Unification of Graph Data Models for Heterogeneous Security Information Resources' Integration

Stupnikov Sergey[1], Miloslavskaya Natalia[2], and Budzko Vladimir[1,2]
[1] Institute of Informatics Problems, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences
[2] The National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)
Moscow, Russia

sstupnikov@ipran.ru, NGMiloslavskaya@mephi.ru, VBudzko@ipiran.ru

***Abstract*—Integration of large heterogeneous data collections, gathered usually for making decisions on information security (IS) management issues, requires a preliminary step – unification of their data models. It is provided by mapping the source data models into the canonical information model. Information and semantics of data definition languages (DDL) and semantics of the operations of data manipulation languages (DML) have to be preserved by the mapping. This research is devoted to the unification of the graph data models (GDM) – the important kind of the existing various data models. The distinguishing features of modern GDM are discussed as well as their application in the information security (IS) area. The issues of proof of DDL and DML semantics preserving by the mapping of the GDM into the object-frame canonical model are briefly considered. Future work steps in applying research results to different IS management areas are indicated in conclusion.**

Keywords—graph data models, attributed graphs, data model mapping, security information resources' integration, information security management

I. INTRODUCTION

Significance of *data* has been increasing during last years in different fields: scientific research, health, education, industry, information security (IS) management and others. Modern enterprises of different scope and size collect and store a huge amount of data about the current state of their IT infrastructure (ITI). These data need to be processed correctly and promptly to identify and visualize possible IS threats, vulnerabilities to be eliminated and at least IS incidents occurred, to optimize ITI monitoring strategy and resources, to calculate current and forecast further IS risks and so on. The data are generated from the different types of information considered in a particular context. This information comes not just from the separate domain controllers, proxy servers, DNS servers, information protection tools (IPT). It also describes current configuration of network devices, generalized characteristics of network traffic, application and network services functioning, activity and specific actions of individual end-users, as well as contains e-mails, web-based content, digitized audio and video, the data of business processes, enterprise's internal documents and analytical data for many years of its existence.

Thus a new scientific paradigm – the Fourth Paradigm [1] – in IS management area is strengthening. The paradigm is based on intensive usage of data. Volumes and heterogeneity of IS data and related activity for further scrupulous monitoring and analysis are very high. A problem of structured and consolidated presentation of data to make timely and informed decisions in the field of IS management for all enterprise's ITI assets is one of the biggest IT challenges nowadays. Ever-increasing volumes of data on IS events, ITI assets, their vulnerabilities, users, IS threats, IS risks and related information, as well as the need for more rapid obtaining of systematic and analyzed in a certain way "raw" heterogeneous information for faster understanding of the current situation in IS ensuring for ITI are well-known problems of big data. Hence one of the main issues of the IS management systems in general and security information and event management (SIEM) systems in particular supporting data-intensive science is that volume of data to be analyzed exceeds significantly the capabilities of methods and tools for data storage, processing and analysis.

New IT and approaches for information systems' conceptualization, organization and implementation are evolving. They are characterized by data as the dominating factor. Not only data management methods and tools manipulating data volumes that exceed the capabilities of conventional databases are required, but new approaches allow to handle the diversity of massively and chaotically evolving data models and languages.

The paper continues ongoing research devoted to unification of data models for virtual or materialized data resource integration during development of federated databases or data warehouses as data intensive systems. These data models include various NoSQL models, ontological and semantic models, array models and others.

978-1-4673-8103-1/15 $31.00 © 2015 IEEE
DOI 10.1109/FiCloud.2015.13

During materialized integration it is assumed that a *data warehouse* is created. Data from the resources to be integrated are transformed in a way to conform the warehouse's schema and then loaded into the warehouse.

A possible way to implement virtual integration is creation of *subject mediators* [3][4]. The mediators form an intermediate layer between a user (an application) and heterogeneous information resources. Data from the resources are not materialized in mediators. Access to data is supported using queries expressed in terms of federated schema of a mediator. The queries are rewritten into sets of partial queries over resources. Partial queries are executed on resources. Results of partial queries are transformed into the format supported by the mediator, united and passed to the user. Final united result conforms the federated schema of the mediator.

Unification of a resource data model is mapping of the model into the canonical information model serving as common language in the environment of heterogeneous resource data models [2]. The mapping has to preserve information and semantics of data manipulation language (DML). An object-frame data model – the SYNTHESIS language [3] – is considered in the paper as the canonical one. The SYNTHESIS language is intended to the development of subject mediators and data warehouses for problem solving over heterogeneous IS-related information resources. The language is supported by subject mediation runtime environment [4].

