Active Data Dictionary: a Method and a Tool for Data Model Driven Information System Design

Vadim A. Ermolayev, Natalya G. Keberle

Zaporozhye State University, 66 Zhukovskogo st., 330600, Zaporozhye, Ukraine,
E-mail: eva@zsu.zaporizhzhe.ua

Abstract. Database applications are subjected to numerous changes within their life cycle. Traditional methods and tools for design and implementation of information systems do not possess the means flexible enough to robustly face these changes. The paper presents a method and a tool for the design of flexible and, therefore, easily maintained database applications. The method is based upon a modified IS design procedure assuming data model to be active control part of the application. Data Dictionary discussed is supplied with active control elements and is a program media for data model to serve as information system control engine. Introduced is the concept and the prototype of active data dictionary as well as some extensions to classical relational data model. This theoretical data model enhancements deal with the aspects of extending data model with the property of attribute computability and finding the ways to data model controlled dependencies between the data and the program code of information system.

1. Introduction

Data modeling and database Implementation form the branch being significant for scientific and industrial development. Databases as information sources and data media for the variety of application domains are spread over different bounds playing an important role in achieving progress in all branches of science and technology as well as in human and social activities. Therefore, research aimed to work out formal basis, methods and instrumental means for reliable design and implementation of robust information systems meets high demand. Interested reader can find variety of publications presenting different trends in the field from refinements in classical relational methods of data modeling and database implementation to using object oriented technologies and databases for the purpose. As far as the overview of these publications has been done by V. Ermolayev in [1] we'll just save some space and concentrate on the contributions to the research presented in this very paper.

The aim of this paper is to present the approach based upon the understanding of the leading role of data model and its active influence upon the process of IS design and IS flexibility within the phases of its life cycle. The paper introduces the study aimed to give the answer to the basic question:

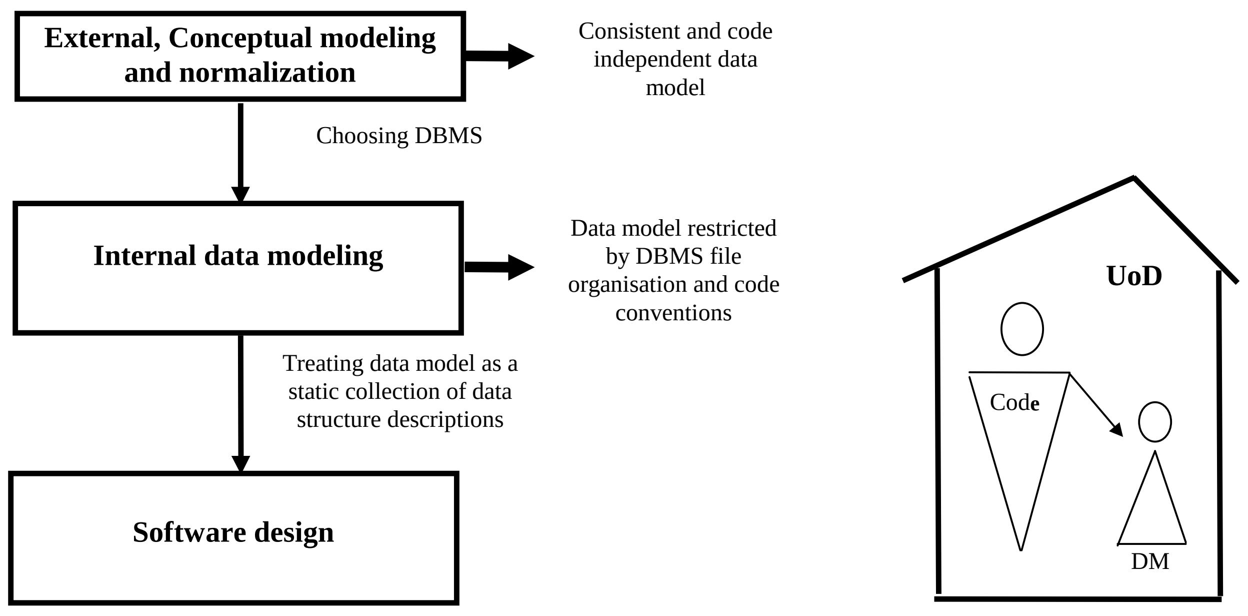
External, Conceptual modeling and normalization
Consistent and code independent data model
Choosing DBMS
Internal data modeling
Data model restricted by DBMS file organisation and code conventions
Treating data model as a static collection of data structure descriptions
Software design
UoD
Code
DM

Fig 1. Traditional Design Procedure

Are the methods of IS design fine enough, or there is the way to improve the quality of the resulting system by changing the method of its design? The answer we have for the moment is the concept of an alternate IS design procedure based upon the understanding of the fact that data model should start playing active role in the process of IS design and functioning. Presented are the concept of this Ideal IS design procedure, Active Data Dictionary (being the program shell for data model in the role of IS control center) and some extensions of relational data model on conceptual level, which have been used as theoretical background for Active Data Dictionary design.

