Organ-Matrix and Cell-Matrix

Arnold Sher
ICSD, Inc.
46 Donora Drive
Toronto, ON M4B 1B5
CANADA

Abstract – The structures called organ-matrix and cell-matrix are introduced. Organ-matrices contain data and operators as data, which can be stored in database tables and reflect business logic in particular business domain. An algorithm can be presented by one row of an organ-matrix. A set of algorithms in a particular business domain can be represented by an entire organ-matrix. It’s shown that an organ-matrix is a generalization of widely used 2-dimensional rectangular numeric matrices. Operations upon organ-matrices are introduced. Organ-matrices contain all the information necessary for building dynamic stored procedures. The effectiveness of organ-matrices in the re-engineering of the business system is shown.

I. INTRODUCTION

Software covers different types of businesses, artificial intelligence and scientific fields. There is a big difference between the logic of business and of let’s say physics. Physics laws don’t change much. In physics, operations upon data are well defined and very stable. In business, operations upon data depend on rapidly changing market conditions, the type of business, users and customers, states and countries, even on the scale of the business. Business logic is changed when companies merge and when they split, when they have a lot of competition and when they are the major players on the market. Business logic depends on hundreds and thousands of different parameters and hundreds of thousands of different combinations of parameters and conditions.

But when it comes to development of software, the methodology used for scientific and business software is the same: the heavy usage of stored procedures (where business and scientific logic is located and hard-coded) and database tables where data are stored.

In this paper an alternative approach to the building of business and artificial intelligence systems, which require flexible and rapidly changing logic are considered.

II. DEFINITION OF ORGAN-MATRIX OF GENERAL FORM

Note a structure as \( \hat{A} = \{M_j(t)\}_{j=1}^{m(j)} \), where

\[
M_j(t) = \left\{ s_i^j(t), \bullet_i^j \right\}_{i=1}^{L(j)}
\]

with elements

\[
s_i^j(t) = \left\{ \rho_{s_i^j}^k(t) \right\}_{k=1}^{K(j)}, \text{ } s_i^j \text{ is a set of meanings } \rho_{s_i^j}^k \text{ out of some } R^K \]

space – which is generally speaking different for different \( s_i^j \) and operator

\[
\bullet_i^j \in \{\in, \notin, \mu(),+, -, \times, \..., \&\land, \lor, \land, \neq, \neq, \neq, \neq, \}
\]

or any other operations or references to operations. Elements \( s_i^j \) can be numeric or non-numeric; they can belong to ordered or not ordered sets; they can be functions or inequalities; they can refer to other objects, and they can be of different data types. There are no restrictions so far on the nature and scales of \( s_i^j \) elements.

In the real world sometimes it is easier to describe the set that parameter’s value does not belong rather than vise versa. Operator \( \bullet_i^j \) can have strict a mathematical meaning of “belong to” or “doesn’t belong to” or can be a “soft” membership function like it is in fuzzy set theory and can be stored in database tables as data associated with other data \( s_i^j(t) \). Operator \( \bullet_i^j \) can have simple mathematical sense or serve as a reference to a complicated stored procedure.

General matrix \( \hat{A} = \{M_j(t)\}_{j=1}^{m(j)} \) we’ll call organ-matrix. Organ-matrices are 3-dimensional in static and 4-dimensional in dynamic (when time is taken into account). Organ-matrix contains different number of rows in columns and different number of columns in rows.

Any sub-matrix of an organ-matrix we’ll call cell-matrix.

Let’s consider elements \( s_i^j(t) = \left\{ \rho_{s_i^j}^k(t) \right\}_{k=1}^{K(j)} \) of a matrix. Assume that for

\[
\forall l = 1, L(j) : \exists p \in \{1, 2, ..., L(j)\} \& p \neq l : s_p = \text{Name}(s_i^j).
\]

Element \( s_p \) can be the name of \( s_i^j \) or a reference to the name of the last one. To keep formulas as simple as possible, name and reference to the name will be treated in the same way though for practical data processing they are different.
III. RELATIONSHIP BETWEEN ORGAN-MATRICES AND COMMONLY USED 2-DIMENSIONAL RECTANGULAR NUMERIC MATRICES

The definition of organ-matrix is a generalization of a mathematical (2-dimensional numeric rectangular) matrix and it can be reduced to it if simultaneously all following restrictions applied:

1) \( \forall j: L(j) = \text{const} \) and
2) \( \forall j: m(j) = \text{const} \) and
3) \( \forall j: K(j) = 1 \) and
4) \( \forall j, l, k : \rho^j_{lk} \) is number and
5) \( \forall l, j : \cdot^j_l \in \{+,-\} \).

