

A Declarative Concept-Based Query Language as a mean for Relational Database Querying

Vladimir Ovchinnikov (ovch@lipetsk.ru), Yuri Vahromeev (vakh_yv@lipetsk.ru)

Abstract. The aim of the paper is to present a tool executing queries of a declarative concept-based query language over a relational database. As the concept-based query language the Semantically Complete Query Language (SCQL) is taken because of its property to be used without referencing to proper relation names. The tool is an additional program layer over a RDBMS of any vendor. The layer takes a conceptual query, maps it to SQL query, executes the SQL query using JDBC, and returns back the result. The tool is controlled by XML-based knowledge base that defines: named SCQL queries, a SCM model, and rules of mapping of the model to relational model. The tool is the part of a data integration system based on Semantically Complete Model (SCM).

1 Introduction

Conceptual query language is a query language built over a conceptual model. Now there is a row of conceptual query languages such as LISA-D [5], ConQuer [2], RIDL [6], COQL [10], OSM-QL [3]. The enumerated languages are based on conceptual modeling approaches ORM [1, 4], CODM [10], OSM [3]. ConQuer and OSM-QL are graphical query languages, RIDL and COQL are not pure declarative and have procedural aspects.

Concept-based query language is considered as a declarative conceptual query language which allows query formulation by using application domain concepts only. Today there is one pure declarative concept-based query language, Semantically Complete Query Language (SCQL) [7, 9], based on the conceptual model SCM [7]. Other languages use explicit references to relations or roles (ORM), and some of them have procedural aspects.

So, the main properties of SCQL are the following. SCQL is concept-based: queries are formulated using application domain concepts (without using connections' proper names or abbreviations). And queries of SCQL are formulated in declarative way.

The first feature is a consequence of completeness property of SCM: each connection is completely defined with a set of concepts underlying it, and each concept is completely defined with a set of connections in which it participates. Literally, the property means that each connection completely describes interconnection of all concepts underlying it, and a model does not contain connections contradicting each to

other: based on the same set of concepts or on a proper subset of concepts of another connection.

Another property of SCM that is important for SCQL query formulation is simplicity. The core of SCM model is a set of concepts and their connections. It is very close to human way of thinking about things. Therefore, SCM and SCQL allow direct mapping of a mind conceptual model to a formal conceptual model and queries, without introducing artificial additional elements. As a result, intuitive clearness of the model and queries for application domain experts not being specialists in IT (for checking or even developing purposes) is achieved.

These main properties of SCM and SCQL allow to use them as a conceptual data access interface for distributed and heterogeneous information space [7, 8]. The first step on this way is creation of a tool that executes SCQL queries over a RDBMS. This is the issue of the paper.

2 Architecture of SCQL Query Executer

Data transformation during process of SCQL query mapping and execution is shown on the fig. 1. Source data for the process are SCQL query and SCM knowledge base (SCM KB). SCM KB stores definition of SCM model and mapping of SCM model to relational model (RM). Also, the knowledge base can contain named SCQL queries, that can be used as subqueries of other queries by reference (similar to views of RDBMS). The knowledge base has XML format. Output data of the query executer is a flat result set, columns of which are role concepts of the executed SCQL query.

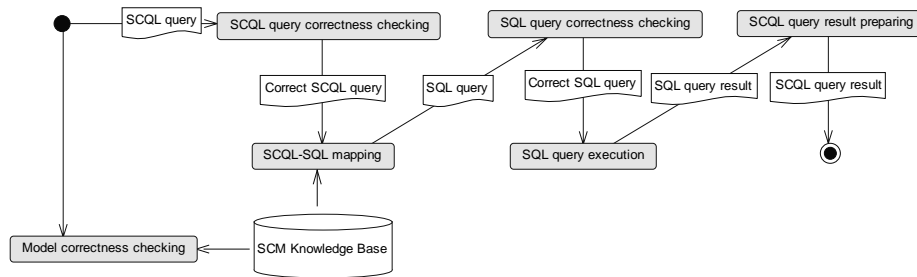


Fig 1. SCQL Query Executer Data Flow Diagram

Main processes of the tool are SCQL-SQL mapping, SQL query execution and SCQL query result preparing. SCQL-SQL mapping is the core of the algorithm. It uses knowledge base for creation of SQL query being equivalent for the initial SCQL query.

The tool includes correctness checking for each involved object: SCM model, SCM-RM mapping, SCQL query, SQL query. Checking of SQL query is mainly used for mapping algorithm testing.