Both materialized and virtual integration paradigms allow integrating heterogeneous resources (containing, for instance, security information) and representing it in a uniform way. Both paradigms have scalability issues. Virtual integration is scalable w.r.t. the number of resources to be integrated because every resource is integrated into a subject mediator independently of the others. The restriction of virtual integration is that results of partial queries should not be very large: the bottleneck is the bandwidth of the network channels between resources and a mediator. Data warehouses in their turn are comparatively scalable w.r.t. size of the resources, intensity of queries and result size: warehouses can be deployed on clusters using, for instance, Hadoop-based solutions. In order to join the advantages of both paradigms combined virtual and materialized integration infrastructures can be provided.

This paper considers graph data models (GDM) constituting the important class of the information models. Investigation of GDM began in the middle of 1980s. The GDM mathematical basis is the graph theory. The aim of GDM was to overcome difficulties related to representation of graph data structures using conventional data models. Significance of GDM is motivated by variety of applications – social networks, recommender systems (aimed at increasing pertinence of the information), geospatial and security applications, etc. Modern graph DBMSs are intended to processing and analysis of large graphs including billions of vertices and edges. For instance, the Facebook network includes 1.3 billion user profiles and 150 billion friendship relationships among users. The Web itself can be considered as a graph. Vertices (pages) are connected by edges (hyperlinks). Leading Internet-companies like Google operate exactly the graph of web-pages.

The paper is organized in the following way. The related work is considered in section II. Modern graph DBMS, data models and their applications in IS management area (for fraud detection, authorization and access control, cyber traffic analysis) are discussed in section III. A synthetic attributed GDM (AGDM) is presented and illustrated by an example in section IV. General principles of AGDM into the SYNTHESIS language mapping are shown and illustrated by some examples in section V. Section VI considers issues of preserving the information and semantics of data manipulating operations by the mapping. Conclusion summarizes the results and discusses future works as well as potential application areas.

II. Related Work

Few publications investigating issues of GDM integration or mapping are known. For instance, in [5] a hypergraph query language is used for definition of views during integration of graph data bases in subject mediators. In [6] a hypergraph data model is also used for integration of graph databases. A set of operations over hypergraphs for resource schema into federated schema transformation are provided. In these papers the model heterogeneity is not an issue: both resource and canonical model are based on the same hypergraph model.

A mapping of the relational data model into a hypergraph data model as well as imperative implementations of relational operations in hypergraph data model are discussed in [7]. So the hypergraph data model is considered as the canonical one and the relational model is considered as a source data model.

Another set of papers considers issues of query containment, query answering and query rewriting using views. State-of-art in this field is presented in [8]. Upper bounds for complexity of query answering and query rewriting using GLAV-views are provided. Decidability of query containment is proven.

The distinguishing features of this paper are the following. The aim of the paper is overcoming the data model heterogeneity of modern graph DBMS for their virtual or materialized integration. As the source model for the mapping a synthetic GDM is used (section IV). Capabilities of the synthetic model covers the capabilities of modern graph DBMS data models based on simple and attributed graphs. As the canonical (target) model for the mapping the object-frame model (the SYHTNESIS language) is used. A technique for formal proof of preserving the information and DML operations by the mapping is provided (section VI).

III. Graph Databases, Graph Data Models and their Applications

The distinguishing features of GDMs are the following [9]:

- data or/and schema of data are represented using graphs or more general structures (hypergraphs, hyperedges);
- data manipulation are represented using graph transformation operations. Parameters of operations include such graph structures and properties as paths, subgraphs, connectivity and others;

- integrity constraints are closely connected with graphs as data structures. Examples of constraints are uniqueness of labels of edges and vertices; typing of edges and vertices; constraints on domains and ranges of edges and vertices' properties.

GDMs are applied when information about data relationships and their topology (in our case let us call them IS management's context) is at least the same significant as data itself. Another reason for applying GDM is a lack of expressiveness of query languages of conventional data models.

GDMs are widely used in systems for management and analysis of complex networks: social, biological, information, transport, telecommunication and others [10]. Due to the limited size of the paper the applications only in IS management area are considered below.

