Модел. и анализ информ. систем. Т. **22**, № **2** (2015) 259–277

UDC 519.682;681.324.06

Completeness of the Dynamics of the Attributes Values of Data in the Database DIM

Petrov A. N., Roublev V. S.

P. G. Demidov Yaroslavl State University, Sovetskaya str., 14, Yaroslavl, 150000, Russia

e-mail: axel_petroff@mail.ru, roublev@mail.ru

received March 20, 2015

Keywords: object DBMS, dynamics data, description completeness

This paper is devoted to justifying the possibility of DBMS DIM usage and its interaction mechanism as an algorithmically complete implementation of an object-dynamic model. An extension for a static *OD*-model by including sets of algorithmic procedures which modify values of object attributes and also create, remove and modify objects themselves is considered. To ensure the possibility of modifying DIM DB data in a way equivalent to *OD*-model modifications, interaction and history relations between DIM objects are considered. To minimize the dependence from concrete language constructions, which describe *OD*-model algorithmic procedures, the reduction to the universal form – Turing machine is performed. A way to create a Turing machine equivalent to *OD.MT* in terms of DIM, where a special set of PL/ODQL procedures is used as a control unit and a functional table is proposed. Later, a mechanism to form a memory tape of such *DIM.MT* by encoding information about DIM object, and their subsequent decoding back to DIM objects is described. The process of work of such a machine is modelled by using an endless cycle of executing some PL/ODQL procedures of reading and writing objects from / to the memory tape. Basing on the earlier proved theorem about the static completeness of data representation in DIM, at the end of the paper the proof on the completeness of representation of the Objects attributes values dynamics is considered.

The article is published in the author's wording.

Introduction

Due to the fact that existing DBMS technologies are not technically or conceptually perfect ([1]–[6]), a tryout to innovate a new objective approach to DB engineering was taken by our research group. During a number of years was made a lot of work on this concept, which includes ability not only to modify objects data, but also to change object types and relations between objects (i.e. dynamic DB scheme engineering). In addition to this ability were identified 6 base relations between objects: *inheritance*, *inclusion*, *inner inheritance*, *inner inclusion*, *history*, *interaction* which can completely

describe all kinds of sophisticated relations between objects. This DB concept was called a Dynamic Informational Model or DIM. Also was introduced a specialized objective query language ODQL ([7]–[10]) and some other programming languages that fit all the needs to manipulate data in DIM.

The introduction on the new DB technology requires strong grounding on data description completeness and data dynamics completeness. The last one was connected with exact domain description, or rather with mathematic model creation, which was called OD-model.

The previous paper [11] was devoted to proving static completeness of data description in DBMS DIM. Wherein the *OD*-model was not extended to describe algorithmic procedures related to data changing dynamics. Such a domain description was quite enough to be able to build a projection from *OD*-model to DIM for any static data slice (at arbitrary moment of time), which can adequately reflect properties of the objects from *OD*-model and links between those objects.

This paper acts as a continuation of [11] and it is devoted the proof of DBMS DIM objects attribute dynamics completeness. According to this, now it is possible to split the proof into 3 steps:

first, to refine *OD*-model description in the field of algorithmic procedures designed to modify data;

second, to describe interaction relations and history relations, which are the fundamental data management mechanism in DBMS DIM and temporal links mechanism for data in DBMS DIM;

third, it is necessary to build a projection between *OD*-model algorithmic procedure into DIM interaction, which will conserve all the properties for projection between two static data splices: before algorithmic procedure call and after this call was finished.

1. OD-model extension

Previously, in [11] *OD*-model was described as

$$(O, A, \overline{A}(o), V(o), L_p, L_o, L_f, \overline{A}_{L_f}(o_l^j), V_{L_f}(o_l^j), F, T),$$

where

O – a finite set of objects,

$A = \bigcup_o A_o$ – a finite set of objects attributes with their types (each element is a pair (a, V^a) – attribute, type of attribute),

$\overline{A}(o)$ – a function to retrieve cortege of object's o attributes,

$V(o)$ – a function to retrieve cortege of object o attributes values (the order of the values matches to the order of object attributes in the cortege $\overline{A}(o)$),

$L_p = \bigcup_{j \in L_p} \{l_j^p = \{o, o1\}\}$ – a set of simple links between objects,

L_o – a set of objects-links ($O \cap L_o = \emptyset$),

$L_f = \bigcup_{j \in L_f} \{(l_j^f, o_l^j \in L_o)\}$ – a set of functional links between objects,

$\overline{A}_{L_f}(o_l^j)$ – a function to retrieve a cortege attributes of object-link o_l^j of functional links L_f,

$V_{L_f}(o_l^j)$ – a function to retrieve a cortege of values of object-link o_l^j' attributes of functional links L_f

F – a finite set of algorithmic procedures, intended to modify objects and their attributes,

T – a descrete timeline.

Some of attributes of every object stay constant during the object «lifetime». But some other attributes values vary in time under the dynamic determined rules for objects interaction. Such an attribute modification does not depend on time moment of modification, but it depends on values of other attributes of the object, and, possibly, depends on other objects linked with the object. Let $F^{'}$ be a set of algorithmic procedures, while execution any of them $f \in F^{'}$ at a moment of time $t \in T$ on a set of corteges of values $\{V_{o_j}(t) \mid j \in \overline{1, m(t)},\ o_j \in O,\ t \in [t_{o_j}^n, t_{o_j}^k]\}$ of all existing values ($m(t)$ – is an amount of such objects), which are significant for interaction procedure execution, and a set L of links between these objects are values of such corteges an the next moment of time:

$$< V_{o_1}(t+1), V_{o_2}(t+1), \ldots, V_{o_{m(t)}}(t+1) > \ = f(V_{o_1}(t), V_{o_2}(t), \ldots, V_{o_{m(t)}}(t), L),$$
$$t \in [t_{o_j}^n, t_{o_j}^k],\ j \in \overline{1, m(t)}.$$

Procedures from set $F^{'}$ define dynamics of modifying model object attributes.