2. Ideal IS Design Procedure

We'll start with the analyses of traditional procedure for information system design before attempting to apply any refinements to it. Traditional design procedure for database applications comprises the following principal steps:

- external, conceptual modeling and normalization
- internal data modeling
- program code design.

Schematically it may be presented as shown on Fig 1.

The process of a database application design at the early stages is initiated and guided by well formalized methods of data model creation. We can refer to wide bibliography starting from Codd [2] and Chen [3]**.** Data model, if properly designed and normalized, looks quite consistent and code independent. The following stages however add more and more restrictions to the model and cause numerous changes in it. Since the 1-st step of the design procedure is completed (refer to Fig 1) the data model looses its leading role and becomes just a set of data structure descriptions heavily influenced by the program code. Low-level data representation produced at the step of internal data modeling is already implementation dependent and therefore is under control of the program code. Even more problems arise while designing the program part of the application. As far as traditional procedure of application code design treat data model as a static collection of data descriptions, data descriptions are duplicated and embedded into the program code.

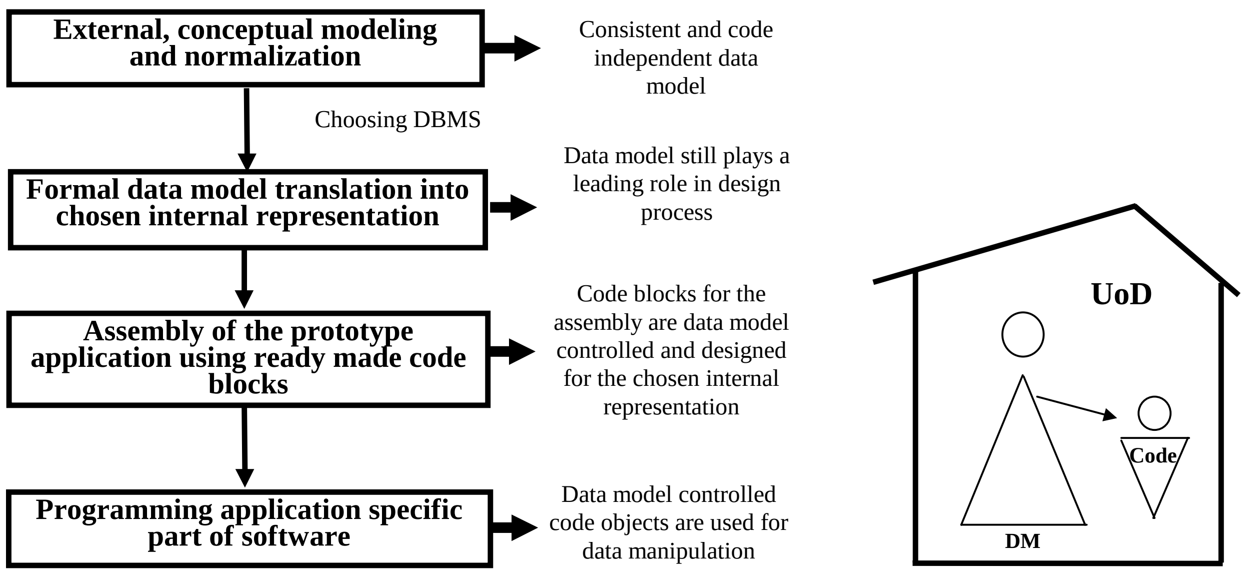
External, conceptual modeling and normalization
Consistent and code independent data model
Choosing DBMS
Formal data model translation into chosen internal representation
Data model still plays a leading role in design process
Assembly of the prototype application using ready made code blocks
Code blocks for the assembly are data model controlled and designed for the chosen internal representation
Programming application specific part of software
Data model controlled code objects are used for data manipulation
UoD
Code
DM

Fig 2. Ideal Design Procedure

Since that data control becomes entirely the function of the program code and therefore the system as a whole becomes heavily constrained: any change of data description leads to numerous modifications of the program part. As a result we are lacking reusability and meet lots of monkey work within maintenance period.

We are deeply convinced that the main problem of the procedure described lies in the understanding of what is the proper “application engine”. Traditionally designed information systems are program controlled.

Thus, major disadvantages of traditional IS design procedure seem to be as follows:

- Traditionally designed database applications are program controlled
- Data model is treated just as an auxiliary data structure description
- Data descriptions are subjected to numerous replication and embedment into the program code

Drawbacks outlined force us to find some conceptual changes in the methods of information system design. Let’s imagine an ideal design procedure and have a strict look at its steps. It is obvious that the task is to find the proper ground for flexible application control. Our strong feeling is that data model is the best to serve as an application engine. Under this assumption the procedure for database application design might be as follows:

- External and conceptual modeling plus normalization phase
- Formal data model translation into the chosen internal representation
- Assembly of a prototype application using ready made data model driven code blocks appropriate to the chosen DBMS (internal representation)
- Programming application specific part of code using data model driven program objects for data retrieval and manipulation.

Conceptual chart of this ideal procedure is given on Fig 2.

