Using the metagraph approach for addressing RDF knowledge representation limitations (draft version)

Valeriy Chernenkiy, Yuriy Gapanyuk, Anatoly Nardid,
Maria Skvortsova, Anton Gushcha, Yuriy Fedorenko
Informatics and Control Systems
Bauman Moscow State Technical University, BMSTU
Moscow, Russia

Abstract — **This paper proposes an approach for addressing Resource Description Framework (RDF) knowledge representation limitations using an emergent graph model. The metagraph model is proposed to solve this problem. The formal definition of the metagraph model is given. Metagraph and hypergraph models are compared and it is shown that the hypergraph model does not implement the emergence principle. Textual predicate representation of the metagraph model is given covering all the main elements of the metagraph data model. RDF reification limitation and N-ary relationship limitation are examined. It is shown that the metagraph model addresses RDF limitations in a natural way without emergence loss. Proposed textual representation of the metagraph model allows clear and emergent description of the examined problems.**

Keywords — Knowledge representation, RDF, Flat graph model, Hypergraph model, Metagraph model

I. INTRODUCTION

There is no doubt that knowledge representation is one of the most important tasks in AI and software development. This is in particular due to the fact that modern software has become more and more complex combining elements of AI, Internet technologies and other emerging technologies. Using poor or limited knowledge representation models may significantly increase the complexity of information system algorithms, the size of the program code, and the complexity and size of the database. Thus knowledge representation model optimization still remains an important research task.

Graph models are one of the central points in knowledge representation. In the field of web-oriented information systems it is possible to reveal two approaches in graph model knowledge representation: a) using a semantic web approach and b) using a general graph technologies approach.

In the case of the semantic web approach, Resource Description Framework (RDF) is used as the data model, and SPARQL is used as the query language. RDFS (RDF Schema) and OWL (OWL2) are used as ontology definition languages, built on the base of RDF. Using RDFS and OWL, it is possible to express various relationships between ontology elements (class, subclass, equivalent class, etc.) [1]. For RDF persisting and SPARQL processing, special storage systems are used e.g. Apache Jena.

Semantic web technologies are brought to the level of industrial technologies and are used in a number of information systems. But this approach has several limitations. The first limitation is that the RDF data model consists of very small data items – "subject-predicate-object" triples. As a result an average sized relational database may correspond to a triple store containing billions of triples. The second limitation is the N-ary relation limitation [2]. This limitation is that the RDF model does not allow simple ways to describe N-ary relations between vertices of the semantic graph, which complicates the description of complex situations in the semantic graph.

Because of these limitations, the semantic web approach uses a general graph technologies approach based on graph databases. Graph databases use the flat graph or multigraph data model. Usually SQL-like languages with graph extensions are used as query languages. The best-known example of such a database is Neo4j.

Nowadays, there is a tendency to complicate the graph database data model. An example of this tendency is the HypergraphDB database that is the component of the OpenCog AI project. As the name implies, HypergraphDB uses the hypergraph as data model.

Later in this paper we will show that the hypergraph model gives limited benefits compared to the flat graph model for complex situations description. Thus, we propose the use of the metagraph model for this purpose.

II. THEORETICAL BACKGROUND

In this section we will formally define the metagraph model, and compare it with the hypergraph model. We also propose a textual representation of metagraph model.

A. The brief description of metagraph model

A metagraph is a kind of complex network model, proposed by A. Basu and R. Blanning [3] and then adapted for information systems description by the present authors [4]. According to [4]:

$$MG = \langle V, MV, E \rangle,$$

where *MG* – metagraph; *V* – set of metagraph vertices; *MV* – set of metagraph metavertices; E – set of metagraph edges.

A metagraph vertex is described by the set of attributes:

$$v_i = \{atr_k\}, v_i \in V,$$

where v_i – metagraph vertex; atr_k – attribute.

A metagraph edge is described by the set of attributes, the source and destination vertices and edge direction flag:

$$e_i = \langle v_S, v_E, eo, \{atr_k\} \rangle, e_i \in E, eo = true \mid false,$$

where e_i – metagraph edge; v_S – source vertex (metavertex) of the edge; v_E – destination vertex (metavertex) of the edge; e_o – edge direction flag (e_o=*true* – directed edge, e_o=*false* – undirected edge); atr_k – attribute.