When we mention numerical matrices we wouldn’t refer to particular publications but rather to scientific domains called “algebra” and “mathematical programming”, where numerical matrices are the main objects of research and development and on which tens (maybe hundreds) of thousands publications were written in the last century.

We can expect that there are many features, which characterize 2-dimensional numerical matrices (we’ll call them n-matrices), which are not applied to organ-matrices. There are features of organ-matrices that are applied to n-matrices as well.

Arithmetic operations between scalar and organ-matrix or between organ-matrices don’t have any sense if not all \( \rho \) are numeric.

Arithmetic operations between n-matrices have sense if and only if they are produced on elements with the same name, which means that a sum between birth date and blood pressure has no sense. In n-matrices number \( i \) of a column and number \( j \) of a row can serve as the name of the element laying on intersection between \( i \)-column and \( j \)-row or they can refer to \( i \)-th name in a list of column names and \( j \)-th name in a list of row names.

IV. OPERATIONS UPON ORGAN-MATRICES

Statement 1: Any algorithm (formal procedure) can be represented by a respective organ-matrix. The reverse assertion, generally speaking, is not true.

Proof. By definition an algorithm is a set of data and operations upon them. As far as the definition of organ-matrices doesn’t contain any restrictions on data and operations as data, so organ-matrices include any data and any operations upon them, which constitute an algorithm and make true the direct assertion. On the other hand, by definition, an organ-matrix may or may not contain operators. If it doesn’t contain operators, then an algorithm doesn’t exist, which can be equivalent to this organ-matrix. The Statement 1 is proved.

In the sense of the Statement 1 an organ-matrix is a generalization of the concept “algorithm”.

Definition: Organ-matrix \( \widehat{A} = \left\{ \begin{array}{l} l_{i, j}^{(m(j))} \end{array} \right\}_{j=1}^{m(j)} \) is reducible upon columns \( l_1, l_2 \in \{1, 2, \ldots, L(j)\} \), \( l_1 \neq l_2 \) for given row \( j \in \{1, 2, \ldots, m(j)\} \), if \( \text{Name}(s^j_{l_1}) = \text{Name}(s^j_{l_2}) \).

Definition: The reduction of organ-matrix \( \widehat{A} \) upon columns \( l_1, l_2 \in \{1, 2, \ldots, L(j)\} \), \( l_1 \neq l_2 \) for given row \( j \in \{1, 2, \ldots, m(j)\} \) is an organ-matrix \( \widehat{A} = \widehat{A} \setminus \{s^j_{l_1}, s^j_{l_2}\} \), with \( s^j_{l_1} = s^j_{l_1} \cup s^j_{l_2} \).

In other words, if an organ-matrix has a row with two columns with the same name (column numbers \( l_1, l_2 \) refer to the same name), then the meanings of their elements \( \rho \) are united: \( s^j_{l_1} \cup s^j_{l_2} = \left\{ \rho^j_{l_1 k} \right\}_{k=1}^{K(j)} \cup \left\{ \rho^j_{l_2 k} \right\}_{k=1}^{K(j)} \).

The reduction operation is symmetrical relatively to columns and rows.

Definition: An organ-matrix \( \widehat{A} \) is irreducible if \( \forall j = 1, m(j) : \exists l_1, l_2 = 1, L(j) \) & \( l_1 \neq l_2 \); and \( \text{Name}(s^j_{l_1}) = \text{Name}(s^j_{l_2}) \)

\( \forall l = 1, L(j) : \exists j_1, j_2 = 1, m(j) \) & \( j_1 \neq j_2 \);.