Application of the tool implies the following order of work.

Step 1. Definition of SCM model by means of textual notation. Mapping the textual notation to XML with structure described below.

Step 2a. Creation ER diagram on basis of SCM model, taking decisions about representing concepts and their connections with entities and relationships. This process should be automated in future. Generation relational schema on basis of the created ER diagram. Modifying attribute names to comply with their semantics and role in a table.

Step 2b. Definition of SCM-RM mapping rules for the concrete SCM model and relational schema. This step should be automated in future.

Step 3. Definition of named and anonymous SCQL queries on basis of the SCM model.

Step 4. SCQL queries execution.

SCM knowledge base represents a hierarchy of XML documents located in one or several root directories, specified with SCM_ROOT environment variable. The root element of the XML documents is “SCM”. The full name of the document file, starting from a root directory, is considered as a name of an appropriate application domain. All objects defined in the file are defined within this application domain. Any other application domain can refer to these objects by means of their dot-separated full name.

For example, consider two application domains “Production.Material Unit” and “Production.Indiscrete”. The application domains are described in files:

```
Production\Indiscrete.scm
Production\Material Unit.scm
```

Let the application domain “Production.Material Unit” declare concept “Material Unit”:

```
<concept name="Material Unit"/>
```

To use the concept, the application domain “Production.Indiscrete” imports it using the concept full name:

```
<import>
<conceptimp>Production.Material Unit.Material Unit</conceptimp>
</import>
```

Importing allow to use the concept further by last component of the full name. The application domain may not import the concept and use it by full name each time. Each application domain file is to have the extension “scm”. If an application domain has both nested application domains and objects, then this application domain is to be represented both as a directory and as a “*.scm” file. Application domains and objects of each type are to have unique names within an application domain containing them.

3 Definition of SCM model

Definition of SCM model by means of textual notation is described in [7, 9]. As an example, consider the following short SCM model being the part of a model of a Manufacturing Execution System of an indiscrete production:

```
Material Unit has a Thickness →
Material Unit has a Width →
```

Material Unit has a Mass →
Mass Transition is from a Consumed Material Unit →
Mass Transition is to a Produced Material Unit →
 Mass Transition has a Mass →
Consumed Material Unit is a Material Unit ≡
Produced Material Unit is a Material Unit ≡

Each phrase is a declaration of a connection based on a set of concepts marked with capital first letters. Three types of constraints are used here: the mandatory constraint by underlining, the functional constraint by the symbol ‘→’, and the equivalence constraint by the symbol ‘≡’. For detailed information please refer to [7, 9].

Consider definition of the SCM model in the knowledge base. Any application domain can contain a document part used for model definition with the following DTD:

```

<!DOCUMENT SCM [
  <!ELEMENT connection (conceptref+, copula*, rcopula*)>
    <!ATTLIST connection      equivalent CDATA "false"
                             biequivalent CDATA "false">
  <!ELEMENT concept          (conceptref*)>
    <!ATTLIST concept         name CDATA #REQUIRED>
  <!ELEMENT conceptref (#PCDATA)>
    <!ATTLIST conceptref     mandatory CDATA "false"
                             determinant CDATA "false">
  <!ELEMENT copula           (#PCDATA)>
  <!ELEMENT rcopula         (#PCDATA)>
  <!ELEMENT import          (conceptimp | serverimp | queryimp)+>
  <!ELEMENT conceptimp      (#PCDATA)>...]>
  
```

#PCDATA of conceptref element is a full name of a concept, or its last component if the concept is imported. A full name of a concept is a full name of an appropriate application domain, a dot, plus a name of the concept. Mandatory and functional constraints are defined as Boolean attributes of a conceptref element nested to a connection element. If the attribute “mandatory” has the value “true”, the appropriate concept is considered as mandatory within the connection. If the attribute “determinant” has the value “true”, the appropriate concept is considered to be determinant in the connection’s functional constraint. If the connection has the attribute “equivalent” with the value “true”, the connection is considered to be equivalently constrained on two groups of concepts: one group is of determinant concepts, and another group is of non-determinant concepts.

Copula and rcopula elements contains direct and reverse verbal copulas of a connection textual notation. According to SCM properties [7] there are additional constraints on this DTD part: each connection has at least two concepts; there is no a connection based on concept set being proper subset of concepts of another connection. Nesting of concepts to other concepts, biequivalence of connections, and importing of servers are used for SCM-RM mapping definition and are not essential for SCM model definition per se.