Apart from the set $F^{'}$, introduce a set of algorithmic procedures $F^{''}$ such that while the executing procedure $g \in F^{''}$ at a moment of time t some of existing objects cease to exist («die»): $t_o^k = t$, but some of them come into existence («born») at the next moment of time $t+1$: $t_o^n = t+1$.

$$< o_{new1}(t+1), \ldots, o_{new_{m(t+1)}}(t+1) > \ = g(o_{old1}(t), \ldots, o_{old_{n(t)}}(t), L),$$
$$t_{o_{old_j}}^k = t,\ j \in \overline{1,\ n(t)};\ t_{o_{new_j}}^n = t+1,\ j \in \overline{1,\ m(t+1)},$$

where in replace for old objects o_{old_j} ($j \in \overline{1,\ n(t)}$) «born» new objects o_{new_j} ($j \in \overline{1,\ m(t+1)}$).

Procedures $F^{''}$ define the dynamics for OD-model objects modifications, and the union $F = F^{'} \cup F^{''}$ defines the full dynamics for the model.

This paper is aimed at considerations of the dynamics for objects attributes modifications.

According to this clarification, it is possible to describe extended OD-model as:

$$(O, A, \overline{A}(o), V(o), L_p, L_o, L_f, \overline{A}_{L_f}(o_l^j), V_{L_f}(o_l^j), F^{'}, F^{''}, T),$$

where

O – a finite set of objects,

$A = \bigcup_o A_o$ – a finite set of objects attributes in conjunction with types of those attributes (each element is a pair (a, V^a) – attribute, type of attribute),

$\overline{A}(o)$ – a function to retrieve cortege of attributes of the object o,

$V(o)$ – a function to retrieve a cortege of values of attributes of objects (order of the values matches to the order of object attributes in a cortege $\overline{A}(o)$),

$L_p = \bigcup_{j \in L_p} \{l_j^p = \{o, o1\}\}$ – a set of simple links between objects,

L_o – a set of objects-links ($O \cap L_o = \emptyset$),

$L_f = \bigcup_{j \in L_f} \{(l_j^f, o_l^j \in L_o)\}$ – a set of functional links between objects,

$\overline{A}_{L_f}(o_l^j)$ – a function to retrieve cortege of attributes of the functional links' L_f object o_l^j,

$V_{L_f}(o_l^j)$ – a function to retrieve cortege of values of attributes of the functional links' L_f object o_l^j,

F' – a finite set of algorithmic procedures for modifying objects' attributes values,

F'' – a finite set of algorithmic procedures for modifying objects,

T – a descrete timeline.

2. DIM extensions (interaction relation and history relation)

2.1. Interaction relation

To introduce such a type of relation define a special class c_h of *interactions*, which describes methods for interactions for objects of classes *From*, *Where* and *What*.

The interaction relation is defined on the set

$$B \subseteq \{(c_f, c_t, c_w, o_h) \mid c_f \in C,\ c_t \in C,\ c_w \in C,\ o_h \in c_h\}$$

(B – from *behavior*) of fours of 3 classes: *From* (c_f – *from*), *Where* (c_t – *to*), *What* (c_w – *what*) and an object of interaction class c_h *How* (o_h – *how*).

Projections B_f, B_t, B_w of this set respectively define sets of classes which take part in roles *From*, *Where* or *What*, and $B_h \equiv c_h$ is a set of interaction objects, or, that is the same – an interaction class. This class can act as *How* class: it is possible to view, modify or remove methods of interaction.

The interaction class structure includes such attributes as *interaction name, interaction script (or its procedure), interaction duration, interaction mode (usermode or automatic), interaction resources modification* and *interaction conditions*. The last attributes describes conditions imposed on resources needed by interaction, so when these conditions are satisfied – interaction can be executed in automatic mode. This fact allows to introduce *events*, which can be fired during DIM DBMS worktime in *automatic mode*. This mode can be described in a few key points: when interaction executes, it can modify resource conditions of other interaction, so it can cause execution of other interactions. Thus, it is possible to introduce *interaction graph*, where nodes are interactions, and edges are resource conditions connecting interactions with each other. Such an approach gives serious benefits: after introducing timeline scale and applying automatic mode of interactions execution, it is possible to simulate data modification forecast that gives a user understanding of the feasibility of applying business-solutions. This mechanism brings *temporal* data management features for future into DIM.

Interaction procedure (interaction script) – defines *how* the interaction execution process will be performed ([17],[18]). For describing logic for working with data inside

of DIM interaction, specialized, DIM oriented programming languages were designed: **PL/ODQL** and **DIM-FL**. First of them – **PL/ODQL** is the main programming tool in DIM ([12], [13]). PL/ODQL in most of its aspects inherits idiosyncrasies and constructions of well-known PL/SQL language [16] used in Oracle DBMS. Just as in the case with SQL extension, PL/ODQL application case is grouped into separate modules (in case of PL/SQL such modules are usually called packages). The language has a wide area of features for working with ODQL-queries, sets of DIM objects, and also it gives the possibility to use mature exceptions mechanism. In addition to general purpose the programming language PL/ODQL, a specialized mathematical formula oriented programming language – **DIM-FL** (Formula Language) was designed ([14],[15]). Due to its nature, DIM-FL language must give developers the ability to develop algorithms relying on mathematical calculations techniques, and its syntax must be very simple, concentrated on mathematical constructions.

In some parts the execution of this procedure can be interactive (for example: defining of objects *From*, *Where* and *What*), but in other parts, procedures written in DIM-FL or PL/ODQL can be launched (to launch them, special built-in DIM-FL and PL/ODQL interpreter must be executed with a parameter set in interactive mode of passed from *Interaction procedure* body). Beside this fact, interaction execution can be interrupted for some reasons, and, to be able to find the reason, verbose log is written into *Interaction log* during the execution.

Certain parts of interaction and the order of their execution are described using DIM-Script:

- determination of objects *What*, *From* and *Where*;
- sequential execution of interaction subparts (procedures and functions) written in PL/ODQL and mathematical formulas written in DIM-FL language, and calls of another interactions.