First-party bank fraud involves fraudsters who apply for credit cards, loans, overdrafts and unsecured banking credit lines, with no intention of paying them back [11]. First-party fraud losses are estimated as 10-20% of unsecured bad debt at the leading US and European banks. Typical scenario of a fraud ring operations looks as follows. A group of people is organized into a fraud ring, shares a subset of legitimate contact information (for instance, phone numbers and addresses), combining them to create a number of synthetic identities and open accounts using these synthetic identities. New accounts are applied for unsecured credit lines, credit cards, personal loans, etc. and are used normally, with regular purchases and timely payments. Banks increase the credit lines over time, due to the observed responsible credit behavior. One day the ring "busts out", coordinating their activity, maxing out all of their credit lines, and disappearing. The key point of fraud ring detection is *entity link analysis*, allowing to find out the set of interconnected synthetic identities sharing elements of legitimate data. Graph databases provide a natural way for modeling entity linkage and uncovering rings by traversing the graph in real time.

A typical *insurance fraud* scenario implies a ring of fraudsters working together to stage fake accidents and claim soft tissue injuries. Such rings normally include a number of roles: doctors (diagnosing false injuries), lawyers (filing fraudulent claims), body shops (misrepresenting damage to cars), drivers, passengers, pedestrians, witnesses. Participants are "recycled" to stage many accidents with different roles. A large number of costly fake accidents with a small number of participants can be generated. As in the bank fraud example, finding fraud rings with a graph database becomes a question of traversing the graph and does not require multiple joins of a number of tables as in the relational model.

An *e-Commerce fraud* deals with online transaction with the following identifiers: user ID, IP address, geo location, a tracking cookie, and a credit card number. The relationships between these identifiers are expected to be close to one-to-one. Patterns with a large numbers of users having transactions originating from the same IP, large numbers of shipments to different addresses using the same credit card, or a large numbers of credit cards using the same address can serve as indicating signals of a fraud event. Graph databases allow a real-time discovery of such patterns. By putting checks into place and associating them with the appropriate event triggers, such schemes can be uncovered before they are able to inflict significant damage. Triggers can include events such as login in, placing an order, or registering a credit card.

Applying graph database solutions in *authorization and access control* systems with millions of domestic and business users subscribing to products and services is driven by performance and business responsiveness requirements [10]. Relational database systems of this kind use recursive joins to model complex organizational structures and product hierarchies, and stored procedures to implement the access control business logic. Because of the join-intensive nature of the data model, many of the most important queries are unacceptably slow. Graph databases provide an elegant and efficient way for representation and evaluation of such queries like finding all accessible resources for an administrator, determining whether an administrator has access to a resource or finding administrators for an account.

Cyber traffic analysis (CTA) involves observing and analyzing connections between clients, servers, hosts, and actors within IP networks, over time, to detect suspicious patterns [12]. So called *IPFLOW data* are available from routers and servers which summarize coherent groups of IP packets flowing through the network. The IPFLOW data are naturally represented by massive graphs. An example of cyber-attack scenario which is amenable to treatment in IPFLOW graph is called the *Exfiltration.* An attacker wishes to transmit a large volume of data from a target to an Internet host (drop box) under the adversary's control. The adversary uses a relatively low bandwidth control channel to issue commands prior to a large data flow from a target host to the drop box. Given flow records at the perimeter of the primary target, defenders look for a pattern of activity where an unusually large amount of data is transmitted by a target host that does not generally transmit large amounts of data to external hosts. Additionally, defenders want to find the low-bandwidth control channel used by the adversary to initiate the outbound large data transfer.

Reviews of modern graph DBMSs [9] show that the most of existing graph databases are based on simple or attributed (property) graphs. In these graphs attributes (properties) are associated with edges and vertices. That is why the AGDM are chosen in this paper as the source data models to be unified. Attribute graph databases include Neo4j [13], Dex, InfiniteGraph, OrientDB, VertexDB, Filament, OQGraph, Horton, InfoGrid and others. The mentioned systems are intended to large distributed graph processing using various distributed infrastructures. For instance, widely used Neo4j database [13], utilizes a cluster "master-slave" architecture. The distinguishing feature of the architecture is the replication of a graph on every node of the cluster. At present the Neo4j supports graphs with ten billion vertices and edges and is admitted as the most effective system among the competitors.

Note that in this paper RDF-based systems are not considered. Although RDF is often treated as a graph data model, we investigate RDF unification issues separately from

other graph models due to the wide application area of the RDF, specific DDL, DML and semantics.

IV. ATTRIBUTED GRAPH DATA MODEL

To provide the generality of the approach this paper considers a synthetic AGDM. Data structures of the model cover the capabilities of such systems as Neo4j [13], Dex, InfiniteGraph, OrientDB, VertexDB, Filament, OQGraph, Horton, InfoGrid.