This part of DTD allows to define any SCM model with XML. Let the SCM model described above be defined in three application domains: “Production.Material Unit”, “Production.Indiscrete”, “General.Physical Object”.

Consider the first application domain “Production.Material Unit”. First of all, the application domain is to import used concepts of other application domains. The

concepts “Thickness”, “Width”, and “Mass” are defined in the application domain “General.Physical Object” to share them in various application domains.

```
<scm>
<import>
  <conceptimp>General.Physical Object.Geometry.Thickness</conceptimp>
  <conceptimp>General.Physical Object.Geometry.Width</conceptimp>
  <conceptimp>General.Physical Object.Gravity.Mass</conceptimp></import>
```

Then the concept “Material Unit” contained in the application domain and connections based on it are to be defined:

```
<concept name="Material Unit" />
<connection>
  <conceptref mandatory="true" determinant="true">Material Unit
  </conceptref>
  <copula>has</copula><rcopula>characterizes</rcopula>
  <conceptref>Thickness</conceptref>
</connection>
<connection>
  <conceptref mandatory="true" determinant="true">Material Unit
  </conceptref><conceptref>Width</conceptref>
</connection>
<connection>
  <conceptref mandatory="true" determinant="true">Material Unit
  </conceptref><conceptref>Mass</conceptref>
</connection></scm>
```

Full definition of the SCM model example can be found at [11].

4 Mapping of SCM model to Relational Model

Before defining SCM-RM mapping rules, we need to create a relational model reflecting the SCM model. Fig. 2 presents the ER model created on basis of the SCM model example described above. During ER model creation we have taken some decisions. We have decided to represent our connections with two entities: “Material Unit” and “Mass Transition”. The concept “Material Unit”, functionally determining “Thickness”, “Width”, and “Mass”, becomes the prototype for primary key attribute of the appropriate entity. “Thickness” and “Width” have been grouped to one attribute with the type “GEOMETRY”. “Consumed Material Unit” and “Produced Material Unit” of “Mass Transition” become the prototypes for dependent relationships of the “Mass Transition” entity.

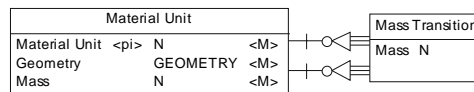


Fig. 2. ER Model of MES Part

Mandatory constraints have been reflected by mandatory constraints of the appropriate attributes, and function constraints have been reflected by primary key constraints and dependent relationships of the ER model. So, the created ER model is equivalent to the initial SCM model. The decisions can be taken automatically from

SCM model structure. The appropriate tool of ER model generation should be created in future.

A relational model generated from the ER model is represented at the fig. 3. Here, the attributes “Consumed Material Unit” and “Produced Material Unit” have been named according to their senses.

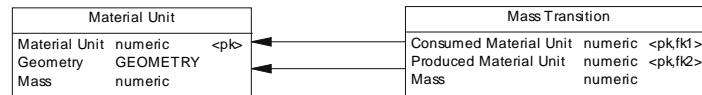


Fig. 3. Relational Model of MES Part

SCM-RM mapping rules are of two categories: connection-to-relation (connection-to-table) mapping rules and attribute-to-concept (field-to-concept) mapping rules. In our example, the following connections are mapped to the table “Material Unit”:

Material Unit has a Thickness →

Material Unit has a Width →

Material Unit has a Mass →

And the rest of connections is mapped to the table “Mass Transition”. Attributes are mapped to concepts according to their names: all the attributes “Mass” are mapped to the concept “Mass”, the attribute “Thickness” is mapped to the attribute field “Geometry.Thickness” (since the attribute “Geometry” is of the object type “GEOMETRY”), etc. Definition of SCM-RM mapping rules in the SCM knowledge base is performed according to the following DTD. This DTD is continuation of the knowledge base DTD started above.

```
<!DOCUMENT SCM [...
  <!ELEMENT table      (field+, connectionref*)>
  <!ATTLIST table     name CDATA #REQUIRED serverref CDATA #REQUIRED>
  <!ELEMENT field      (field | conceptref)*>
  <!ATTLIST field     name CDATA #REQUIRED pk CDATA "false">
  <!ELEMENT connectionref (conceptref)+>
  <!ELEMENT server     (#PCDATA)>
  <!ATTLIST server   name CDATA #REQUIRED
                  connect_string CDATA #REQUIRED>
  <!ELEMENT serverimp (#PCDATA)>... ]>
```

Definition of an underlying relational model and SCM-RM mapping is to be done together. The table element contains both field descriptions and enumeration of reflected connections. If a field is of an object type, it contains its constituent fields, otherwise it contains reference to a reflected concept. Since a concept set identifies a connection unambiguously [7], a connection reference is a set of concept references simply. SCM-RM mapping requires that each connection not being calculated one is to be based on concepts reflected to all primary key fields of the appropriate table. Primary key fields are marked with the “pk” attribute having the value “true”.