\( \text{Name}(s^{j_1}_{l}) = \text{Name}(s^{j_2}_{l}) \)

Reduction operations can convert any reducible organ-matrix into an irreducible one.

To simplify notation we’ll assume further that all organ-matrices are irreducible.

Definition: An organ-matrix \( \widehat{C} = \widehat{A} \bigcap \widehat{B} \) \( (\widehat{C} \) is an intersection of organ-matrices \( \widehat{A} \) and \( \widehat{B} \) ) if

\( \forall j \in \{1, m_j\} \) & \( l \in \{1, L_j\} \):

\( \exists \left( j_a = 1, m_j(j_a) \right) \) & \( l_a \in \{1, L_j(j_a)\} \) & \( \exists \left( j_b = 1, m_j(j_b) \right) \) & \( l_b \in \{1, L_j(j_b)\} \):

\[ \text{Name}(s^{l_a}_{j_a}) = \text{Name}(s^{l_b}_{j_b}) = \text{Name}(s^{l_a}_{j_b}) \] & \( s^j_{l_a} = s^j_{l_b} \cap \).

Definition: An organ-matrix \( \widehat{C} = \widehat{A} \bigcup \widehat{B} \) \( \widehat{C} \) is a union of organ-matrices \( \widehat{A} \) and \( \widehat{B} \) if
∀[(ja ∈ 1, ma(ja)) & (la ∈ 1, La(ja))]:  
∃[(ja ∈ 1, ma(ja)) & (la ∈ 1, La(ja))]:  
[Name(sj^a) = Name(sj^b)]

and

∀[(jb ∈ 1, mb(jb)) & (lb ∈ 1, Lb(jb))]:  
∃[(jb ∈ 1, mb(jb)) & (lb ∈ 1, Lb(jb))]:  
[Name(sj^a) = Name(sj^b)]

and

∃j, l, c:

[∀ja, la : Name(sj^a) ≠ Name(sj^b)] &
& [∀jb, lb : Name(sj^a) ≠ Name(sj^b)]

and

if Name(sj^a) = Name(sj^b) = Name(sj^b),
then sj^a = sj^b ∪ sj^b otherwise

if Name(sj^a) = Name(sj^b),
then sj^a = sj^b otherwise

if Name(sj^a) = Name(sj^b),
then sj^a = sj^b .

An organ-matrix can be embedded in another algorithm (to be an element of an algorithm). An organ-matrix can absorb an algorithm, which the former is a part of.

By definition, intersecting two or more organ-matrices is a cell-matrix for any of them. A union of two or more organ-matrices is an organ-matrix and initial organ-matrices are cell-matrices with respect to their union. A cell-matrix can be a complicated structure on its own. The definition of a cell-matrix makes sense only in regard to an organ-matrix, which includes this cell-matrix as its part. We’ll say that a matrix A is a covering matrix in respect to a matrix B if the last matrix is a cell-matrix regarding A or, is the same, if B ⊂ A; matrix B is covered by matrix A.

Definition of equality of organ-matrices:

A = B ⇔ [∀j : ma(j) = mb(j)] &
& [∀j : La(j) = Lb(j)] &
& [∀j : Ka(j) = Kb(j)] &
& [∀j, l, k : ρ^a_{lk} = ρ^b_{lk}]

We can consider grades of organ-matrices “likeness” or “closeness”. Generally speaking, organ-matrices are not strictly numerical, which means that the “metric” concept isn’t applied to them. But the “metric” and “closeness” concepts are applied to some of their elements with the same Names. If some elements (with the same Names) of organ-matrices are defined on an ordered set, then the “closeness” of these elements depends on the positions they are occupying in a set on which they are defined. For example, on a set [tiny, small, average, biggish, large, enormous] the “distance” between “tiny” and “average” is greater than between “tiny” and “small”.

It’s easy to make a conclusion about equality or inequality of organ-matrices in the sense of the above-mentioned definition. We also can make a conclusion about “closeness” of organ-matrices base on their structure when their elements (with the same names) defined on not ordered sets are the same. We don’t have a constructive view on “closeness” of organ-matrices based on their constructions in general. All we can say is that organ-matrices can be close in one sense and not close in another.