The declarative Cypher language [13] is considered as the DML of the synthetic model. The language is developed by Neo4j, so the given GDM is in fact an extension of the Neo4j data model. The Cypher language is quite general and includes capabilities for vertices and edges adjacency, fixed-length paths reachability, regular simple paths, shortest path search, pattern matching [9]. So the synthetic model is chosen in a way to provide a possibility of application the methods of its mapping into the canonical model (section 5) for unification of various graph data models supported by real graph DBMSs mentioned earlier.

A database in the synthetic model is a graph with *typed* edges and vertices. A type of an edge or a vertex is a set of attributes (properties) associated with the edge or the vertex. The set *VertexTypes* of possible vertex types and the set *EdgeTypes* of possible edge types are formally defined below.

The *VertexTypes* is a set of triples like *<id, name, A>*. Here *id* is a type identifier (for instance, an integer number), *name* is a type name (character string), *A* is a subset of the set *Attributes* of possible attributes.

The *Attributes* is a set of triples like *<id, name, type>*. Here *id* and *name* are attribute identifier and attribute name respectively, $type \in B$ is an attribute type, *B* is the set of built-in-types (like *boolean, int, float, string,* array type and others).

The *EdgeTypes* is a set of tuples like *<id, name, A, directed, restricted, head, tail>*. Here *id* is a type identifier; *name* is a type name; $A \subseteq Attributes$; $directed \in \{true, false\}$ is the indicator whether an edge is directed or not; $restricted \in \{true, false\}$ is the indicator whether types of vertices connected by an edge are restricted or not; $head \in VertexTypes$ is the type of the output vertex of the edge; $tail \in VertexTypes$ is the type of the input vertex of the edge.

For all types $T \in EdgeTypes$ if $restricted(T) = true$ then *head(T)* and *tail(T)* are defined and undefined otherwise.

A schema of the synthetic AGDM $S = <VT(S), ET(S)>$ includes two sets: the set $VT(S) \subseteq VertexTypes$ of types of vertices and the set $ET(S) \subseteq EdgeTypes$ of types of edges.

A database (a graph) *G* conforming the schema *S* looks like $G = <V, E>$. Here

- $V = \{v \mid \exists T.(T \in VT(S) \ \& \ v: T)\}$ is a set of vertices such that a vertex is of type belonging to *VT(S)*;
- $E = \{<e, t, h> \mid \exists T.(T \in ET(S) \ \& \ e: T \ \& \ t \in V \ \& \ t: tail(T) \ \& \ h \in V \ \& \ h: head(T))\}$ is a set of edges such that an edge is of type belonging to *ET(S)* and connects a pair of vertices belonging to *V*.

Typing *x: T* means that for a vertex (an edge) *x* values of attributes belonging to *A(T)* (attributes constituting the type *T*) can be defined.

Let us consider an example of a graph database schema *Cinema* [13] conforming the synthetic AGDM. The schema describes a movie database. To make the schema simpler the type and attribute identifiers are omitted. It is supposed that names identify types and attributes in an unique way:

```
VT(Cinema) = { people, movie }
ET(Cinema) = { cast, directs }
A(movie) = {<id, long>, <title, string>,
    <year, integer>}
A(people) = {<id, long>, <name, string>}
cast = <{<character, string>}, false, false,
    undefined, undefined>
directs = <∅, true, true, people, movie>
```

The schema includes two types of vertices (*people, movie*) and two types of edges (*cast, directs*). The *movie* type includes three attributes (*id, title, year*), the *people* type includes two attributes (*id, name*), the *cast* type includes one attribute (*character*), the *directs* type includes no attributes. Edges of the *cast* type are undirected; types of vertices connected by these edges are not restricted. Edges of the *directs* type are directed from vertices of the *people* type to vertices of the *movie* type.

An example of a graph *g* conforming the *Cinema* schema looks as follows:

```
g = <{m, p}, {e}>
m: movie = <id: 1,
    title: "Lost in Translation", year: 2003>
p: people = <id: 2,
    name: "Scarlett Johansson">
e: cast = <<character: "Charlotte">, m, p>
```

V. MAPPING OF THE SYNTHETIC AGDM INTO THE CANONICAL INFORMATION MODEL

As the canonical information model the object model of the SYNTHESIS language [3] is considered in this paper. Object models served well during unification of different kinds of models, namely structured models, ontological, service, and process models [15][16][17]. Thus the choice of canonical object models for integration of heterogeneous information resources is motivated. GDM are considered as a kind of source data models to be integrated.

Alongside with the SYNTHESIS language, some other object and object-relational models like ODMG and SQL:2011 can be used as the canonical model. In such case the proposed methods for data model unification can be applied as well.