Each table is to have a reference to a server holding it. A server is declared with the help of the server element. The element is to have the attributes “name” and “connect_string”. The attribute “name” is used for referencing to the server. To import a server to an application domain, the serverimp element is used. #PCDATA of the element is a server’s full name. For referencing to a server, its name is to be used if it

is imported, and otherwise its full name is to be used. The attribute “connect_string” is a standard JDBC connect string.

In our example, let SCM-RM mapping be defined in the separate application domains “Production Implementation.ISCO_MES.Material Unit” and “Production Implementation.ISCO_MES.Mass Transition”. First of all, a server is to be declared. We declare it in the application domain “Production Implementation.ISCO_MES” to share it between nested application domains.

```
<scm><server name="ISCO_MES" connect_string="" /></scm>
```

Consider the application domain “Production Implementation.ISCO_MES.Material Unit”. Its import part imports the server and the used concepts.

```
<scm><import>
  <serverimp>Production Implementation.ISCO_MES.ISCO_MES</serverimp>
  <conceptimp>General.Physical Object.Geometry.Thickness</conceptimp>...
</import>
```

Then the table “Material Unit” and its mapping to the SCM model must be defined.

```
<table name="MES.MATUNITS" serverref="ISCO_MES">
  <field name="MATUNIT_ID" pk="true">
    <conceptref>Material Unit</conceptref></field>
  <field name="GEOMETRY">
    <field name="THICKNESS">
      <conceptref>Thickness</conceptref></field>
    <field name="WIDTH"><conceptref>Width</conceptref></field></field>
  <connectionref><conceptref>Material Unit</conceptref>
    <conceptref>Thickness</conceptref></connectionref>
  <connectionref><conceptref>Material Unit</conceptref>
    <conceptref>Width</conceptref></connectionref>
  <field name="MASS"><conceptref>Mass</conceptref></field>
  <connectionref><conceptref>Material Unit</conceptref>
    <conceptref>Mass</conceptref></connectionref></table></scm>
```

Full definition of the example of SCM-RM mapping rules can be found at [11].

5 Definition of SCQL queries

The last part of SCM knowledge base DTD serves for creation named SCQL queries on basis of a SCM model. A SCQL query is a tree of selection operations of the types: connection selection, transformation, composition, union, minus, and logical selection. An operation of connection selection is always a tree leaf. It selects data of a connection indicated by a set of concepts. After selecting data, each concept can get a role it plays in the query. So, the result signature of a connection selection is a set of role concepts, where each role concept is a concept plus its role.

A transformation operation serves for changing signature of a subquery. A subquery is a tree of selection operations that is nested to the transformation operation. A signature of the subquery is a signature of its root selection operation. Signature change process can include projection, grouping and role concept calculations at the same time. Actually, each transformation determines a set of result role concepts and the way of their calculation on basis of role concepts of a nested selection operation.

A result role concept can be included immutably, or it can be calculated using an algebraic formula. The last operation of the formula can be an aggregate function. Grouping of data is applied implicitly if at least one aggregate function is used. In this case, grouping is done on those result role concepts that are not used in aggregate functions.

A composition operation is an analogue for a join operation. The main distinction is absence of an explicit join criterion. A composition operation is based on a set of nested selection operations of any type. No any additional information is specified for it. Nested selection operations are composed according to role concepts identity. The result signature does not contain identical role concepts repeatedly.

A union and a minus are analogues for relational union and minus. The main difference is possibility to apply these operations to nested operations having different signatures and arities. The operations are performed not according to column alignment, but according to role concepts identity. So, a union operation is equivalent to relational full outer join in the case of different signatures of nested operations.

A logical selection operation is another leaf operation. This operation can be nested to a composition only and represents a logical predicate. Result data of a composition are filtered according to all logical selections nested to it. A query as a whole, like SQL query, can define ordering of the result data.