The metagraph fragment:

$$MG_i = \{ev_j\}, ev_j \in (V \cup E \cup MV),$$

where MG_i – metagraph fragment; ev_j – an element that belongs to union of vertices, edges and metavertices.

The metagraph metavertex:

$$mv_i = \langle \{atr_k\}, MG_j \rangle, mv_i \in MV,$$

where mv_i – metagraph metavertex belongs to set of metagraph metavertices *MV*; atr_k – attribute, MG_j – metagraph fragment.

Thus a metavertex in addition to the attributes includes a fragment of the metagraph. The presence of private attributes and connections for a metavertex is distinguishing feature of a metagraph. It makes the definition of metagraph holonic – a metavertex may include a number of lower level elements and in turn may be included in a number of higher level elements.

From the general system theory point of view, a metavertex is a special case of the manifestation of the emergence principle, which means that a metavertex with its private attributes and connections becomes a whole that cannot be separated into its component parts.

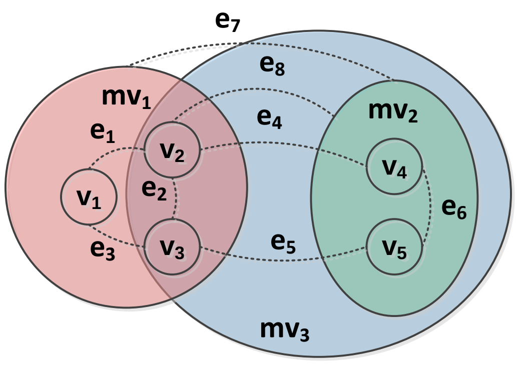


Fig. 1. Example of metagraph

The example of metagraph (shown in fig. 1) contains three metavertices: mv$_1$, mv$_2$ and mv$_3$. Metavertex mv$_1$ contains vertices v$_1$, v$_2$, v$_3$ and connecting them edges e$_1$, e$_2$, e$_3$. Metavertex mv$_2$ contains vertices v$_4$, v$_5$ and connecting them edge e$_6$. Edges e$_4$, e$_5$ are examples of edges connecting vertices v$_2$-v$_4$ and v$_3$-v$_5$ respectively, and are contained in different metavertices mv$_1$ and mv$_2$. Edge e$_7$ is an example of an edge connecting metavertices mv$_1$ and mv$_2$. Edge e$_8$ is an example of an edge connecting vertex v$_2$ and metavertex mv$_2$. Metavertex mv$_3$ contains metavertex mv$_2$, vertices v$_2$, v$_3$ and edge e$_2$ from metavertex mv$_1$ and also edges e$_4$, e$_5$, e$_8$ showing the holonic nature of the metagraph structure.

B. Metagraph and hypergraph models comparison

In this section we examine the hypergraph model, which is used as the data model in the HypergraphDB database, and we compare it to the metagraph data model. According to [5]:

$$HG = \langle V, HE \rangle, v_i \in V, he_j \in HE,$$

where *HG* – hypergraph; *V* – set of hypergraph vertices; *HE* – set of non-empty subsets of *V* called hyperedges; v_i – hypergraph vertex; he_j – hypergraph hyperedge.

A hypergraph may be directed or undirected. A hyperedge in an undirected hypergraph only includes vertices whereas in a directed hypergraph, a hyperedge defines the order of traversal of vertices.

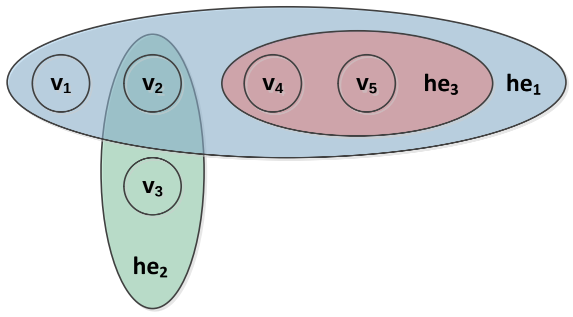


Fig. 2. Example of a hypergraph

The example of an undirected hypergraph (shown in fig. 2) contains thee hyperedges: he$_1$, he$_2$, he$_3$. Hyperedge he$_1$ contains vertices v$_1$, v$_2$, v$_4$, v$_5$. Hyperedge he$_2$ contains vertices v$_2$ and v$_3$. Hyperedge he$_3$ contains vertices v$_4$ and v$_5$. Hyperedges he$_1$ and he$_2$ have a common vertex v$_2$. All vertices of hyperedge he$_3$ are also vertices of hyperedge he$_1$.