Object models are applicable for conceptual modeling and querying in virtual integration systems. In this case there is no need to transform legacy data into an object model or navigate through an object database – object model queries are rewritten into the resource native model queries and executed in resources. Materialized integration systems (data warehouses) in their turn can apply object-relational models like SQL:2011.

To save space the whole formal mapping of the AGDM's DDL and DML into the object model is omitted in this section. The mapping is illustrated by examples and basic principles of the mapping are summarized at the end of subsections V.A and V.B.

A. Mapping of Data Definition Language

A schema conforming the synthetic GDM is represented in the SYNTHESIS language as a module of the same name. The module includes classes containing vertices (the *vertices* class) and edges (the *edges* class) of a graph as objects. The module representing the *Cinema* schema (section 4) looks as follows:

```
{ Cinema; in: module;
  { vertices; in: class; … },
  { edges: in: class; … };
   …
}
```

A type of vertices (for instance, *movie*) is represented in the SYNTHESIS language as a class of the same name. The class is defined as a subclass of the *vertices* class:

```
{ movie; in: class; superclass: vertices;
  instance_type: {
    id: long;
    title: string;
    year: integer; };
}
```

Attributes of a type of vertices are represented as attributes of the instance type (*instance_type*) of the respective class. A bijection between the sets of built-in-types of the synthetic GDM (for instance, *long, string, int* etc.) and of the SYNTHESIS language is established.

A type of edges (for instance, *directs*) is also represented in the SYNTHESIS language as a class of the same name. The class is defined as a subclass of the *edges* class:

```
{ directs; in: class; superclass: edges;
  instance_type: {
    metaframe
      directed: true;
      restricted: true;
      startVertexType: people;
      endVertexType: movie;
    end
    edgeConstr: {in: invariant;
      {{ all e/directs.inst (directs(e) ->
            people(e.startVertex) &
            movie(e.endVertex)) }}
    }; };
}
```

Attributes of a type of edges are represented as attributes of instance type of the respective class.

Note that the information about directivity of edges (*directed*), restrictedness of input and output vertices types (*restricted*), type of output vertex of an edge (*startVertexType*), type of input vertex of an edge (*endVertexType*) is represented by a specific construct of the SYNTHESIS language called *metaframe* [3]. The mentioned metaframe is associated with the instance type of a class. Metaframes are intended to define additional metainformation associated with modules, types, classes, functions.

Besides the metaframe, the type constraint for input and output vertices is represented by the *edgeConstr* invariant. The invariant is defined as a formula of typed first order logic. In the formulae above and below the *all* keyword is the universal quantifier, -> is the logical implication, & is the conjunction, *x/T* means typing of the variable *x* with the type *T*, *C.inst* means the instance type of the *C* class. The predicate *C(x)*, where *C* is the class name, turns into true on instances of the *C* class.

Note that for the variable *e* of type *directs.inst* the attributes *startVertex* and *endVertex* are defined. These attributes do not belong directly to *directs.inst* type. They are common for all types of vertices and belong to the instance type of the *edges* class:

```
{ edges; in: class;
  instance_section: {
    startVertex: vertices.inst;
    endVertex: vertices.inst;
    isValidEdge: { in: predicate;
      params: {+stVtx/vertices.inst,
               +endVtx/vertices.inst,
               returns/Boolean };
      {{ (stVtx = this.startVertex &
          endVtx = this.endVertex ->
          returns = true) &
          ((stVtx <> this.startVertex |
           endVtx <> this.endVertex) ->
           returns = false) }}
    };
}
```

The *edges.inst* type includes also a predicate method *isValidEdge*. The adjacency predicate *e.isValidEdge(v1, v2)* turns into true if output vertex of the *e* edge (*e.startVertex*) equals to *v1,* and input vertex of the edge *e* (*e.endVertex*) equal to *v2*. The method is specified by the first order formula binding input and output parameters of the method. Symbol '|' means disjunction, predicate '<>' means inequality, keyword *this* denotes the object the method is called for.

The basic principles of AGDM schema into object schema mapping illustrated by the example shown above are the following:

- a schema of the synthetic AGDM $S = <VT(S), ET(S)>$ is mapped into the module of the same name containing the *vertices* class and *edges* class;
- every vertex type $v \in VT(S)$ is mapped into a class of the same name belonging to the module; the class is defined as a subclass of the *vertices* class;
- every edge type $e \in ET(S)$ is mapped into a class of the same name belonging to the module; the class is defined as a subclass of the *edges* class;
- instance type of the *edges* class contains attributes *startVertex*, *endVertex* and a predicate method *isValidEdge* representing vertex and edge adjacency relation;
- information about directivity of edges, restrictedness of input and output vertices types, types of input and output vertices of an edge is mapped into the

metaframe of the instance type of the edge class. Type constraint for input and output vertices is also mapped into invariant defined as a formula of typed first order logic;

- attributes of vertex and edge types are mapped into attributes of the instance types of the respective classes.