The detailed description of SCQL can be found at [9]. The knowledge base part concerning SCQL queries has the following DTD.

```
<!DOCUMENT SCM [...
  <!ELEMENT query      ((connection | composition | transformation |
    union | minus | queryref), order*, natural*, formal*)>
  <!ATTLIST query      name CDATA #REQUIRED>
  <!ELEMENT order      (roleconcept)>
  <!ATTLIST order      type CDATA "ASC">
  <!ELEMENT connection (roleconcept)+>
  <!ELEMENT composition(connection | composition | transformation |
    union | minus | queryref | predicate)+>
  <!ELEMENT transformation(roleconcept+, (connection | composition |
    transformation | union | minus | queryref))>
  <!ELEMENT union      (connection | composition | transformation | union |
    minus | queryref)+>
  <!ELEMENT minus      (connection | composition | transformation | union |
    minus | queryref)+>
  <!ELEMENT queryref   (#PCDATA)>
  <!ELEMENT roleconcept(function | roleconcept)>
  <!ATTLIST roleconcept concept CDATA #IMPLIED role CDATA #IMPLIED>
  <!ELEMENT function   (function | roleconcept)*>
  <!ATTLIST function   type CDATA #IMPLIED constant CDATA #IMPLIED>
  <!ELEMENT predicate  ((function | roleconcept)* | predicate*)>
  <!ATTLIST predicatetype CDATA #REQUIRED >
  <!ELEMENT queryimp   (#PCDATA)>]>
```

SCQL features require imposing additional restrictions on the DTD. A connection element and a composition element are to have at least two nested elements. Role concept elements nested to a connection element cannot have nested elements since they serves for referencing a connection only. A role concept element nested to another role concept element or a function cannot have nested elements since it serves as function variable. Concept attribute of a role concept element is to refer to a declared concept. Union and minus elements must have exactly two nested elements. #PCDATA of a queryref element is to refer to a declared query. Queries can be im-

ported similar to concepts and servers with the queryimp element. We intentionally have separated DTD of the connection element on two parts: used for SCQL query and SCM model definition, to emphasize that application of the elements differs.

Each function element is to have either type or constant attribute. Type of the predicate element is to comply with types of nested elements: comparison operations are applicable to function or role concept elements and logical operations are applicable to predicate elements. Consider an example of SCQL query definition.

```
<query name="Material units thicker 0,5">
  <composition>
    <connection><roleconcept concept="Material Unit"/>
    <roleconcept concept="Thickness"/></connection>
    <predicate type=">"><roleconcept concept="Thickness"/>
    <function constant="0.5"/></predicate></composition></query>
```

In this example, material units having thickness more than 0.5 are selected: (Material Unit, (Thickness>0.5)). The following query uses an aggregate function to calculate summary mass of mass transitions going from each material unit: (Material Unit-Consumed Material Unit-Mass Transition-Mass).(Material Unit, SUM(Mass) ~Mass).

```
<query name="Outcoming material unit mass">
  <transformation><roleconcept concept="Material Unit"/>
  <roleconcept concept="Mass"><function type="SUM">
    <roleconcept concept="Mass"/></function></roleconcept>
  <composition><connection><roleconcept concept="Material Unit"/>
    <roleconcept concept="Consumed Material Unit"/></connection>
  <connection> <roleconcept concept="Consumed Material Unit"/>
    <roleconcept concept="Mass Transition"/></connection>
  <connection> <roleconcept concept="Mass Transition"/>
    <roleconcept concept="Mass"/></connection></composition>
</transformation></query>
```

More queries for the example of SCM model can be found at [11].

6 Application of the Tool in Practice

We have defined SCM knowledge base: SCM model, SCM-RM mapping rules, named SCQL queries. Now we are ready to execute named or anonymous SCQL queries with the tool. The first stage is mapping of a SCQL query to a SQL query. This stage is very complex and is the subject of another paper. Let us illustrate the execution process using the SCQL queries described above.

```
Query: Production Queries.Material Unit.Material units thicker 0,5
SELECT t1."GEOMETRY.THICKNESS" "GEOMETRY.THICKNESS0", t1.MATUNIT_ID
MATUNIT_ID1 FROM (
  SELECT MATUNIT_ID, GEOMETRY.THICKNESS "GEOMETRY.THICKNESS"
  FROM MES.MATUNITS) t1
WHERE t1."GEOMETRY.THICKNESS">0.5
```

```
Query: Production Queries.Material Unit.Outcoming material unit mass
SELECT (SUM(t1.MASS0)) MASS00, t1.CONSUMED_MATUNIT_ID1
CONSUMED_MATUNIT_ID11 FROM (
  SELECT t1.MASS MASS0, t1.CONSUMED_MATUNIT_ID CONSUMED_MATUNIT_ID1,
  t1.CONSUMED_MATUNIT_ID CONSUMED_MATUNIT_ID2, t1.CONSUMED_MATUNIT_ID
  CONSUMED_MATUNIT_ID3, t1.PRODUCED_MATUNIT_ID PRODUCED_MATUNIT_ID4
  FROM (SELECT CONSUMED_MATUNIT_ID, PRODUCED_MATUNIT_ID, MASS
  FROM MES.MASSTRANSES) t1) t1 GROUP BY t1.CONSUMED_MATUNIT_ID1
```