The first stage in interaction execution is determination of objects *What*, *From*, *Where*, which take part in the interaction execution process. To mark which objects are needed, the user at design time of interaction chooses objects and classes using a special interactive wizard (interaction and ODQL-queries generation master). At runtime of interaction the executor finds out *What*, *From*, *Where* objects by performing bound ODQL-queries. Using DIM-Script a syntax, a developer can use keywords **WHAT**, **FROM** and **TO** to mark objects from classes *What*, *From*, *To*, respectively:

<keyword> “{” <ODQL-query> [, <ODQL-query>] . . .“}”

<keyword> ::= **WHAT** | **FROM** | **TO**.

Constructions with queries, which can fetch objects that take part in the interaction execution are located at the very beginning of the interaction body before any other script constructions. Descriptions of all the objects **WHAT**, **FROM** and **TO** are not obligatory, because not every interaction should be bound with all the objects *What*, *From*, *Where*. Some objects are settled by default values at runtime.

Also some special calling statements were introduced in DIM-Script:

- to call PL/ODQL procedures **EXECUTE** statement should be used: **EXECUTE** <name of PL/ODQL Procedure >;

- to call sophisticated calculations described in DIM-FL formulas **CALCULATE** statement should be used: **CALCULATE** <name of DIM-FL Formula >;
- to call another interaction, **CALL** should be used: **CALL** <name of Interaction>.

2.2. History relation

The history relation introduced in DIM, is used to represent the temporal aspect of data management. For all the objects, the history of which is necessary to store, 2 additional parameters: *moment of birth* t_o^b and *moment of death* t_o^d of objects are introduced. After the object death it ceases to exist, but to replace it (in the next moment of time) new objects can appear, which are called *successors*

$$o_1^{sc}, o_2^{sc}, \dots \ (t_{o_1^{sc}}^b = t_{o_2^{sc}}^b = \dots = t_o^d + 1)$$

of the initial object, and for a new object birth other objects with preceded by *moment of death* $t_d = t_b - 1$, these objects are called *predecessors*

$$o_1^{pr}, o_2^{pr}, \dots \ (t_{o_1^{pr}}^d = t_{o_2^{pr}}^d = \dots = t_o^b - 1).$$

To be exact, for an objects o, *successor* object – object o_{sc}, which replaced the initial one in the moment *next to the moment of death* $t_o^d + 1 = t_{o_{sc}}^b$, and, possibly, replaced another objects. *Predecessor* object – object o_{pr}, which was replaced by a current object o at the moment of time *previous to the moment of birth* of initial object $t_o^b - 1 = t_{o_{pr}}^d$ (also, possibly, together with some other objects). In the set of all the objects of model O is introduced, an antisymmetric transitive binary relation, which is called *relation of history of objects*

$$H = \{(o_{pr}, o_{sc}) | o_{pr}, o_{sc} \in O, t_{o_{sc}}^b - t_{o_{pr}}^d = 1\}.$$

The relation defines an object-predecessor o_{pr} for object o_{sc}, object-successor o_{sc} for object o_{pr} and a moment of modification $t = t_{o_{pr}}^d = t_{o_{sc}}^b - 1$ Using this relation, it is possible to find out the *set of its successors* $\{o_1^{sc}, o_2^{sc}, \dots\}$ $(t_{o_1^{sc}}^b = t_{o_2^{sc}}^b = \dots = t_o^d + 1)$ at the *moment of time just before its death* and a *set of its predecessors* at the *moment of time just after its birth* $\{o_1^{pr}, o_2^{pr}, \dots\}$ $(t_{o_1^{pr}}^d = t_{o_2^{pr}}^d = \dots = t_o^b - 1)$.

To describe the dynamics of classes modification, some analogous constructions for classes are introduced: *moment of birth of class*, *moment of death of class*, *relation of history of classes*, *classes-predecessors* and *classes-successors*. If class modification is performed, all the objects of class undergo changes, and links between classes and objects can be also modified. To make modifications of classes (or objects), some special procedures are introduced. If some class c_{old} is replaced by a class c_{new}, then all the object belonging to class c_{old} cease to exist, and to replace them new objects belonging to class c_{new} appear. Wherein addition/removing of class parameters and addition/removing of the links with other classes are performed in a way described later:

1) the class definition is copied into a new class c_{new} with a new class identifier (thus, the class c_{old} with an old identifier becomes a predecessor, but the class c_{new} becomes a successor in a relation of the classes history);

2) the data from objects belonging to the old class c_{old} are copied into the data of objects of the new class, every object gets its own new identifier (thus, every object with the old identifier becomes the predecessor of the object with new identifier in the relation of classes history);

3) attributes belonging to the new class c_{new} are added as additional attributes;

4) for all the objects of new class values of those added attributes are defined;

5) affiliation to a group of each added attribute of the new class c_{new} is changed according to the need;

6) unneeded attributes of the new class c_{new} are removed, and also integrity constraints for this class are checked;

7) new links with other classes are added to the new class c_{new} and also for those objects of those other classes that need to be linked with objects of the new class, some links are determined (such objects obtain a new identifier and new links with objects with the old identifier are settled with the help of the relation of objects history);

8) unneeded links of the new class c_{new} with other classes and links between objects of these classes are removed (as for such objects of other classes, data are copied into objects with new identifiers and links between old and new objects are created using relation of objects history); then integrity constraints for all the objects for which old class c_{old} had a parent object are checked.

Similar to these constructions, dynamics constructions for attributes were introduced. For example, it is possible to modify datatype of an attribute and this modification will be kept as a relation of attributes history item.

To describe the dynamics of interaction it is also necessary to introduce similar constructions: *moment of interaction birth*, *moment of interaction death*, *relation of history of interactions*, *interactions-predecessors* and *interactions-successors*, and also *period of application of interactions*. Usually, only topical interactions are in use. But sometimes, period of application of interaction may be longer than its «lifetime», thus it can be applied to objects which «lifetime» period is outside the interaction «lifetime» period, although that period should be inside the period of interaction application.

2.3. Object type of DIM

The type of an object in DIM is defined by a set of class attributes and a set of interactions, for each of them the class of the object or its parent class acts as 1 of 4 *roles* in interaction. A set of type attributes is defined by:

1) *class parameters* of the object;

2) *parameters of parent classes* for the class of object, if they exists;

3) *links* with any other classes, which are linked by *inclusion* with the class of object or with its parent classes.