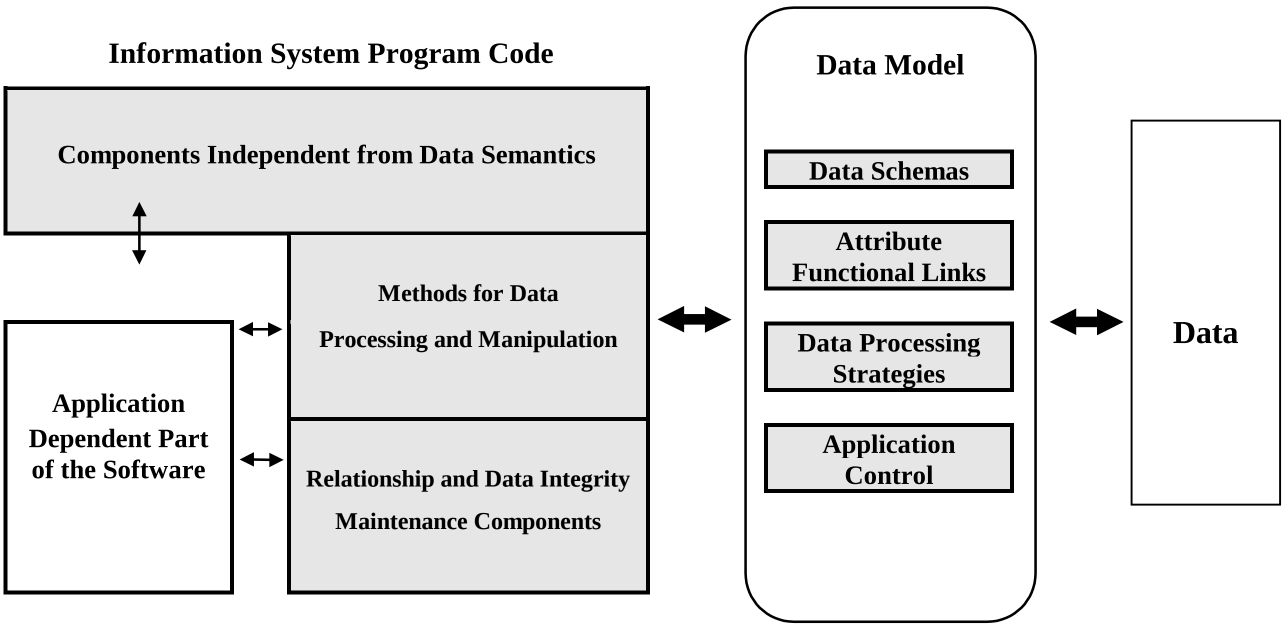
Information System Program Code
Components Independent from Data Semantics
Methods for Data Processing and Manipulation
Application Dependent Part of the Software
Relationship and Data Integrity Maintenance Components
Data Model
Data Schemas
Attribute Functional Links
Data Processing Strategies
Application Control
Data

Fig 3. Principal Structure of an Information System

Fixing data model as the major control unit in the application under design we obviously make this procedure almost free of the drawbacks of the traditional one. Thus the major advantage of this ideal design procedure is:

- Data model keeps its role as application engine from the beginning up to the end of the IS life cycle

We just simply become closer to nature and put all the things to their native order. Housekeeping is a nice analogy for the illustration. Normally men (program code in our notation) think themselves to be senior within the family (the UoD) and women (data model) are to do all the housework under their control - see Fig 1. Actually everything is quite up side down. Who normally knows what and how to do in the house? Who knows where lies what and how to get it? Who makes shopping lists and does the cooking? Probably we know the answer - the Data Model - see Fig 2.

To bring this ideal design procedure closer to practice we are to find the answer to the following conceptual question: *Is it reasonable and possible to use compilative technique for data model controlled code portions and at the same time to overcome the flexibility restrictions? - Some part of program code will anyway be dependent from the application domain.*

We are looking for the answer by way of trying to define the structure of an abstract information system and to figure out the parts being application dependent and those being invariant to application domain. We assume that the principal structure of an abstract information system comprises three major parts: *data*, *data model* and *program code* (refer to Fig 3 for some details). Further analyses of this structure outlines:

- The following parts can be extracted from information system program code: *application dependent portion* implementing the algorithms specific to the application domain *and code objects invariant to specific character of application* and corresponding data model
- *Data model* is an essential part of this structure and its major function is to contain data descriptions as well as data access and manipulation methods descriptions. The structure of data model is invariant to application domain and may be described by means of data model itself

