a table on the set H, then:

$$
\begin{aligned}
&\text{"AR is an } \textit{ARRAY}\text{"} \Leftrightarrow \text{AR} = (\text{current};\text{T}) \wedge \\
&\quad \text{'seq\_no'} \in \text{H} \wedge \\
&\quad (\forall t \in \text{T}: t(\text{seq\_no}) \in \mathbb{N} \setminus \{0\} \wedge \\
&\quad\ t(\text{seq\_no}) \leq \#\text{T}) \wedge \\
&\quad (\ \forall t_1, t_2 \in \text{T}: t_1 \neq t_2 \Rightarrow \\
&\quad\ t_1(\text{seq\_no}) \neq t_2(\text{seq\_no})\ ) \wedge \\
&\quad (\ \text{T} \neq \varnothing \Rightarrow \text{current} \in \{t(\text{seq\_no}) \mid t \in \text{T}\}\ ) \wedge \\
&\quad (\ \text{T} = \varnothing \Rightarrow \text{current} = 0\ ).
\end{aligned}
$$

Since formal definition the array is based on the formal specification of a table (i.e. relation), the presentation model can employ all the operators contained by the relational algebra. Besides those classical operators, a set of array operators are needed [15]:

- Cardinality ($\#_a$)
- The Extract Attribute Value operator (get_att_val)
- The Extract Current Tuple operator (get_tuple)
- The Get Cursor operator (get_current)
- The Set Cursor operator (set_current)
- The Array to Table conversion operator (A2T)
- The Table to Array conversion operator (T2A)
- The Tuple Insert operator (insert_tuple)
- The (Current) Tuple Update operator (update_tuple)
- The (Current) Tuple Delete operator (delete_tuple)
- Next()
- Prior()
- First()
- Last()

It has to be said that the last four operators, named also *navigation operators*, are just shortcuts for some expressions involving the operators *get_current()* and/or *set_current()*:

next(AR) := set_current(AR, get_current(AR)+1),
if get_current(AR)≠#$_a$(AR);
:= AR, otherwise.
prior(AR) := set_current(AR, get_current(AR)-1),
if get_current(AR)>1;
:= AR, otherwise.
first(AR) := set_current(AR, 1), if π_2(AR) ≠ ∅;
:= AR, otherwise.
last(AR) := set_current(AR, #$_a$(AR)).

4. Integrity Constraints

The user view skeleton is formally defined like a set-valued function.

Definition: "UV_S : AR → ℘(AT) is a user view skeleton" ⇔

"AR is the set of the array structure names contained in the user view" ∧

"AT is the set of the attribute names of the arrays contained in AR" ∧

/* array-specific requirement */

(∀AR$_i$∈AR: seq_no∈UV_S(AR$_i$)).

The characterization of an array structure is formally defined like a set-valued function.

Definition: "chr_AR$_i$: UV_S(AR$_i$) → ℘(D) is a characterization for the array structure AR$_i$" ⇔

"D is the set of all the possible values of the attributes contained in AT" ∧

/* array-specific requirement */

(chr_AR$_i$(seq_no) ⊆ ℕ\{0}).

Note: every element of the function chr_AR$_i$, i.e., every attribute-domain pair, represents an ***attribute constraint***, or an *a priori* constraint.

For every array structure, it is defined a ***tuple universe***, as the set of all the admissible tuples for the given array structure.

tup_AR$_i$:= { t | t∈Π(chr_AR$_i$) ∧
P$_1$(t) ∧ P$_2$(t) ∧ … ∧ P$_n$(t) },

where:

P$_i$(t) – represent *tuple predicates*, or ***tuple constraints***.

For every array structure, it is defined a ***table universe***, as the set of all the admissible tables for the second coordinate of the array with the given array structure.

tab_AR$_i$:= { TAB | TAB∈ ℘(tup_AR$_i$) ∧
/* array-specific requirement */
(∀t∈TAB: t(seq_no) ≤ #T) ∧
(∀t$_1$, t$_2$∈T: t$_1$≠t$_2$ ⇒
t$_1$(seq_no) ≠ t$_2$(seq_no)) ∧
/* user defined constraints */
P$_1$(TAB) ∧ P$_2$(TAB) ∧ … ∧ P$_n$(TAB) },

where:

P$_i$(TAB) – represent *table predicates*, or ***table constraints***.

For every array structure, it is defined an ***array universe***, as the set of all the admissible arrays for the given array structure.

arr_AR$_i$:= { ARR | ARR∈ ℕ x tab_AR$_i$ ∧
/* array-specific requirement */
(π_2(ARR)≠∅ ⇒
π_1(ARR)∈{t(seq_no) | t∈π_2(ARR)}) ∧
(π_2(ARR)=∅ ⇒ π_1(ARR)=0) }.

The ***user view characterization*** is formally defined as a set valued function:

UV_CHR : AR → (arr_AR$_1$ ∪ arr_AR$_2$ ∪ …
∪ arr_AR$_n$),

where:

UV_CHR(AR$_i$) = arr_AR$_i$,
∀i: 1≤ i≤ n ∧ AR$_i$∈AR.