Let there be two organ-matrices A, B, a function (algorithm, procedure, stimulus) f and a cell-matrix C where C ⊂ A and C ⊂ B.

Definition of equivalence of organ-matrices:

A ∼= B ⇔ f(A) = f(B).

In other words, two organ-matrices A and B are equivalent in the sense of function f in the domain of cell-matrix C if and only if their functions f(A) and f(B) are equal in domain C.

As far as a continuum of functions f exists and there is an enormous number of possible cell-matrices for a given organ-matrix of a big size so there is a continuum number of “senses” of equivalence of organ-matrices that is possible for consideration and research.

Equal organ-matrices are always equivalent, but equivalent matrices are not always equal. Not equal organ-matrices can be equivalent in one sense and not equivalent in another. The concept of equivalence implies that there is a domain where both matrices produce the same result with the same stimulus. The result can be exactly the same or similar. As far as organ-matrices containing not just data but operators as data as well we can speak about their “behavior”. We can analyze organ-matrices based on their structure and/or their “behavior”.

In order for an organ-matrix to behave there should be a mechanism that starts (launches) of the matrix, which is external with respect to the matrix itself.
V. ORGAN-MATRICES AND DYNAMIC PROCEDURES

Assume that we need to write procedure \( P^*(a, b) \) that executes \( S = a + b \).

1. Input: \( a, b \in \{ \text{numbers} \} \).
2. Execute: \( S = a + b \).
3. Output: \( S \in \{ \text{numbers} \} \).

Here step 2 is hard coded. We can change data \( a \) and \( b \), but for any \( a \) and \( b \) the result of step 2 will always be \( S = a + b \) until we change procedure \( P^* \).

Using organ-matrix we write the procedure

\[
P(\rho_1, \rho_2, \rho_3, \rho_4, \rho_5) =
\]

\[= \text{concatenate}(\rho_1, \rho_2, \rho_3, \rho_4, \rho_5) =
\]

\[= \rho_1 \| \rho_2 \| \rho_3 \| \rho_4 \| \rho_5,
\]

1. Input: \( \rho_1, \rho_2, \rho_3, \rho_4, \rho_5 \in \{ \text{numbers, operations} \} \).
2. Execute: \( \rho_1 \| \rho_2 \| \rho_3 \| \rho_4 \| \rho_5 \)
3. Output: \( \rho_1 \),

where \( \| \) is concatenation operation.

As we can see in this case, when

\[
\rho_1 = S, \rho_2 = S, \rho_3 = a, \rho_4 = +, \rho_5 = b,
\]

procedure \( P(\rho_1, \rho_2, \rho_3, \rho_4, \rho_5) \) = \( S = a + b \) produces the same result as procedure \( P^* \). But now in \( P \) we can change not only data \( \rho_3 = a \) and \( \rho_5 = b \), but operators \( \rho_2 \) and \( \rho_4 \) without needing to change procedure \( P \). Procedure \( P \) in this case is very simple and defines the sequence of variables \( \rho \) in concatenation. For example, if

\[
\rho_1 = S, \rho_2 = S, \rho_3 = a, \rho_4 = x, \rho_5 = b
\]

then \( P(\rho_1, \rho_2, \rho_3, \rho_4, \rho_5) \) = \( S = a \times b \). Input data and operators as data \( \rho \) were changed while procedure \( P(\rho_1, \rho_2, \rho_3, \rho_4, \rho_5) \) was not.

The goal of procedure \( P \) is to recognize an operator and concatenate data depending on the operator. For example:

\[
P(a_1, a_2, a_3, a_4, a_5) :
\]

\[
\text{if } (\rho_1, \rho_3, \rho_5 \text{ are numbers }) \& (\rho_2 \in \{=, <, >, \neq, \leq, \geq \}), \text{ then :}
\]

\[
\begin{align*}
\text{if } \rho_4 \in \{+, -, x, \&, \vee \}, \\
\text{then } \rho_1 \| \rho_2 \| \rho_3 \| \rho_4 \| \rho_5 \\
\text{if } \rho_4 \in \{\div \}, \\
\text{then, if } \rho_5 = 0, \\
\text{then exception,} \\
\text{else } \rho_1 \| \rho_2 \| \rho_3 \| \rho_4 \| \rho_5, \\
\text{if } \rho_4 \in \{\max, \min \}, \\
\text{then } \rho_1 \| \rho_2 \| \rho_4 \| (\| \rho_3 \|, \| \rho_3 \|)
\end{align*}
\]