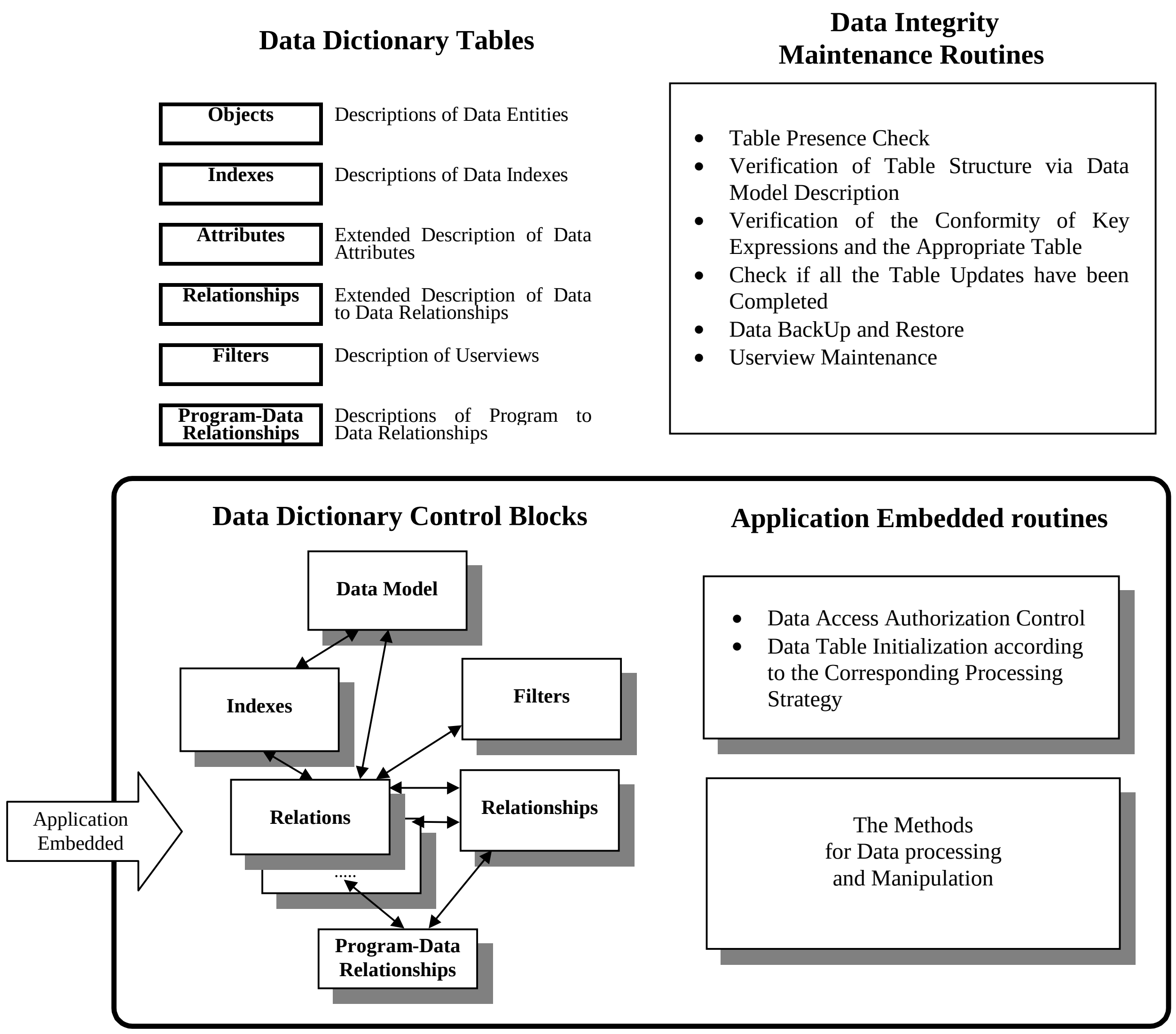
Data Dictionary Tables
Objects
Descriptions of Data Entities
Indexes
Descriptions of Data Indexes
Attributes
Extended Description of Data Attributes
Relationships
Extended Description of Data to Data Relationships
Filters
Description of Userviews
Program-Data Relationships
Descriptions of Program to Data Relationships
Data Integrity Maintenance Routines
Table Presence Check
Verification of Table Structure via Data Model Description
Verification of the Conformity of Key Expressions and the Appropriate Table
Check if all the Table Updates have been Completed
Data BackUp and Restore
Userview Maintenance
Data Dictionary Control Blocks
Data Model
Indexes
Filters
Relations
Relationships
.....
Program-Data Relationships
Application Embedded
Application Embedded routines
Data Access Authorization Control
Data Table Initialization according to the Corresponding Processing Strategy
The Methods for Data processing and Manipulation

Fig 4. Active Data Dictionary Components

- Data model (as an engine) together with application independent part of program code combined together may serve as a compilable program shell controlled by data model and is called *Active Data Dictionary*.

We have developed the concept and the prototype of Active Data Dictionary (ADD) - see [4] for details - to be a tool for solving the problem. Its structure is presented on Fig. 4.

3. Extensions of Relational Data Model

The research we run today is aimed to enhance the properties of ADD methods and technology. The directions under examination are to enhance the descriptive features of relational data model and conceptual modeling technique as well as to add some flexibility to program - data relationships.

Computable Attributes

Information modeling and database application implementation often require to use functional dependencies between different data attributes.

Known results in the field are theoretical and are based upon the functional dependencies and appropriate inference axioms. We utilize the property of "computability" for an attribute and consider it to be a case of a functional dependency.