The type of the object is defined by its class only in simplest cases, as we can see from the definition.

3. Dynamic completeness of OD-models' descriptions in regard to DIM

In the previous paper [11] a projection G for transformation arbitrary OD-model into DIM *classes scheme* was constructed. Based on this projection was proved theorem *on completeness of static data representation* in DIM.

Now, it's considered to continue investigation on this projection to extend into ability to transform algorithms from set F of OD-model into DIM interactions. This transformation should keep in sync objects from both models, their attributes and attributes values before execution of any algorithm $f \in F$ (or respective interaction $G(f) \in B$) and after its execution.

To minimize $G(f)$ transformation's dependence from f and $G(f)$ descriptions characteristics, it's a good approach to use universal well-known algorithm description in the form of *Turing machine* (MT). To be able to use this approach, Turing machines for OD-model ($OD.MT$) and for DIM-model ($DIM.MT$) are investigated.

3.1. OD-model's Turing machine's description

Build OD-model of arbitrary Turing machine (MT). To accomplish this task it is necessary to represent *memory tape* and MT *control unit* (CU) as objects, and also describe an interaction between them using set of OD-model functions, which will comply with modification of model's objects at each step of MT work.

Let outer MT alphabet B consist of m symbols and some empty symbol $b_0 : B = \{b_0,\ b_1, \ldots, b_m\}$, and let inner MT alphabet Q consist of n states, and a stop state $q_0 : Q = \{q_0,\ q_1, \ldots, q_n\}$. At time of MT start, i.e. at the moment of time $t = 0$, control unit is in an initial state q_1.

The MT memory tape is infinite in both directions, so as such it cannot be described as an ordinary OD-model. But, there are non-empty symbols only on each cell of finite section of the memory tape. Because of this, it is possible to bind to a pair of indexes with MT memory tape: i_{min} and i_{max}, which will point to starting and ending cell of that finite portion. Value of index i_{min} is chosen in such a way that all the cells left of this point are empty b_0. And value of index i_{max} is chosen in such a way that all the cells right of this point are empty b_0.

In this way, it is possible to represent MT memory tape using finite set of cells or objects, each of them have their own index in the range from i_{min} to i_{max}. Each cell from memory tape describes object o_i^{Cell}, where $i \in [i_{min},\ i_{max}]$.

Object o_i^{Cell} has 4 attributes $A_{o_i^{Cell}} =< i,\ b,\ t_0,\ t_d >$, where
i – is a current cell index on memory tape,
b – a symbol in the current cell ($b \in B$),
t_0 – a birth time of the cell,
t_d – a death time of the cell.

MT tape itself is represented by an o^{Lenta} object, which has 3 attributes $A_{o^{Lenta}} =< i_{min},\ i_{max},\ t_0 >$, where
i_{min} – is an index of the leftmost cell, containing non-empty symbol in the considered tape section,
i_{max} – an index of the rightmost cell, containing non-empty symbol in the considered tape section,
$t_0 = 0$ – a birth time of MT memory tape object.

Since the memory cells are located on the MT memory tape, then there are links $l^{o^{Lenta}}_{o_i^{Cell}}(i \in [i_{min},\ i_{max}])$ between object o^{Lenta} and objects o_i^{Cell} which indicate that the cell o_i^{Cell} belongs to the tape o^{Lenta}.

Due to the fact that procedures $f \in F'$ were initially designed to use a set of corteges of values $\{V_{o_j}(t)|j \in \overline{1, m(t)}, o_j \in O, t \in [t^n_{o_j},\ t^k_{o_j}]\}$ of all the objects which are significant for the procedure and a set of links among objects as arguments to produce a set of corteges of values at the moment next after the procedure's execution, we assume that there are such significant objects and object-links only are placed on the tape (in the form of attribute values, delimited by special characters).

Introduce format, in which OD-model objects are written on memory tape. For simplicity, only b cells attributes (characters) are considered significant for objects representation on memory tape, other cell attributes belong to cells only and does not refer to the objects from OD-model, written on memory tape. First, we consider how object's attribute is written: attribute name, then – delimiter character "-", then – attribute value, for example, $a - 2$. Object consists of several attributes, delimited by "+" character. And finally, objects are delimited by "*" character. Beside of representing object attributes themselves, is is also necessary to identify an object, to accomplish this, object representations are prefixed by their IDs (OD-model object ID) and a special delimiter character "|": $1|a - 1$. For example, some objects o_1, o_2 with attributes $(a = 1,\ b = 2,\ c = 3)$, $(g = 5,\ l = 3)$ are written in a form $o_1|a-1+b-2+c-3*o_2|g-5+l-3$. All these delimiters must be included in $OD.MT$ external alphabet. We consider that only object's representation format is predefined, the formats of result, or order of objects are unique for every $OD.MT$.

At each moment of time MT control unit inspects a cell on the memory tape, and, according to its functional table and current state, MT determines what symbol will be written into current cell and where control unit's head should be moved. Control unit is defined by object o^{UU}, which has 4 attributes $A_{o^{UU}} =< i,\ q,\ d,\ t_0 >$, where
i – is a cell index, which is currently inspected by CU's head,
q – a current Turing machine state $(q \in Q)$,
d – CU's head movement on the current MT work step (1 – one cell to the left, -1 – one cell to the right, 0 – no movement),
$t_0 = 0$ – time of CU's object creation.

Each step of MT's work is defined by MT's functional table: using current state q and currently inspecting symbol b CU, basing on rules from functional table, makes decision what symbol should be written to the tape and determines new q_{new} state of MT and direction of the head movement d.

MT's functional table is described by a set of $o^{MT}_{q,\ b}$ objects. Each $o^{MT}_{q,\ b}$ describes exactly one rule from the instructions table. Object $o^{MT}_{q,\ b}$ has no links with other objects $(L_{o^{MT}_{q,\ b}} = \oslash)$ and has 6 attributes $A_{o^{MT}_{q,\ b}} =< q,\ b,\ q_{new},\ b_{new},\ d,\ t_0 >$, where

q – is a current MT state ($q \in Q$),
b – a character, currently inspected by MT ($b \in B$),
q_{new} – a new MT state ($q_{new} \in Q$),
b_{new} – a new MT character ($b_{new} \in B$),
d – head movements (1 – one cell to the left, -1 – one cell to the right, 0 – no movement),
$t_0 = 0$ – time of instruction's creation.