The ***user view universe*** is defined as the set of all admissible states of the user view, as follows.

UV_U := { uvs | uvs∈Π(UV_CHR) ∧
P$_1$(uvs) ∧ P$_2$(uvs) ∧ … ∧ P$_n$(uvs) },

where:

P$_i$(uvs) – represent *user view predicates*, or ***user view constraints***.

The state transition universe of the user view has the following formal definition:

ST_UV_U := { (uvs$_1$;uvs$_2$) |
uvs$_1$,uvs$_2$∈UV_U ∧
P$_1$(uvs$_1$, uvs$_2$) ∧
P$_2$(uvs$_1$, uvs$_2$) ∧ … ∧
P$_n$(uvs$_1$, uvs$_2$) },

where:

P$_i$(uvs$_1$, uvs$_2$) – represent *user view state transition predicates*, or ***user view state transition constraints***.

All the integrity constraints defined so far are defined as close as possible from the corresponding database definitions. The part of the system which doesn't have yet a formal specification is represented by *the transformations/mappings* between the integrated *community view* (i.e., the database) and the considered *user view* of the system.

The presentation model formally defines them as part of two ***State Transition Universes of the System***:

- ***The Update Transitions Universe of the System*** (ST_System_Update_U);
- ***The Refresh Transitions Universe of the System*** (ST_System_Refresh_U).

The first is meant to specify the UV→ DB transformations; the second is meant to specify the DB → UV transformations. Any transaction initiated by the end user at the *user view* level should be accepted if and only if ***all*** the integrity constraints specified as part of:

- The User View Universe (UV_U);
- The State Transition Universe of the User View (ST_UV_U);
- The Database Universe (DB_U);
- The State Transition Universe of the Database (ST_DB_U);
- The Update Transitions Universe of the System (ST_System_Update_U);
- The Refresh Transitions Universe of the System (ST_System_Refresh_U);

are satisfied.

How all these integrity constraints will be enforced is a matter of implementation and should be automatically determined by the system.

The enforcement of the *database constraints* (defined at the DB_U and ST_DB_U level) is outside the scope of the presentation model.

The ***Update Transitions Universe*** is defined based on a formal specification of a *system state*, represented as an ordered pair, *(dbs;uvs)*, whose first element is a database state and the second, a user view state.

ST_System_Update_U := {
$((dbs_1;uvs_1);(dbs_2;uvs_2))$ |
$dbs_1,dbs_2 \in DB\_U \wedge uvs_1 \in UV\_U \wedge$
$uvs_2 \in UV\_U^* \wedge$
$(dbs_1=dbs_2 \vee (dbs_1;dbs_2) \in ST\_DB\_U) \wedge$
$P_1((dbs_1;uvs_1), (dbs_2;uvs_2)) \wedge$
$P_2((dbs_1;uvs_1), (dbs_2;uvs_2))$
$\wedge \ldots \wedge P_n((dbs_1;uvs_1), (dbs_2;uvs_2))$ },
where:

$P_i((dbs_1;uvs_1), (dbs_2;uvs_2))$ – represent *system update predicates*, or ***system update constraints***;

UV_U^* has the same definition as UV_U, but without the *user defined tuple constraints* and *user defined table constraints.*

$UV\_U^* := \{ uvs \mid uvs \in \Pi(UV\_CHR^*) \}$,
where:
$UV\_CHR^*(AR_i) := arr\_AR_i^*, \forall AR_i \in AR$;
$arr\_AR_i^* := \{ ARR \mid ARR \in \mathbb{N} \times tab\_AR_i^* \wedge$
/* array-specific requirements */
$(\pi_2(ARR) \neq \varnothing \Rightarrow$
$\pi_1(ARR) \in \{t(seq\_no) \mid$
$t \in \pi_2(ARR)\}) \wedge$
$(\pi_2(ARR) = \varnothing \Rightarrow \pi_1(ARR) = 0) \}$;
$tab\_AR_i^* := \{ TAB \mid TAB \in \wp(tup\_AR_i^*) \wedge$
/* array-specific requirements */
$(\forall t \in TAB: t(seq\_no) \leq \#T) \wedge$
$(\forall t_1, t_2 \in T: t_1 \neq t_2 \Rightarrow$
$t_1(seq\_no) \neq t_2(seq\_no)) \}$;
$tup\_AR_i^* := \{ t \mid t \in \Pi(chr\_AR_i) \}$.

The Refresh Transitions Universe of the System is formally defined as follows.

ST_System_Refresh_U := {
$((dbs;uvs_1);(dbs;uvs_2))$ |
$dbs \in DB\_U \wedge uvs_1 \in UV\_U^* \wedge uvs_2 \in UV\_U$
$\wedge P_1((dbs;uvs_1), (dbs;uvs_2))$
$\wedge P_2((dbs;uvs_1), (dbs;uvs_2))$
$\wedge \ldots \wedge P_n((dbs;uvs_1), (dbs;uvs_2))$ },
where:

$P_i((dbs;uvs_1), (dbs;uvs_2))$ – represent *user view refresh predicates*, or ***user view refresh constraints***.