To design a database for certain application domain, we normally try to present this domain as a collection of objects, then we analyze how these objects interact and describe the relationships. This step of conceptual modeling includes also the description of object properties. If more information about the Universe of Discourse will be present in the conceptual model such a model will be closer to reality and therefore to ideal model. From the other hand, however, the choice of one or another data model for representation of the UoD on logical and internal levels restricts possibilities of data description.

It's known [2] that the notion of functional dependency is one of the key features of relational data model. This notion is actually modeling the fact that one group of object (relationship) properties defines another group of properties of the same object (relationship).

Functional dependencies are used in normalization of data model relationships, as they are allowing to avoid one of the relational data model disadvantages - possible storage anomalies and data redundancy. However, we can use functional dependencies more effectively - for the evaluation of attribute values. Functional dependencies may be described by a formula, a rule, an algorithm, or a program section and therefore produce the possibility to compute the value(s) of one or several properties using the values of another group of properties as arguments.

We treat such functional dependencies as "computable". Actually, the method of computation exists for such a functional dependency. We introduce the notion of computable Functional Dependency to distinguish the dependencies having the method of its computation in specific UoD.

Resembling workaround was described for instance by Maier [5], but for the evaluation of temporary tables while processing queries and their results. Therefore the accent of his results was directed to the analyses of query processing on relationships with C-dependencies. Our approach is aimed to improve relational data model foundations on conceptual level, because the notion of computable attribute might somehow extend its descriptive properties. However the questions are: *What do we get after adding the characteristics of computability to descriptive properties of relational data model? Does this enhanced model remain relational or this new property violates relational model bounds restrictions?*

Formally the problem may be defined as follows: *Verify if it is possible to enhance an attribute of relational data model with the property of computability and stay within the formal bounds of relational data model.* To solve the problem we'll give the definition of computable attribute basing upon the known definitions of relation and functional dependency (see for instance [5]):

Definition 3.1. Relation (Entity):
Relation r having schema R (r(R)) is a finite set of $\{c^1, \ldots, c^n\}$,
where:

c^i	- is the mapping of the set of attribute names to the set of values from corresponding domains.
n	- cardinality of relationship r

c^i is also a tuple: $c^i(A_j) \in D_j$, j=1,...,m, quantity of attributes in schema R.

Definition 3.2. Functional dependency:

If r(R) - relation, $X \subseteq R$, $Y \subseteq R$, then

X functionally depends from Y if and only if for any distinct value of a tuple on Y exists unique value of the tuple on X.

Definition 3.3. Computable attribute:

Let schema $R = [I, X \cup Y \cup A^*, D(X) \cup D(Y) \cup D(A^*), T]$

Attribute A* is computable on X if exists some function $f: D(X) \rightarrow D(A^*)$

that: $c^i(A^*) = f(c^i(X))$, $i=1,...,n$

f	- function semantically defined by application domain.

After the notion of computable attribute is defined we pass to the analyses of the properties of algebra B' produced from relational algebra B and enhanced by the set of functions for attribute value computation. Algebra B' is defined as follows:

Definition 3.4. Enhanced algebra:

B' is the 8-nested tuple $\{U, D, dom, R, d, \Theta, O, F\}$,

where:

U	- set of attributes - universum
D	- set of domains
$dom: U \rightarrow D$	- function, transforming a set of attributes into a set of domains
$R = \{R_1,...,R_p\}$	- set of different relational schemas, $R_i \in U$
d	- set of relations with schemas from R
Θ	- set of binary relations upon domains from D, including at least $=$ and $\neq$ for any domain
O	- set of relational operations: projection, selection, join, division, intersection, conjunction, product, addition
F	- set of functions: for any $f \in F$ exists $D' \subset D$, $D'' \in D$ and $D'' \notin D'$ and $f: D' \rightarrow D''$

Proved [6] is that all the properties of both boolean and special relational operations are true for algebra B'. Moreover, if we suppose the elements of F to be bijective, some relational operations gain extra properties. Thus, algebra B' is closed and it is as expressive as relational algebra is: it is relational and has some extra properties. Actually the basic practical result is that we can treat the data model based upon algebra B' as relational and apply all the procedures known for relational data model to this enhanced model.

In addition we've checked out the applicability of the Inference Axioms to computable F-dependencies as a way to produce all computable F-dependencies upon the database schema. Proved is that independent system of Axioms (Armstrong's Axioms [7]) is applicable to the set of F-dependencies including computable ones. Both procedures of decomposition and synthesis upon the relations comprising computable attributes are therefore applicable. Additional requirement of function f bijectivity helps to avoid the appearance of the so called "hidden" transitive dependencies.

Practical benefits gained by the usage of computable attributes and appropriate data modeling technique seem to be as follows:

- Using computable attributes as primary keys will help to formalize and simplify the algorithms and procedures of cascade update and deletion even in cases of indirect relationship maintenance

- Computable attributes may be also used for representation of some integral value of an attribute subset on a tuple
- The usage of computable attributes is therefore the way of modeling multiply relationships between the subsets of attributes of appropriate entities in data model

Program to Data Relationships

Another relational model extension is handling program - data dependencies by means of Active Data Dictionary model.