Another example, assume that we have stored procedure \( (SP) \) in SQL (structural queue language):

\[
\begin{align*}
\text{if } a = b & \text{ then } \\
& s = c; \\
\text{else if } a < b & \text{ then } \\
& s = d; \\
\text{else } \\
& s = z;
\end{align*}
\]

end if;

We can represent this stored procedure by one row of organ-matrix (which is the table of its own):

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \rho_1 )</th>
<th>( \rho_2 )</th>
<th>( \rho_3 )</th>
<th>( \rho_4 )</th>
<th>( \rho_5 )</th>
<th>( \rho_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( a = b )</td>
<td>( s = c )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( a &lt; b )</td>
<td>( s = d )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( a &gt; b )</td>
<td>( s = z )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then external procedure \( P \) gathers data from the table:

\[
P(\rho_{lk}) : \forall l \in \{1, 2, 3\} :
\]

\[
\text{if } \rho_{l1} \| \rho_{l2} \| \rho_{l3}, \text{ then } \rho_{l4} \| \rho_{l5} \| \rho_{l6}.
\]

It's obvious, that three elements of the organ-matrix represent given stored procedure \( SP \) with the use of procedure \( P \), which shows the way elements of the organ-matrix represent given stored procedure \( SP \):

\[
l = 1 : \text{ if } a = b \text{ then } s = c \\
l = 2 : \text{ if } a < b \text{ then } s = d \\
l = 3 : \text{ if } a > b \text{ then } s = z.
\]

Stored procedure \( SP \) is hard-coded, which means that given operations upon data can't be changed automatically. Procedure \( P \) is dealing with data and operators as data, which stored in database table and can be changed automatically by other procedures.

Formally speaking, procedure \( P \) (with data and operators as data stored in database table) has more general form than stored procedure \( SP \). It has additional parameters \( \rho_2 \) and \( \rho_5 \) to change compared to procedure \( SP \).

Procedures having the form known at execution time are called dynamic procedures. Organ-matrices contain (represent) all information necessary for building dynamic procedures.

Organ-matrices are self-sufficient. They contain all information necessary for their execution. But they are relatively self-sufficient because they need an external mechanism ("body-builder") to run them. This mechanism defines what parts of an organ-matrix and in what order it should be concatenated. This mechanism can have other
functions as well but “concatenation in particular order” is its most basic and important function.

One approach is to create an organ once, to store it (this is what we have when an entire business logic is concentrated in stored procedures in database or in functions and scripts in front-end). Obviously a body-builder for such a complex system is going to be very complicated and it requires a lot of resources for its maintenance.

Another approach is to recreate an organ from cells every time we want to use the organ and to have a relatively simple body-builder for launching it. This is where organ-matrices are a good fit.

The combination of the above mentioned approaches and the redistribution of business logic between stored procedures and organ-matrices allow to build computerized systems, which are optimal upon multiple criteria: speed, changeability and maintainability.

VI. IMPLEMENTATION OF ORGAN-MATRICES FOR RECOGNITION OF QUOTABILITY

Leasing contract payout quotability is the ability of a contract to meet determined requirements for calculations of payout. In this case there is a set of combinations of all requirements for payout calculation and a contract can meet (then it’s quotable) or can not meet (not quotable) at least one combination of requirements.

Let note a contract as

\[ \exists \tau \in [0, L] : \{ Name(x_i) = Name(s_j) \} \wedge \{ x_i(t^*) \cdot s_j^*(t^*) \} \]

In our terminology the last procedure is a body-builder for quotation of a contract \( X(t^*) \) against an organ-matrix \( A \).