Note that schema transformation intended to implement the mapping illustrated above is one-time process for every resource (graph database) both during materialized and virtual integration. The complexity of transformation is linear w.r.t. the size of a schema.

B. DML Mapping

During the integration of heterogeneous resources like databases, services etc. a mapping of the source data definition language (DDL) into the canonical DDL is required. For the DMLs the mapping is reversed: queries to the integration system defined in the canonical model have to be transformed into the queries to the resources.

The SYNTHESIS query (program) language is a Datalog-like language in an object environment. A program is a set of conjunctive queries (rules) of the following kind:

```
q(x/T) :- C1(x1/T1), … , Cn(xn/Tn),
          F1(X1, Y1), … Fm(Xm, Ym), B.
```

The body of the query is a conjunction of collection predicates, functional predicates and a constraint. Here C_i are collection (class) names, F_j are function names, x_i are variables, T_i are variables types, X_j and Y_j are input and output parameters of functions respectively, B is a constraint over x_i, X_j, Y_j. Values of x_i run through objects of C_i respectively.

A collection predicates like *movie([title, year])* are used below. Informally this notation means that we are not interested in whole objects of the *movie* class, but only in the attributes *title, year* of the objects. Formally, *movie([title, year])* is a shorthand of *movie(_/movie.inst[title, year]).* Here _ is the anonymous variable, *movie.inst* is the anonymous instance type of the *movie* class, *title* and *year* are the required attributes of the instance type. A predicate like *source([i, j, val1/val])* denotes renaming of the *val* attribute into *val1*.

Mapping of some basic DML constructions is illustrated by an example of conjunctive query. The query uses the adjacency predicate *isValidEdge* (subsection 5.1). The query returns names of actors with the last name *Cruz,* who play in a movie alongside with *Scarlett Johansson*:

```
q([colleague_name]) :-
  people(scarlett/[name]),
  movies(m),
  people(colleague/[name]),
  cast(c1), cast(c2),
  c1.isValidEdge(m, scarlett),
  c2.isValidEdge(m, colleague),
  scarlett.name = "Scarlett Johansson",
  colleague.name.like("*Cruz*").
```

The query returns a nonempty result set if the database contains such movies *m*, and such edges *c1, c2* of type *cast,* that *c1* connects *m* with *scarlett*, and *c2* connects *m* with *colleague.*

The query mapped into the Cypher language looks as follows:

```
MATCH
  (m: movies)-[c1:cast]-(scarlett: people),
  (m)-[c2:cast]-(colleague: people)
WHERE
  scarlett.name = 'Scarlett Johansson' AND
  colleague.name =~ /*Cruz*/
RETURN colleague.name
```

Every query of the Cypher language is a pattern for search in a graph database.

The MATCH section contains a search pattern. In the query considered above the pattern contains the instruction to search for movies *m* with *scarlett* as an actor and to search for the actors played in the same movie (*colleague*). The WHERE section contains a constraint for the search. In the query considered above the name of the colleague is constrained to contain *Cruz* substring. The RESULT section contains a pattern for the result to be returned. In the query considered above the return pattern is just the full name of an actor.

The basic principles of conjunctive queries of the object model mapping into the Cypher language illustrated by the example shown above are the following:

- a conjunctive query is represented in the Cypher language by a query returning a result (the RETURN section);
- collection predicates and adjacency predicates are represented by patterns of the MATCH section. Every adjacency predicate is represented by its own pattern. Variables typed in predicate collections are represented by variables of the same name used in patterns;
- constraint predicates are represented by the respective predicates of the WHERE section;
- attributes specified in the head of the conjunctive query are represented by the attributes of the same name specified in the RETURN section.

The problem of DML (query) transformation arises during usage of systems for virtual integration (such as subject mediators). It is easy to see that the complexity of query transformation intended to implement the mapping illustrated above is linear w.r.t. the size of a query.

VI. Preserving DML Information and Operations' Semantics by Mapping

This section considers issues of preserving the information and semantics of data manipulating operations by mapping of GDM into object one. It is proposed to prove the such preservation using the Abstract Machine Notation (AMN) formal specification language [18][19]. The method was first proposed and applied for unification of an array data model in [20].

The AMN language is based on the set theory and typed first order logic. Specifications of the language are called *abstract machines*. They combine state space and behavior defined by operations changing the state. A specific relationship between specifications called *refinement* [18] is formally defined.