In practice while implementing an information system we often use the values of some data attributes as switches in the program code. We qualify such attributes as Control Attributes. The value of a control attribute once been embedded into the program code should remain unchanged in database (or changed synchroneously together with the corresponding portion of the program code). This fenomenon straightly causes the so called “hidden” redundancy in the data collection of the information system. To say more - it causes some significant flexibility restrictions since any modification of such a Control Attribute requires the modification in the program and vice versa.

The problem may be defined as follows: *Work out formal methods together with corresponding algorithms to enshure automatic update of the Control Attribute Shadow in the program code within one transaction together with corresponding value in the database.*

We distinguish two types of program - data dependencies (relationships): a value relationship and an identifier relationship.

Value relationships represent the phenomenon of using the values of some attributes for program code control. The following example may be proposed for the illustration. Let's consider an information system for publications retrieval from the Conference "XYZ" Proceedings Web. The fragment of the data model for this application may contain the elements, presented on Fig. 5.

An example of the procedure for a reader request processing may be described as follows:
1. Request reader's Name and Status (a Conference SIG member, a Student,…, a Stranger-on-the-Web)
2. Redisplay Volume contents according to the Reader's Status
3. Process Reader's request of the specific Article:

The code for point 3 of our generic procedure will definitely contain a sort of CASE structure with different code branches for each Processing Mode defined in "Processing Modes" entity (see Fig. 6).

Let's now examine the case of **Processing_Modes.Code** attribute value modification. If we simply change 1 to 9 and 9 to 1 the effect will be quite confusing: SIGXYZ members will gain a nice opportunity to pay full price for an article and at the same time it becomes free for almost anybody exploring the Web. It is definitely clear that this simple example belongs to a wide set of cases, which lead IS administrator to maintain both data and program code or just fix the values of such an attribute. **Processing_Modes.Code** attribute is the example of control attribute of value relationship type.

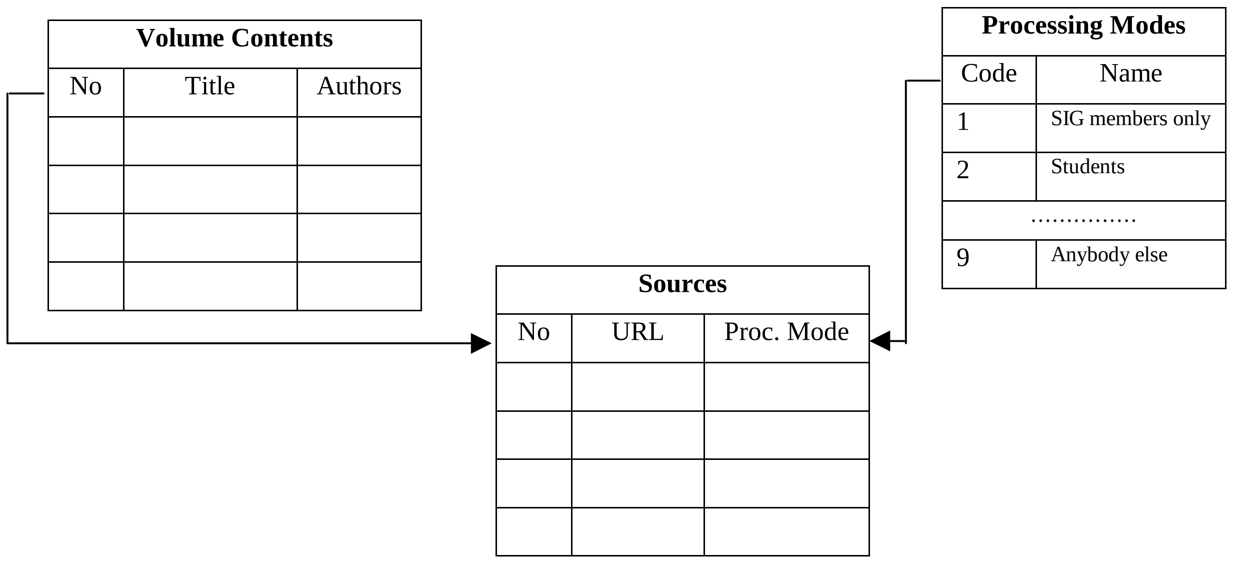

Fig. 5. An example of data structure for Conference "XYZ" publications retrieval.

Obviously, the solution for value relationship control attribute maintenance is:

- 1-st - to replace the constants similar to control attribute values in program code by global variables getting appropriate values during application initialization
- 2-nd - to apply a sort of program shell to synchroneously modify both Control Attribute and appropriate variable values within one transaction.