Having operators \( \{\in, \notin\} \) is enough for our goal to determine whether an object \( x_i \) belongs or doesn’t belong to a set \( s_j \) while in practice, depending on a scale we measure objects, different operators can be used:

if \( a_y \leq x < b_y \) and \( x = x_i \),

then \( x_i \in [a_y, b_y] \);

if \( f(x) = 0 \) and \( x = x_i \) and \( f(x) = 0 \),

then \( x_i \in [f(x) = 0] \);

if \( s_j^* = \{ \rho_{hK} \}_{k=1}^n \) and \( \exists k : \rho_{hK} = x_i \),

then \( x_i \in s_j^* \);

if \( G(x) \leq 0 \) and \( x = x_i \) and \( G(x) <= 0 \), then \( x_i \in [G(x) <= 0] \).

Further we’ll avoid obvious (when it’s obvious) interpretation of mathematical operations in terms of \( \{\in, \notin\} \). The last operations can be interpreted in terms of membership functions of fuzzy sets as well.

Another simplification for current consideration is that we work in data integrated environment, which means that an apple is compared to an apple or fruit is compared to fruit or a plant is compared to a plant. They have to be selected from the same set:

\[ \forall \exists l, j : Name(s_j^*) = Name(x_i) \Rightarrow \]

\[ \Rightarrow \text{DataType}(s_j^*) = \text{DataType}(x_i) \wedge \]

\[ \wedge (s_j^* \subseteq S) \wedge (x_i \in S) \]

However, from a practical point of view, the assumption of data integrity can be right or wrong depending on the goal being considered.

In practice, a row, a column and a set of data for an element on their intersection can have different meaning. For example (Fig.1), a row can represent a procedure. Each element of a row can represent the name of a parameter, a combination of conditions, operators and an action and/or a reference to actions. Then all rows represent all procedures in a particular business domain. Fig.2 shows an example of an operation of concatenation of elements of an organ-matrix. This operation forms the
WHERE clause for select statement. The resulting WHERE clause formed by the algorithm shown on Fig.2 using data from Fig.1 is shown bellow:

select count(distinct c.Contract_No) from IL_CONTRACTS c, IL_ASSETS a where c.Contract_No = a.Contract_No and (c.RENEWAL_DATE is null or c.RENEWAL_DATE > (SYSDATE)) and (c.CONTRACT_TYPE is not null and c.CONTRACT_TYPE IN ("CS", "LP", "TL")) and (c.LESSOR_CODE is not null and c.LESSOR_CODE IN ("800", "801", "810", "811", "820", "821", "790", "791")) and (c.ACTIVATION_DATE is not null and c.ACTIVATION_DATE < (SYSDATE)) and (c.USER_CODE is null or c.USER_CODE NOT IN ("MF", "FR")) and (c.SUSPEND_CODE is null or c.SUSPEND_CODE IN ("C") and (c.PURPOSE_OF_LOAN is null and c.PURPOSE_OF_LOAN IN ("03", "04", "05")) and (c.ALLOW_QUOTE is not null and c.ALLOW_QUOTE IN ("1")) and (c.FLOAT_RATE is not null and c.FLOAT_RATE IN ("1")) and (c.FIRST_REFUSAL is null or c.FIRST_REFUSAL IN ("C", "D")) and (a.ASSET_VENDOR is not null and a.ASSET_VENDOR NOT IN ("0002000003") and (a.DISP_DATE is null or a.DISP_DATE > (SYSDATE)) and (a.PURCHASE_OPT_DATE is null or a.PURCHASE_OPT_DATE > (SYSDATE))

Assume that a business has some independent business unit \( b \in \{1, 2, \ldots, B\} \). Then we have a contract

\[
X^b(t) = \{x^b_1(t), x^b_2(t), \ldots, x^b_n(t)\} \in R^n,
\]

\( b \in \{1, 2, \ldots, B\} \) of business unit \( b \). For each business unit we have a respective matrix \( A^b \). Then the contract

\[
X^b(t^*) \text{ is quotable at a time } t^* \text{ if }
\]

\[
\exists j = \overline{1, m} : \forall l = \overline{1, L} : \\
[\text{Name}(x^b_j(t)) = \text{Name}(s^b_l(t))] \& \\
(\forall i \in \{1, \ldots, l\} : s^b_i(t^*) \neq 0) \\
\]
It means that the recognition of a contract of another business unit with different logic doesn’t require writing special procedures but rather to develop another organ-matrix (stored data and operators as data). In this case each business unit is associated with a particular organ-matrix and set of organ-matrices constituting a business “body”.