The main idea of the method is the following. Consider a source data model *S* and a target data model *T*. Construct a mapping θ of *S* into *T*. Define semantics of the data models as AMN abstract machines M_S and M_T respectively. Data structures of the models are represented by variables, data structure properties are represented by invariants (first order formulae), data model DML operations are represented by operations of abstract machines. Examples of source and target data model DML operations are typical generic queries of the SYNTHEIS and Cypher languages respectively.

Corresponding operations of source and target models have to be bound by the DML mapping. The DDL mapping is represented by *refinement invariant* – a closed formula binding states of M_S and M_T abstract machines.

The mapping θ *preserves information and semantics of DML operations*, if M_S abstract machine corresponding with the source model *refines* the M_T abstract machine corresponding with the target model [20]. The refinement is proven using automatic and interactive provers [19].

Due to the limited size of the paper only general principles of GDM semantics representation in AMN are provided below. Principles of the SYNTHESIS language semantics representation in AMN are provided in [18]. Partial semantics of the SYNTHESIS language is represented in AMN by the REFINEMENT specification (a sort of abstract machine) [18] called *ObjectDM*. Semantics of the synthetic GDM is represented in AMN by the REFINEMENT specification called *GraphDM*. Abstract variables of the specification correspond to vertex and edge types identifiers, attributes identifiers, types and attributes names, edges directivity, types of attributes, types of input and output vertices of edges, vertices and edges constituting the graph database, input and output vertices of edges, attribute values. All variables are typed in the invariant of the specification by means of a first order formulae. Graph manipulation operations (for instance, vertex deletion) and typical generic queries are represented as operations of the specification.

To prove formally that the *GraphDM* specification refines the *ObjectDM* specification the refinement invariant was constructed. The refinement invariant binds variables of the *GraphDM* specification and the *ObjectDM* specification. The invariant formalizes the principles of the DDL mapping presented in subsection 5.1 and combines them into a conjunction.

The *ObjectDM* and *GraphDM* abstract machines complemented with the refinement invariant were uploaded into the Atelier B toolkit [19]. A set of theorems formalizing the refinement were generated automatically. For instance, refinement of the vertex deletion operation was formalized by the set of 15 theorems. All of them were proven automatically.

VII. CONCLUSION

The paper analyzes the main features of GDM and several application of the graph DBMSs in IS management area. A synthetic AGDM is considered to provide the generality of the GDM unification approach. Data structures of the model cover the capabilities of such systems as Neo4j, Dex, InfiniteGraph, OrientDB, VertexDB, Filament, OQGraph, Horton, InfoGrid. As the DML of the synthetic model the declarative Cypher language developed by Neo4j is presented. General principles of DDL and DML of AGDM into the canonical information model (the object-frame SYNTHESIS language) mapping are discussed and illustrated by some examples. Issues of preserving the information and semantics of data manipulating operations by the mapping using AMN formal specifications are briefly shown.

Future work includes the following steps:

- selection of specific GDM based on simple and attributed graphs and building of transformations implementing the foregoing mapping;
- extension of tools supporting the subject mediators for the virtual integration of graph databases;
- application of the subject mediators' technology for solving specific problems in an IS management domain over a set of heterogeneous resources including graph databases.

Among these potential application areas are the following:

- network attacks recognition, anomalous activity detection, detection of unauthorized (fraud) operations with information and control over the unreliable employees;
- information stream analysis, tracing of the network packets moving through the communication channels, finding the ways of computer viruses dissemination or botnets functioning; access control to all ITI's resources and ITI's vulnerabilities detection;
- examination of the malware code or virus detection and extraction of the network attacks' signatures in a large volume of information from the affected systems;
- system configurations control, analysis of the IPTs' settings effectiveness, the study of the interaction of the various IS maintenance technologies and separate IPTs;
- correlation of IS events detected and some IS events "view" for dedicated communication channels, network devices, network protocols, services, applications, etc.;
- establishing some trends (that is possible only on the basis of big data usage), modeling and development of IS maintenance rules (for example, to configure a firewall or IDS/IPS);
- allocation of priorities requiring corresponding immediate actions to improve IS management for the whole enterprise's ITI;

- visualization of a computer crime's evidence for further investigation, etc.

ACKNOWLEDGMENT

This research has been done under the support of the Ministry of Education and Science of the Russian Federation (project's unique identifier RFMEFI60414X0139RFBR).

REFERENCES

[1] The Fourth Paradigm: Data-Intensive Scientific Discovery. Eds. Tony Hey, Stewart Tansley, and Kristin Tolle. Redmond: Microsoft Research, 2009.