The formal model extension for value relationship control attributes is as follows. Let $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_p\}$ be a set of variables shadowed by the values of some control attributes. Let control attributes of value relationship type is determined by:

```
DO CASE
   CASE Readers_Status = 1    /* SIGXYZ member - free delivery
        CALL SIGXYZ_Delivery(Article No, …)

   CASE Readers_Status = 2    /* a student - 60% discount
        CALL Discount_Delivery(Article No, …)

                …………..

   CASE Readers_Status = 9    /* Mr.Nobody - full-price delivery
        CALL FullPrice_Delivery(Article No, …)

END CASE
```

Fig. 6. Case structure for processing different groups of readers

Variable Identifier	Entity Name	Control Attribute Name	Value
$\mathbf{v}_1$			. . . . .
$\mathbf{v}_2$			. . . . .
. . . . .			
$\mathbf{v}_i$	$\mathbf{r}^i$	$\mathbf{a}_j^i$	$\left(\mathbf{c}_j^k\right)_{\mathbf{a}_j^i}$
. . . . .			
$\mathbf{v}_p$			. . . . .

Fig. 7. The structure of value relationship control attribute maintenance table

Definition 3.5. Value Relationship.

Attribute $\mathbf{a}_j^i$ of the relation $\mathbf{r}^i$ with the schema $\mathbf{R}^i$ is a control attribute of type 1 (value relationship) if there is a tuple $\mathbf{C}^k = \left\{\mathbf{c}_1^k,\ldots,\mathbf{c}_j^k,\ldots\mathbf{c}_m^k\right\}$ belonging to $\mathbf{r}^i$ and $\left(\mathbf{c}_j^k\right)_{\mathbf{a}_j^i} = \mathbf{v}_i$,

where:

$\left(\mathbf{c}_j^k\right)_{\mathbf{a}_j^i}$ - the value of attribute $\mathbf{a}_j^i$ on a tuple $\mathbf{C}^k$, $\mathbf{v}_i \in \mathbf{V}, \mathbf{i} = \mathbf{1},\ldots,\mathbf{p}$,

$\mathbf{m}$ - cardinality of $\mathbf{C}^k$.

Active Data Dictionary application embedded components can serve as the program shell for Control Attribute modification. To supply ADD with information about the relationships between Control Attribute values and corresponding global variables in the program code we use the control table with the structure presented on Fig. 7.

Thus the solution of the problem of value relationship control attribute modification becomes trivial since we have brought into line the variable name and corresponding control attribute instance - refer to Definition 3.5. and Fig. 7. Program code for Processing Modes should obviously be changed as shown on Fig. 8.

```
DO CASE
   CASE Readers_Status = V1    /* SIGXYZ member - free delivery
        CALL SIGXYZ_Delivery(Article No, …)

   CASE Readers_Status = V2    /* a student - 60% discount
        CALL Discount_Delivery(Article No, …)

               …………..

   CASE Readers_Status = V9   /* Mr.Nobody - full-price delivery
        CALL FullPrice_Delivery(Article No, …)

END CASE
```

Fig. 8. Modified Case structure for processing different groups of readers

It is still not clear how to establish the correspondence between the variables V1, V2, …,V9 and appropriate processing modes - the tuples in "**Processing Modes"** entity. The mechanism for setting up this dependency is based upon control attributes of identifier relationship type. In our example the control attribute **Processing_Modes.Name** (see Fig. 5) represents this relationship since we have somehow established the mapping of it's values to the set of variable names "V1", "V2", …, "V9". By setting up this relationship we fix the fact that variable with identifier "V1" is used for processing *SIGXYZ members*, variable with identifier "V2" is used for processing *Students* and, finally, variable with identifier "V9" is used for processing *all those who have no discounts*.

The model for control attributes of identifier relationship type is formally defined as follows:

Definition 3.6. Identifier Relationship.
Attribute $\mathbf{a}_j^i$ of the relation $\mathbf{r}^i$ with the schema $\mathbf{R}^i$ is a control attribute of identifier relationship type if there is a tuple $\mathbf{C}^k$ belonging to $\mathbf{r}^i$ and $\exists\ \mathbf{f}, \mathbf{f}\left(\left(\mathbf{c}_j^k\right)_{\mathbf{a}_j^i}\right) = \mathbf{id}(\mathbf{v}_i)$
where:

$\mathbf{id}(\mathbf{v}_i)$	- identifier of variable $\mathbf{v}_i$, $\mathbf{v}_i \in \mathbf{V}, \mathbf{i} = \mathbf{1,...,p}$,
$\mathbf{f}$	- a function semantically defined by the application.

In our example function f should establish bijective correspondence between active domain of the control attribute **Processing_Modes.Name** and the set of global variables {V1, V2, …, V9}.

The solution of the problem of control attribute of identifier relationship type modification will be achieved and, therefore, integrity restrictions will be satisfied in case if the algorithm for control attribute modification will count on the following restriction for the new value of the control attribute:

$$\forall\left(\mathbf{c}_j^k\right)'_{\mathbf{a}_j^i} \exists \mathbf{f}: \mathbf{f}\left(\left(\mathbf{c}_j^k\right)'_{\mathbf{a}_j^i}\right) = \mathbf{id}(\mathbf{v}_i)$$

where:

$\left(\mathbf{c}_j^k\right)'_{\mathbf{a}_j^i}$ - new value of attribute $\mathbf{a}_j^i$ on a tuple $\mathbf{C}^k$,

$\mathrm{id}(\mathrm{v}_m)$- identifier of variable v_m, $\mathrm{v}_m \in \mathrm{V}, \mathrm{i} = 1,...,\mathrm{p}$, $\mathrm{m} \neq \mathrm{i}$

We are building functions f for control attributes of identifier relationship type in the form of relational entity, which serves as the control table for Active Data Dictionary application embedded components. The proposed structure of this entity is presented by Fig. 9.