At the moment of time $t = 0$ all the objects currently describing MT are created, after that the input word is defined and the starting state is settled. Then MT starts its work and its first step is executing at the $t = 1$ moment of time.

Each step of MT's work is an interaction between two objects: control unit and memory tape. This interaction is described by functions from OD-model, which, by object attributes' values at the moment of time t determine object attributes' values at the moment of time $t + 1$.

To describe OD-model's functions, which can define modification of objects on the memory tape and control unit, interacting on each step of MT's work, introduce objects and attributes marks:

- moment of time, at which the object is inspecting, is marked left to the object identifier in parentheses (for example, $o^{UU}(t)$ – CU object at the t moment of time);
- to specify object attribute's value a notation [*name of the object*].[*name of the attribute*] is used (for example $o^{UU}(t).q$ – current MT's state at the t moment of time);
- to refer to the memory cell object on the MT's memory tape with the specified index brackets to the right of the object name are used (for example, $o^{Lenta}[i]$ – memory cell with index i).

Each step of MT's work is performed in one discrete time interval $(t,\ t + 1]$: in time interval $(t,\ t + n]$ MT performs exactly n steps. Describe one step of MT work using OD-model's functions, which machine performs in time interval from t to $t + 1$.

Head's movement ($o^{UU}(t+1).d$), character currently recording onto the memory tape ($o^{Lenta}[o^{UU}(t).i](t+1).b$) and the new MT's state ($o^{UU}(t+1).q$) are identified by following OD-model's functions:

1. $o^{UU}(t + 1).d = f_d(o^{Lenta}[o^{UU}(t).i](t).b,\ o^{UU}(t).q)$, where

   $o^{UU}(t).i$ – an index of the currently inspected MT's cell at the t moment of time,

   $o^{Lenta}[o^{UU}(t).i](t).b$ – a character in the currently inspected MT's cell at the t moment of time,

   $o^{UU}(t).q$ – an MT's state at the t moment of time,

   f_d – a function, which determines head's movement and depends from current MT's state q and b character in currently inspecting MT's cell at the t : $f_d() = o^{MT}_{b,\ q}.d$ moment of time.

2. $o^{Lenta}[o^{UU}(t).i](t + 1).b = f_b(o^{Lenta}[o^{UU}(t).i](t).b,\ o^{UU}(t).q)$, where f_b – a function, which identifies new character that to be written onto the memory tape cell, that is

currently inspected at the t moment of time. Function f_b depends on current MT's state and a character currently inspected in MT's cell at the t : $f_b() = o^{MT}_{b,\ q}.b_{new}$ moment of time.

3. $o^{UU}(t+1).q = f_q(o^{Lenta}[o^{UU}(t).i](t).b,\ o^{UU}(t).q)$, where f_q is a function, which determines new MT's state at the $t+1$ moment of time. Function depends on current MT's state and a character being currently inspected by CU at the t : $f_q() = o^{MT}_{b,\ q}.q_{new}$ moment of time.

Index of currently inspected MT's cell at the $t+1$ moment time is determined in this way:

$$o^{UU}(t+1).i = o^{UU}(t).i + o^{UU}(t+1).d$$

After identification of index of a new inspected cell, it is possible that a cell with this index does not exists on the memory tape. In this case there is a need of creating a new memory cell and of definition of attributes of the newly created memory cell object. At the cell's birth time, the cell is defined by its number, which is equal to the number of new examined cell. An empty symbol b_0 is written into the cell. The moment of the cell's «birth» time is set to $t+1$, and the cell's «death» is set to infinity. Necessity to create a new cell can appear in 2 cases only:

- if index of a new memory cell is less than leftmost cell index $o^{UU}(t+1).i < o^{Lenta}(t).i_{min}$.
  In this case index of the leftmost cell is decreased by 1 $o^{Lenta}(t+1).i_{min} = o^{Lenta}(t).i_{min} - 1$.

- if index of a new memory cell is greater than rightmost cell index $o^{UU}(t+1).i > o^{Lenta}(t).i_{max}$.
  In this case index of the leftmost cell is increased by 1 $o^{Lenta}(t+1).i_{max} = o^{Lenta}(t).i_{max} + 1$.

After MT finishes its step, a situation when at one of the ends of memory tape utmost cell contains empty symbol b_0 and CU's head do not inspect this cell can happen. In this case such a cell is removed from the tape. If that cell was the leftmost cell, then MT's memory tape's cell's leftmost index is increased by 1 $o^{Lenta}(t+1).i_{min} = o^{Lenta}(t).i_{min}+1$. On the other hand, if that cell was the rightmost cell, then MT's memory tape's cell's rightmost index is decreased by one $o^{Lenta}(t+1).i_{max} = o^{Lenta}(t).i_{max} - 1$.

Using conditional construction **if** ... **then** ... **end if** and the **new** operator for creating a new object, it is now possible to represent one step of MT's work in a form of an algorithmic procedure:

$$\begin{aligned}
&o^{UU}(t+1).d = f_d(o^{Lenta}[o^{UU}(t).i](t).b,\ o^{UU}(t).q)\\
&o^{Lenta}[o^{UU}(t).i](t+1).b = f_b(o^{Lenta}[o^{UU}(t).i](t).b,\ o^{UU}(t).q)\\
&o^{UU}(t+1).q = f_q(o^{Lenta}[o^{UU}(t).i](t).b,\ o^{UU}(t).q)\\
&o^{UU}(t+1).i = o^{UU}(t).i + o^{UU}(t+1).d
\end{aligned}$$

if $o^{UU}(t+1).i < o^{Lenta}(t).i_{min}$ **then**
$o^{Cell}_{new} =$ **new** o^{Cell}
$o^{Cell}_{new}.t_0 = t+1$
$o^{Cell}_{new}.t_d = \infty$
$o^{Cell}_{new}.b = b_0$
$o^{Cell}_{new}.i = o^{Lenta}(t).i_{min} - 1$
$o^{Lenta}(t+1).i_{min} = o^{Lenta}(t).i_{min} - 1$
$o^{Lenta}[o^{Lenta}(t+1).i_{min}] = o^{Cell}_{new}$
end if