To ensure the consistency of the system (DB+UV), some *compensatory updates* should be performed. After their completion, the arrays of the user view will take the values returned by the queries specified in their corresponding *refresh constraints* definition. The sequence in which those queries will be (automatically) performed can be determined from a *dependency graph*, automatically built, based on the arguments declared at design time for the operator *get_att_val()*, as part of the *refresh constraints* specification. The dependency graph is directed and should be acyclic. The fact that the dependency graph is acyclic should be verified and enforced at design time.

5. Conclusions

The presentation model is meant to be a *logical data model* (like the *relational model*), appropriate for the *user interface* specification. The model provides support only for the category of user views exposed to the end user, usually through graphical user interfaces. However, it is orthogonal to the graphical representation of data.

The model can't be used for the logical formulation of ad-hoc queries; for this, we usually don't need to 'leave' the relational model. Our model is limited to the presentation function, where there is no need for complex data manipulation capabilities (which have a better support in a relational context, at the database level). In this respect, the prescribed operators of the model should be used, at design time, only for integrity constraints specification.

The behavior expected at the user interface level (e.g., data filtering, data ordering, master-detail navigation, etc.) will be achieved only through (automatic) integrity constraints enforcement. In this respect, "compensatory updates" [2] (like the CASCADE updates/deletes for the foreign keys, in the relational model) should be used extensively.

References

[1] ANSI/X3/SPARC Study Group on Data Base Management Systems, "Interim Report 1975".

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

[3] Joost J. van Griethuysen (May 2009), "The Orange Report ISO TR9007 (1982–1987) — Part 2 ~ The Seven Very Fundamental Principles," *Business Rules Journal*, vol. 10, no. 5.

[4] E. F. Codd (1970), "A relational model for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377-387.

[5] E.F. Codd (1979), "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems (TODS)*, vol. 4, no. 4, pp. 397-434.

[6] Edgar F. Codd (February 1980), "Data models in database management," in *The 1980 workshop on data abstraction, databases and conceptual modeling*, New York.

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

[8] Debasish Ghosh (July 2011), "DSL for the uninitiated," *Communications of the ACM*, vol. 54, no. 7, pp. 44-50.

[9] Rafael Z. Frantz (January 2009), "A DSL for enterprise application integration," *International Journal of Computer Applications in Technology*, vol. 33, no. 4, pp. 257-263.

[10] Microsoft Corporation (2015, November) Microsoft.Net. [Online]. http://www.microsoft.com/net

[11] Oracle (2015, November) Oracle ADF. [Online]. http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html

[12] Eclipse Foundation (2015, November) Eclipse. [Online]. https://www.eclipse.org/

[13] C. J. Date (2000), *What Not How: The Business Rules Approach to Application Development*.: Addison-Wesley.

[14] Marius Muji (2013), "A Model for User Interface Design in Database-Driven Information Systems,"*Scientific Bulletin of the Petru Maior University of Targu Mures*, vol. 10, no. 1, pp. 24-27.

[15] Marius Muji (2015), "Logical Operators for the Data-oriented Design of the User Interfaces," *Procedia Technology*, vol. 19, pp. 810-815.

[16] Tse-Hsu Chen et al. (2014), "Detecting Performance Anti-patterns for Applications Developed using Object-Relational Mapping," in *International Conference on Software Engineering*, Hyderabad, India, pp. 1001-1012.

[17] Alexandre Torres, Renata Galante, and Marcelo Pimenta (June 2014), "ENORM: An Essential Notation for Object-Relational Mapping," *SIGMOD Record*, vol. 43, no. 2, pp. 23-28.

[18] Red Hat. (2011, November) Hibernate. [Online]. http://www.hibernate.org

[19] Apache Software Foundation (2011, November) Apache Cayenne. [Online]. http://cayenne.apache.org/

[20] Atul Adya, Jose A. Blakeley, Sergey Melnik, and S. Muralidhar (2007), "Anatomy of the ADO.NET entity framework," in *ACM SIGMOD International Conference on Management of Data*, Beijing, China, pp. 877-888.

[21] José A. Blakeley, David Campbell, S. Muralidhar, and Anil Nori (December 2006), "The ADO.NET Entity Framework: Making the Conceptual Level Real," *SIGMOD Record*, vol. 35, no. 4, pp. 32-39.

[22] Bert de Brock (1989), *De Grondslagen van Semantische Databases*. The Nederlands: Academic Service.

[23] Bert de Brock (1995), *Foundations of Semantic Databases*.: Prentice Hall.

[24] Lex de Haan and Toon Koppelaars (2007), *Applied Mathemathics for Database Professionals*: Apress.