We expect that integrity restrictions for identifier relationship type control attribute modification will be fulfilled if we set up a relationship of cardinality $\mathbf{1{:}m}$ between attribute Value of the entity representing function $\mathbf{f}$ and attribute $\mathbf{a}_j^i$ of entity $\mathbf{r}^i$.

The algorithms for program - data dependencies maintenance still wait for implementation and integral testing within ADD prototype for various physical platforms.

Variable Identifier	Entity Name	Control Attribute Name	Value
$\mathrm{id}(v_1)$			. . . . .
$\mathrm{id}(v_2)$			. . . . .
. . . . .			
$\mathbf{id}(\mathbf{v}_i)$	$\mathbf{r}^i$	$\mathbf{a}_j^i$	$(\mathbf{c}_j^k)_{\mathbf{a}_j^i}$
. . . . .			
$\mathrm{id}(v_p)$			. . . . .

Fig. 9. The structure of the control table for handling identifier relationship type control attributes.

Attribute Name Dependencies

One more type of program-to-data dependencies should be mentioned to complete the code flexibility analyses. If we examine an information system code the major part of its links to data and data model is provided via the names of data attributes. Any program section implementing data navigation and maintenance and, moreover, each DML statement contains the names of data attributes under operation. This fact hardly influences the flexibility of program code. In order to adequately react to the data model changes (changing attribute names and/or data types, deleting attributes, adding new attributes) we are to fulfill numerous updates to the appropriate program code portions.
The problem of automated handling the program to data dependencies of this type is somehow inversed to the one of identifier relationship control attributes and is still waiting for the solution.

4. Conclusions

In this paper we have demonstrated that a small role shift in the procedure of IS design might provide drastical benefits in resulting application properties of flexibility and adaptivity to data model changes and, therefore, user demand. Presented is an ideal information system design methodology. Its main difference from the traditional one is the abovementioned role shift in understanding of which part of IS should play the leading role in the resulting product. Our deep confirmation is that the best candidate as application engine in easily maintainable information system is data model.

The primary aim of the enhancements to relational data model presented above is not to improve the model itself, but to uncover some hidden properties to which serious attention of the research community has not been paid yet. These obvious improvements: computable attributes, the models and procedures for handling program-to-data relationships have, anyway, evident practical prospects.

The practical motive and practical implementation in ADD presented just circle the workaround and lighten the way to wide practical usage of the method of flexible IS design, leaving the room for future development. Possible contributions to the enhancement of data model based IS design methods and appropriate ADD features are on the way of:

- Enhancing ADD models to be able to maintain semantically reach models, ERC+ [8] for instance

- Analysing the possibilities to maintain the models of spatio-temporal domain
- Enhancing IS design methods and corresponding ADD models to data federation and data warehousing level
- Developing visual tools for ADD metadata maintenance

5. Acknowledgements

The authors would like to express special gratitude to prof. Gunter Saake from Otto-von-Guericke University, Magdeburg, Germany, who kindly provided the possibility to discuss the results presented here at his seminarium. The authors are also thankful to prof. Stefano Spaccapietra, Laboratory of Databases, EPFL, Lausanne, Switzerland, for fruitful discussion on the perspectives of further development of our approaches and models.

6. References

[1] V. A. Ermolayev Object Oriented Dynamic Data Modeling and Active Data Dictionaries - Some Crosspoints . - "Journal of Metrology and Certification", ZSU, Vol 1, No 1, (Jan-June 1997)

[2] Codd E. F. A Relational Model of Data for Large Shared Data Banks. Comm. ACM, V 13, No 6, 1970, pp. 377-387.

[3] P. P. Chen. The Entity-Relationship Model: Towards a Unified View of Data, ACM TODS, Vol. 1, No. 1, 1976.

[4] V.A. Ermolayev, V. V. Shapar. Using Data Dictionary System for Data Model Design and Verification. In Theses of the Annual Scientific Conference of Zaporozhye State University, Zaporozhye State University, Zaporozhye, Ukraine, 1996, Vol 6, Part 1, p 84-85. - (in russian)

[5] Mayer D. The Theory of Relational Databases. Computer Science Press, 1983, 608 p.

[6] V.A. Ermolayev, N. G. Keberle. On Possibilities to Enhance Relational Attributes by the Property of Computability. - to appear in "Journal of Metrology and Certification", Vol. 2, No 1, (Jan.- Jun. 1998) - (in russian).

[7] Armstrong W. W. Dependency Structures of Database Relationships. In: Proc. Of IFIP Cong., Geneva, Switzerland, 1983, pp. 580-583.

[8] C. Parent, H. Rolin, K. Yetongnon, S. Spaccapietra. An ER Calculus for the Entity Relationship Complex Model, Proc. 8th International Conference on Entity - Relationship Approach, Oct. 18-20, 1989