if $o^{UU}(t+1).i > o^{Lenta}(t).i_{max}$ **then**
$o^{Cell}_{new} =$ **new** o^{Cell}
$o^{Cell}_{new}.t_0 = t+1$
$o^{Cell}_{new}.t_d = \infty$
$o^{Cell}_{new}.s = b_0$
$o^{Cell}_{new}.i = o^{Lenta}(t).i_{max} + 1$
$o^{Lenta}(t+1).i_{max} = o^{Lenta}(t).i_{max} + 1$
$o^{Lenta}[o^{Lenta}(t+1).i_{max}] = o^{Cell}_{new}$
end if

if $o^{Lenta}[o^{Lenta}(t).i_{max}](t+1).b = b_0$ **then**
$o^{Lenta}[o^{Lenta}(t).i_{max}](t+1).t_d = t+1$
$o^{Lenta}(t+1).i_{max} = o^{Lenta}(t).i_{max} - 1$
end if

if $o^{Lenta}[o^{Lenta}(t).i_{min}](t+1).b = b_0$ **then**
$o^{Lenta}[o^{Lenta}(t).i_{min}](t+1).t_d = t+1$
$o^{Lenta}(t+1).i_{min} = o^{Lenta}(t).i_{min} + 1$
end if

3.2. DIM-model's Turing machine description

DIM Turing machine's control unit can be emulated by an PL/ODQL procedure MT. Work of $DIM.MT$ starts from launching corresponding MT procedure, inside of which, in an endless cycle, reading and writing from / to memory tape are performed before stop state q_0 will be reached.

Now, a way to build MT for using in DIM should be considered. And also, very important to construct and consider a projection $G(OD.MT) = DIM.MT$, that allows to construct such a Turing machine for DIM, which will be equivalent to $OD.MT$.

The main idea in constructing $DIM.MT$ is to build a mapping between main $OD.MT$ Turing machines components, and namely memory tapes and control units.

Similarly to $OD.MT$, in $DIM.MT$ an o_{DIM}^{Lenta} object including an array of o_{DIM}^{Cell} objects is used as "input" / "output" memory tape (we assume, that at the every moment of time only finite part of the tape is inspected)

DIM Turing machine's control unit can be modelled as PL/ODQL procedure MT. $DIM.MT$ execution starts from launching corresponding MT procedure, inside of which, in infinite cycle, read write operations of characters from to the memory tape are performed. Information about objects is transformed in according to transition (in a case in inheritance) of some attributes to parental objects. If such an information modification is done for one of the children objects and do not changes for another child object, thus, parent object's attribute values are copied and each of these copies are written to the memory tape and bound with another child objects.

Now, consider how it is possible to build Turing machine analogous to OD.MT using PL/ODQL procedures. Let, external alphabet of such a Turing machine consist of m characters together with an empty character b_0 and earlier described characters from $OD.MT$ alphabet: $B = \{b_0,\ b_1, \ldots,\ b_m\}$. On the other hand, inner machine's alphabet consist of n inner states and a special finish state $q_0 : Q = \{q_0,\ q_1, \ldots,\ q_n\}$. When one launches MT (executes corresponding procedure), i.e. at the moment of time $t = 0$ q variable is set to q_0.

Just like as in the case with $MT.OD$ infinite memory tape can be considered partly: on each step of $MT.DIM$ work (at each iteration of MT procedure's main loop) the machine inspects only the finite part of memory tape *tape*. In this case a PL/ODQL object o_{DIM}^{Lenta} with ad array of o_{DIM}^{Cell}'s is used as a memory tape, consequently, i_{min} is always equals to 0, and i_{max} is equals to memory tape o_{DIM}^{Cell} objects array length.

Below an example of PL/ODQL module, which models Turing machine's work within the DIM's model, is given. The machine's work starts with MT procedure launch, which performs processing of objects located on memory tape.

By analogy with above given Turing machine for OD-model, logic of the machine's operation is defined by a set of functions dim_{f_d}, dim_{f_b} and dim_{f_q}. Function dim_{f_d} defines "reading-out head"ś shift and depends, just like as in case with $MT.OD$-function, on current $MT.DIM$'s state and currently inspecting character b at the current moment of time. dim_{f_b} – used to compute character, which should be written into currently inspecting memory tape's cell and depends on current $DIM.MT$'s state and currently inspecting character b (similar to $OD.MT$'s function f_b). dim_{f_q} – used to compute inner $DIM.MT$'s state q at the moment of time $t+1$, and depends on current $DIM.MT$ state and currently inspecting character (similar to $OD.MT$'s function f_q).

Additional helper functions, also used in MT procedure:

- **get** – obtains character with index i from the memory tape *tape*;
- **set** – writes character b to the memory tape *tape* at i position;
- **lextend** – extends memory tape to the left, i.e. performs array of cells extension, adds o_{DIM}^{Cell} with b_0 character to the left of the array, i_{min} decreases by one, thus adds an empty memory cell at the left border of the memory tape;
- **rextend** – extends memory tape to right, i.e. performs array of cells extension, adds o_{DIM}^{Cell} with b_0 character to the right of the array, i_{max} increased by one, thus adds an empty memory cell at the right border of the memory tape.

```
DIM_MT BEGIN
...
PROCEDURE MT (o_DIM^Lenta tape) IS
    VAR INTEGER i_min := 0; // memory tape left border
    VAR INTEGER i_max := Length(tape); // memory tape right border
    VAR STRING q := q_0; // current MT state
    VAR INTEGER i := i_min; // index of currently being inspected character
    VAR STRING b := b_0; // character currently being written
    VAR INTEGER d := 0; // current shift

        BEGIN
            // main loop
            WHILE (q <> q_0)
            BEGIN
                d := dim_{f_d}(get(tape, i), q); // current shift
                b := dim_{f_b}(get(tape, i), q); // character to write
                q := dim_{f_q}(get(tape, i), q); // new state
                i := i + d; // currently being inspected character's index

                                IF i < i_min
                                    lextend(tape);
                                    i := i_min;
                                    b := b_0;
                                END IF

                                IF i > i_max
                                    rextend(tape);
                                    i_max := i_max + 1;
                                    b := b_0
                                END IF

                    // write character to the memory tape
                    set(tape, i, b);
                END;

            END;
            ...
            END DIM_MT;
```