[2] Kalinichenko L.A., Stupnikov S.A. Constructing of Mappings of Heterogeneous Information Models into the Canonical Models of Integrated Information Systems. Advances in Databases and Information Systems: Proc. of the 12th East-European Conference. Pori: Tampere University of Technology, 2008. – Pp. 106-122.

[3] Kalinichenko L.A., Stupnikov S.A., Martynov D.O. SYNTHESIS: a Language for Canonical Information Modeling and Mediator Definition for Problem Solving in Heterogeneous Information Resource Environments. Moscow: IPI RAN, 2007. 171 p.

[4] Briukhov D., Kalinichenko L., Martynov D., Skvortsov N., Stupnikov S., Vovchenko A., Zakharov V., Zhelenkova O. Application driven mediation middleware of the Russian virtual observatory for scientific problem solving over multiple heterogeneous distributed information resources. Scientific Information for Society – from Today to the Future: Proc. of the 21st CODATA Conference. 2009. Pp. 80-85.

[5] Theodoratos D. Semantic Integration and Querying of Heterogeneous Data Sources Using a Hypergraph Data Model: BNCOD 2002. Lecture Notes in Computer Science. Vol. 2405. Springer, 2002. Pp. 166-182.

[6] Sundaresan S., Hu G: Schema integration of distributed databases using hyper-graph data model: Proc. of Information Reuse and Integration Conf, IRI 2005. Pp. 548-553. ISBN: 0-7803-9093-8.

[7] Tahat A., Ling M.H.T. Mapping Relational Operations onto Hypergraph Model: Proc. of CoRR 2011. The Python Papers, 2011. Vol. 6 Iss. 1. P. 1.

[8] Calvanese D., Giacomo G., Lenzerini M., Vardi M. Query Processing under GLAV Mappings for Relational and Graph Databases: Proc. of the VLDB Endowment. 2012. Vol. 6, No. 2. Pp. 61-72.

[9] R. Angles. A Comparison of Current Graph Database Models: Proc. of IEEE 28th International Conference on Data Engineering Workshops (ICDEW). 2012. Pp. 171-177.

[10] Robinson I., Webber J., Eifrem E. Graph Databases. O'Reilly Media, 2013. 212 p.

[11] Sadowski G., Rathle P. Fraud Detection: Discovering Connections with Graph Databases. Neo Trechnology, 2014. URL: http://goo.gl/cMgJ8r (access date 22.03.2015).

[12] Joslyn C., Choudhury S., Haglin D., Howe B., Nickless B, Olsen B. Massive Scale Cyber Traffic Analysis: A Driver for Graph Database Research: Proc. of the First International Workshop on Graph Data Management Experience and Systems GRADES 2013. ACM Digital Library, 2013. URL: http://goo.gl/q7df68 (access date 22.03.2015).

[13] The Neo4j Manual. 2013. URL: http://docs.neo4j.org/ (access date 22.03.2015).

[14] Jouili S., Vansteenberghe V. An Empirical Comparison of Graph Databases. International Conference on Social Computing (SocialCom). 2013. Pp. 708-715.

[15] Briukhov D.O., Kalinichenko L.A., Tyurin I.N. Extension of Compositional Information Systems Development for the Web Services Platform. Advances in Databases and Information Systems: Proc. of the 7th East European Conference. LNCS 2798. Berlin-Heidelberg: Springer-Verlag, 2003. Pp. 16-29.

[16] Kalinichenko L.A., Stupnikov S.A., Zemtsov N.A. Extensible Canonical Process Model Synthesis Applying Formal Interpretation. Advances in Databases and Information Systems: Proc. of the 9th East European Conference. LNCS 3631. Berlin-Heidelberg: Springer-Verlag, 2005. Pp. 183-198.

[17] Kalinichenko L. A., Stupnikov S. A. OWL as Yet Another Data Model to be Integrated. Advances in Databases and Information Systems: Proc. II of the 15th East-European Conference. Vienna: Austrian Computer Society, 2011. Pp. 178-189.

[18] Abrial J.-R. The B-Book: Assigning Programs to Meanings. Cambridge: Cambridge University Press, 1996.

[19] Atelier B. The industrial tool to efficiently deploy the B Method. URL: http://www.atelierb.eu/index-en.php (access date 22.03.2015).

[20] Stupnikov S. A. Unification of an Array Data Model for the Integration of Heterogeneous Information Resources: Proc. of the 14th Russian Conference on Digital Libraries RCDL'2012. CEUR Workshop Proceedings, Vol. 934. Pp. 42-52.