Adding another business unit to the system will require adding another organ-matrix with no stored procedures changes.

If the business logic of a particular business unit changes, it requires making changes in the corresponding organ-matrix without stored procedures changes and with no affect on other business units while they are sharing the same quotability recognition procedure as an external mechanism for launching them.

Assume that another business unit with another quotability recognition procedure is involved in the business. It means that we just put into an organ-matrix for each business unit a reference to recognition procedure it uses. We develop a new recognition procedure for a new business unit. We put reference to the new procedure to the organ-matrix for this business unit. Because of segregation of organ-matrices between business units, we don’t need to test all other business units but rather the only new one.

VII. CONCLUSION

The structures called organ-matrix and cell-matrix are introduced. According to their definition widely used 2-dimentional rectangular numeric matrices are a particular kind of a cell-matrix. An organ-matrix contains data and operators as data, which can be stored in database tables. Any algorithm can be presented by one row of an organ-matrix or by one element of a row depending on the complexity of an algorithm. A set of algorithms in a particular business domain can be presented by an entire organ-matrix. One element of organ-matrix has, generally speaking, more than one meaning. It contains a set of data, operators and references to actions.

An organ-matrix represents business logic in a particular business unit. Different business units can have different organ-matrices. A set of organ-matrices represent a business “body” in a particular business domain.

An organ-matrix requires external mechanism (procedure) for its launching. This mechanism can be very simple but it can contain other functions as well. A basic function for an external mechanism is a concatenation of elements in a particular order. An organ matrix contains all the data necessary for building dynamic procedures.

Organ-matrix is a form of storage of business logic. Organ-matrices allow re-distributing business logic between stored procedures and them, which makes stored procedure much “lighter” and the whole computerized business system more flexible to constantly changing environment and the accommodation of those changes by the business system. It’s easier to make changes in data than in complicated stored procedures. Having organ-matrices makes the system more maintainable, reduces time for accommodation of business requirements and saves money.

We see an analogy between organ-matrices and organs in live nature. Organ-matrices are (relatively) self-sufficient as organs are. Organs are tied to a body but they can be replaced by other organs, which contain body (data) and functions (operators).

An organ consists of cells of different types. One of the types of cell-matrices is commonly used the 2-dimentional rectangular numeric matrices (n-matrices). These matrices contain numbers with signs \(\{+, -, \} \), which are operators on their own. The types of cells constituting organ-matrices have a much broader base regarding data types and operators than n-matrices.

We showed an example of implementation of organ-matrices for recognition of quotability of financial contracts in the re-engineering project made by the author for one of the biggest non-banking financial company in the world.

Business requires that its rules are constantly changed. Using organ-matrices allows re-distributing business logic between stored procedures, data and operators as data stored in database tables, which make stored procedures relatively simple.

Before the re-engineering of the System, each cycle of accommodation of business requirement took about two months with changes of many stored procedures, testing, etc. After the re-engineering, the same cycle of accommodation of business requirements for changing business logic takes two hours. No changes of stored procedures are required for changing business logic. Any changes are made on the level of data and operators as data stored in database tables.

Organ-matrices allow switching from dealing heavily with (programming) procedures to dealing with data, from complicated procedures to simple ones without losing information about business logic.

Redistribution of business logic between stored procedures and organ-matrices allows building a computerized system, which is optimal upon multiple criteria: speed, changeability and maintainability.

Organ-matrix is a level of abstraction where data and operators as data are treated equally: they can be stored in database tables and manipulated with or without consideration of contents. Organ-matrix can be a subject of analytical research and practical development. Organ-matrices can substitute and/or complement algorithms. Organ-matrices are more fit for changes than stored procedures are. Operators in stored procedures are hard coded and changing them requires programmers to be involved in the changing process. Operators as data can be changed automatically, which open new horizons for artificial intelligent and business systems.