3.3. Object attributes dynamic's completeness

An algorithm $f \in F^{'}$ launch of an arbitrary OD-model at the moment of time $t \in T$ is bound to 2 static descriptions of the model using DIM classes schemes: S_0 at the t moment of time, right before f launch and S_1 at the $t+1$ moment of time right after the procedure launch finishes. Strictly speaking, a *DIM interaction $b \in B$ describes dynamics of objects' attributes' values modifications, which is initiated by f algorithm, if this interaction transforms S_0 DIM scheme classes into S_1 scheme.*

Theorem 1. *Theorem on completeness of the Objects attributes values' dynamic.* For an arbitrary algorithm $f \in F^{'}$ of an arbitrary OD-model there is a DIM interaction DIM $b \in B$, which describes process of changing objects properties, caused by f algorithm execution at arbitrary moment of time $t \in T$.

Proof. Let, an arbitrary OD-model algorithm is represented in a form of $OD.MT$ Turing machine. To prove this statement it is necessary to build DIM Interaction, which performs DIM objects modifications, similar to OD-model's objects modifications which can be done by f algorithm Turing machine. We assume, we decouple MT functionality from the source of objects, and put $OD.MT$ functions into $DIM.MT$, will execute $OD.MT$ inside of DIM and on the DIM objects.

1) build a Turing machine model inside of DIM, which realizes exactly the same modifications of objects, like $MT.DIM$ with OD-objects. As was mentioned above, $DIM.MT$ can be modelled in a form of special procedure, which transforms an array of character objects, written on memory tape. Because of fact that $OD.MT$ execution logic is described by a set of functions f_d, f_b and f_q, which depends on current MT's state and character being currently inspected, it is possible to emulate these functions behavior in $DIM.MT$, inside of DIM_MT module: f_d^{DIM}, f_b^{DIM} and f_q^{DIM}.

2) in DIM, interactions take a set of objects, which are fetched using **FROM** query, as an input parameters, instead of using $DIM.MT$ memory tape object as an argument.

3) it is necessary to transform a set of DIM objects and a o_{DIM}^{Lenta} memory tape object. This transformation can be performed by a special encoding function, which writes objects to the memory tape, according to the format given with the $OD.MT$ (objects representation format is predefined). After memory tape object, which satisfy conditions $G^{-1}(o_{DIM}^{Lenta}) = o^{Lenta}$ and $G^{-1}(o_{i_{DIM}}^{Cell}) = o_i^{Cell}, \forall i \in [i_{min},\ i_{max}]$, was built, it is possible to turn to building procedure's main cycle, modelling MT operating.

BEGIN
// main cycle
WHILE $(q <> q_0)$
BEGIN
$d := dim_{f_d}(\mathbf{get}(tape,\ i),\ q)$; *// shift*
$b := dim_{f_b}(\mathbf{get}(tape,\ i),\ q)$; *// character for write*
$q := dim_{f_q}(\mathbf{get}(tape,\ i),\ q)$; *// new state*
$i := i + d$; *// current state*
...

4) after MT stops (i.e. after main cycle iterations were finished) it is necessary to make reverse transformation: transform $DIM.MT$ memory tape objects back to DIM objects.

   Obtained after application of the inverse transformation, a set of objects is a result of $OD.MT$ processing after DIM objects. However, objects, obtained in such a way, not always correctly represents modifications, for example, if was modified an attribute, which belongs to the parent object (not to the current object), a tryout to persist this object into DB will also modify other objects (which are the children for the parent object).

   The solution lies on the fact that in such cases (when not all the child objects take part in modification of attributes values) the parent objects to be copied and only its copy is modified. Also, the link to the parent object is redefined for the base object, which was modified during algorithm execution.

   The implementation of this approach is an application of procedure $NORMALIZE$, which uses 2 sets of objects: source set of objects *source* and a resulting set *result*:

   1) for an every object o_s from the source set, find corresponding object o_r from the resulting set;
   2) if the source object o_s was modified, identify rather modified attributes belong to the object itself, or to its parent object;
   3) if inherited attributes were changed, it is necessary to create a copy of the unmodified object in the *result* set, for the object o_r itself – change a link to the parent object to the parent object copy with unmodified attributes.

   Thus, attribute values of objects, which are not modified in MT work process, will not be broken.

   Similarly to this process, it it necessary to make such an operations with inclusion objects and including objects – we assume that in this case a new object is created instead of the old one. To implement this, a procedure $NORMALIZE\_INCLUSION$, which uses 2 sets of objects: a source set *set* and a resultant set *result*, is used:

1) for an every object o_s from the source objects set, which is an inclusion or including object, find corresponding object o_r from the resulting set;
2) if the source o_s was modified, it is necessary to create its copy in the resulting set $o_{r_{copy}}$;
3) create history links between object o_s and its modified version $o_{r_{copy}}$.

Analogous actions on creation object-copy and linking it through history relation with the source object are taken for objects, for which have been changed identifying attributes, and these actions are implemented using $NORMALIZE\_ID$ procedure.

5) After executing all the operations from p.4, the resulting set contains objects which are modified in quite the same way as while executing $OD.MT$ on objects from OD-model;

6) Interaction runtime environment persists objects from resulting set into metalevel. Actually, the environment also creates new objects (refer to the p.4) and creates history, inheritance and inclusion links between objects (using ODQL queries).

Thus, after all actions were accomplished, DIM interaction $b \in B$, which describes dynamics of objects attributes values modifications, which were initially performed by f algorithm's execution at arbitrary moment of time $t \in T$ is designed. Execution of the interaction performs exactly the same DIM-model objects' attributes values modifications, as execution of an algorithm $f \in F'$ over objects from the source OD-model.

□

References

[1] Codd E. F., "A relational model for large shared data banks", *Comm. ACM*, 1970.