Since the stage is performed automatically, the SQL query looks somewhat more verbose than it can be. Note that the tool modifies field names since SQL subqueries can contain fields with the same name. When object type fields are used, their components have aliases in double quotes as "GEOMETRY.THICKNESS". Each SQL subquery has a signature that is close to signature of an appropriate SCQL subquery. For example, the composition (Material Unit-Consumed Material Unit-Mass Transition-Mass) has the following look in SQL:

```
SELECT t1.MASS MASS0, t1.CONSUMED_MATUNIT_ID CONSUMED_MATUNIT_ID1,
t1.CONSUMED_MATUNIT_ID CONSUMED_MATUNIT_ID2, t1.CONSUMED_MATUNIT_ID
CONSUMED_MATUNIT_ID3, t1.PRODUCED_MATUNIT_ID PRODUCED_MATUNIT_ID4
FROM (SELECT CONSUMED_MATUNIT_ID, PRODUCED_MATUNIT_ID, MASS
FROM MES.MASSTRANSES) t1
```

As you see, each role concept of the SCQL query has an appropriate field. Note that complex concepts such as "Mass Transition" have several fields according to quantity of simple concepts constituting them. The issue of complex and simple concepts is closely connected with SCQL-SQL mapping algorithm and is a subject of another paper. The next stage is execution of the SQL query using standard JDBC mechanism. So, any RDBMS can be powered by the tool. And in the last execution stage SQL query fields are substituted by role concepts.

The tool described in this paper is implemented and its demo version can be downloaded from [11]. It is considered as a part of a data integration system based on SCM and SCQL [7, 8]. The extend application example can be also downloaded from [11]. Theoretical aspects of SCQL, SCM-RM and SCQL-SQL mappings will be discussed in separate papers.

References

1. Bronts G.H.W.M., Brouwer S.J., Martens C.L.J., Proper H.A. A Unifying Object Role Modelling Approach. *Information Systems*, 20(3), 1995, pp. 213-235.
2. Bloesch A.C., Halpin T.A. Conceptual Queries using ConQuer-II // *Proceedings of ER'97: 16-th International Conference on Conceptual Modeling*, 1997.
3. Embley D.W., Wu H.A., Pinkston J.S., Czejdo B. OSM-QL: a calculus-based graphical query language, Report, Dept. of Comp. Science, Brigham Young Univ., Utah, 1996.
4. Halpin T.A. *Conceptual Schema and Relational Database Design*. Prentice-Hall, Sydney, Australia, 2nd edition, 1995.
5. ter Hofstede A.H.M., Proper H.A., van der Weide. Query Formulation as an Information Retrieval Problem // *The Computer Journal*, 39, 1996, pp. 255-274.
6. Meersman R. *The RIDL Conceptual Language*. Research report, International Centre for Information Analysis Services, Brussels, Belgium, 1982.
7. Ovchinnikov V.V. A Conceptual Modeling Technique Based on Semantically Complete Model, its Applications // In (Doroshenko A., Halpin T., Liddle S., Mayr H. eds.) *Proc. of the 3rd International Conference ISTA'2004: Information Systems Technology and its Applications*, Salt Lake City, GI Lecture Notes in Informatics P-48, 2004, pp. 25-38.
8. Ovchinnikov V.V. Architecture of Heterogeneous Concept Space Managed by non-IT people // *Journal of Conceptual Modeling* (www.inconcept.com/jcm), 33, 2004.
9. Ovchinnikov V.V. A Semantically Complete Conceptual Modeling Technique // *Journal of Conceptual Modeling* (www.inconcept.com/jcm), 32, 2004.

10. Savinov A. Principles of the Concept-Oriented Data Model. Institute of Mathematics and Informatics, Academy of Sciences of Moldova, Technical Report (www.conceptoriented.com), 2004, 54pp.
11. SCM portal (scm.lipetsk.ru), 2005.