[2] Codd E. F., *Further normalization of the database relational model, in Database Systems*, Prentice Hall, Englewood Cliffs, NJ, 1972.

[3] Atkinson M. et al., "The Object-Oriented Database System Manifesto.", *Elsevier Science*, 1990.

[4] Garcia-Molina H. et al., *Database Systems: The Complete Book (2nd Edition)*, Pearson Prentice Hall, Upper Saddle River, NJ, 2008.

[5] Greene Robert, "OODBMS Architectures. An examination of implementations.", *Technical report, Versant Corp.*, 2006.

[6] Kostenko B. B., Kuznetsov S. D., "Istoriya i aktualnye problemy temporalnykh baz dannykh", 2007, http://www.citforum.ru/database/articles/temporal, (in Russian).

[7] Pisarenko D. S., Roublev V. S., "Object DBMS DIM and its main concepts", *Modeling and analysis of information systems*, **16**:1 (2009), 60–87.

[8] Roublev V. S., "The Object Query Language of the Dynamic Information Model DIM", *Modeling and analysis of information systems*, **17**:3 (2010), 144–161.

[9] Roublev V. S., "Zaprosnaya polnota yazyka ODQL dinamicheskoy informatsionnoy modeli DIM", Fiziko-matematicheskie i estestvennye nauki, **1**, Yaroslavl, 2011, 69–75, (in Russian).

[10] Roublev V. S., "Object Query Computing Optimization in the Dynamic Information Model DIM", *Modeling and analysis of information systems*, **18**:2 (2011), 39–51.

[11] Roublev V. S., "Teorema o staticheskoy polnote SUBD DIM", *Problemy teoreticheskoy kibernetiki. Materialy XVII mezhdunarodnoy konferentsii* (Kazan, 16–20 iyunya 2014 g.), Otechestvo, Kazan, 2014, 242–245, (in Russian).

[12] Roublev V. S., Petrov A. N., "Yazyk PL/ODQL i mnozhestva s indeksami", Estestvennye nauki, **3**, YaGPU, Yaroslavl, 2012, 74–83, (in Russian).

[13] Petrov A. N., "PL/ODQL language and sets with indexes", Science Drive – 2012. Yaroslavl, 2012.

[14] Pisarenko D. S., "Yazyk matematicheskikh formul DIM-FL Dinamicheskoy informatsionnoy modeli DIM", *Studencheskie zametki po informatike i matematike: sbornik nauchnykh statey studentov i aspirantov fakulteta IVT*, **3**, YarGU, Yaroslavl, 2008, 88–96, (in Russian).

[15] Petrov A. N., "Yazyk formul DIM-FL i ego realizatsiya v SUBD DIM", *Molodaya nauka v klassicheskom universitete: Tezisy dokladov nauchnykh konferentsy festivalya studentov, aspirantov i molodykh uchenykh* (Ivanovo, 21–25 aprelya 2014), Ivanovsky gosudarstvenny universitet, Ivanovo, 2014, 44–45, (in Russian).

[16] Moore Sh., Belden E., "Oracle Database PL/SQL Language Reference, 11g Release 2", 2013.

[17] Petrov A. N., "Vzaimodeystvi SUBD DIM i ikh realizatsiya", *Sbornik dokladov Mezhdunarodnoy konferentsii "II vesennie nauchnye chteniya"* (17 may 2014), Nauchno-informatsionny tsentr "Znanie", Donetsk, Ukraina, 2014, (in Russian).

[18] Petrov A., "Dynamic Information Model Interactions: Design and Implementation of Database-Driven Workflow Approach", *Preliminary proceedings of the 8th Spring/Summer young researches' colloquiu on software engineering (SYRCoSE 2014)*, Saint Petersburg, Russia, 2014, 177–181.

[19] Petrov A. N., "PL/ODQL interpreter – realizatsiya yazyka programmirovaniya PL/ODQL, primenyaemogo v SUBD DIM", Svidetelstvo o gosudarstvennoy registratsii programmy dlya EVM № 2014661854.

[20] Petrov A. N., "Programma dlya vypolneniya programmnykh moduley, sozdannykh na yazyke DIM-FL", Svidetelstvo o gosudarstvennoy registratsii programmy dlya EVM № 2015611827.

Полнота динамики значений свойств данных в СУБД DIM

Петров А. Н., Рублев В. С.

Ярославский государственный университет им. П. Г. Демидова
150000 Россия, г. Ярославль, ул. Советская, 14

Ключевые слова: объектная СУБД, динамика данных, полнота представления

Данная работа посвящена обоснованию возможности использования объектной СУБД DIM и ее механизма взаимодействий в качестве алгоритмически полной реализации объектно-динамической модели. В статье описывается расширение статической OD-модели путем включения в неё множеств алгоритмических процедур, описывающих изменения значений свойств объектов, а также создание, удаление и изменение самих объектов. Для обеспечения DIM возможностями модификации данных, эквивалентной модификациям в OD-модели, вводятся отношения взаимодействий и истории. Для того, чтобы минимизировать зависимость от конструкций описаний алгоритмических процедур OD-модели, которые могут быть записаны на различных языках, выполняется сведение аппарата процедур к универсальной форме – машине Тьюринга. Представляется способ построения машины Тьюринга эквивалентной $OD.MT$ в рамках DIM, использующей набор PL/ODQL процедур в качестве аналога управляющего устройства и функциональной таблицы. Описывается принцип формирования ленты памяти такой $DIM.MT$ путём кодирования информации об объектах DIM и их последующего декодирования с ленты обратно в объекты DIM. При этом процесс работы такой машины моделируется с помощью бесконечного цикла выполнения PL/ODQL процедур чтения / записи объектов с входной ленты. В заключение приводится доказательство теоремы о полноте представления динамики данных математической модели DIM новой объектной СУБД, основанное на доказанной ранее теореме о статической полноте представления данных в DIM.

Статья публикуется в авторской редакции.

Сведения об авторах:
Петров Алексей Николаевич,
Ярославский государственный университет им. П. Г. Демидова,
аспирант,
orcid.org/0000-0002-1037-866X
Рублев Вадим Сергеевич,
Ярославский государственный университет им. П. Г. Демидова,
профессор,
orcid.org/0000-0001-8095-3132