Comparing metagraph and hypergraph models it should be noted that the metagraph model is more expressive then the hypergraph model. According to fig. 1 and 2 it is possible to note some similarities between the metagraph metavertex and the hypergraph hyperedge, but the metagraph offers more details and clarity because the metavertex explicitly defines metavertices, vertices and edges inclusion, whereas the hyperedge does not. The inclusion of hyperedge he$_3$ in hyperedge he$_1$ in fig. 2 is informal, because according to hypergraph definition a hyperedge inclusion operation is not explicitly defined.

Thus the metagraph is a holonic graph model whereas the hypergraph is a near flat graph model that does not implement the emergence principle. Therefore, in this paper we will use the metagraph model for knowledge representation.

C. Predicate representation of metagraph model

In section *A*, the formal definition and graphical notation for the metagraph model were defined. However, to operate successfully with the metagraph model we also need a textual representation. For this representation, we use the predicate model, popularized by logical programming languages e.g. Prolog.

Classical Prolog uses the following form of predicate:

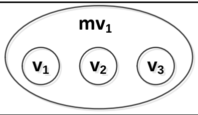
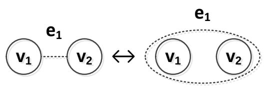
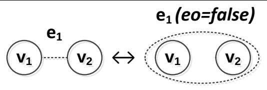
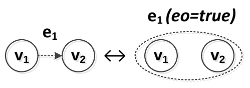
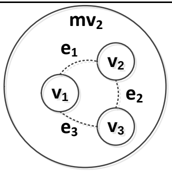
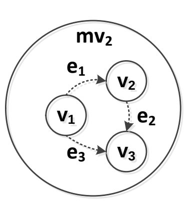
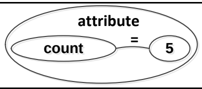
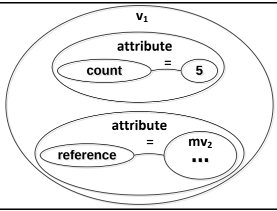
$$predicate(atom_1, atom_2, \ldots, atom_N)$$

We use an extended form of this predicate, where along with atoms, the predicate can also include key-value pairs and nested predicates:

$$predicate(atom, \ldots, key = value, \ldots, predicate(\ldots), \ldots)$$

The mapping of metagraph model fragments into textual representation is shown in table 1.

TABLE 1. TEXTUAL REPRESENTATION OF METAGRAPH MODEL

Case №	Metagraph representation	Textual representation
1		Metavertex(Name=mv_1, v_1, v_2, v_3)
2		Edge(Name=e_1, v_1, v_2)
3		Edge(Name=e_1, v_1, v_2, eo=false)
4		4.1. Edge(Name=e_1, v_1, v_2, eo=true) 4.2. Edge(Name=e_1, v_S=v_1, v_E=v_2, eo=true)
5		Metavertex(Name=mv_2, v_1, v_2, v_3, Edge (Name=e_1, v_1, v_2), Edge(Name=e_2, v_2, v_3), Edge(Name=e_3, v_1, v_3))
6		Metavertex(Name=mv_2, v_1, v_2, v_3, Edge(Name=e_1, v_S=v_1, v_E=v_2, eo=true), Edge(Name=e_2, v_S=v_2, v_E=v_3, eo=true), Edge(Name=e_3, v_S=v_1, v_E=v_3, eo=true))
7		Attribute(count, 5)
8	 v₁ attribute count = 5 attribute reference = mv₂ ...	Vertex(Name=v_1, Attribute(count, 5), Attribute(reference, mv2))

Case 1 shows the example of metavertex mv_1 which contains three nested disjoint vertices v_1, v_2 and v_3. The predicate corresponds to the metavertex, and the nested vertices are isomorphic to atoms that are parameters of the predicate. The predicate name "Metavertex" is used as the corresponding element of the metagraph model. The key-value parameter "Name" is used to set the name of the metavertex. This case is the simplest, since the nested vertices are disjoint, and the metavertex in this case is isomorphic to the hypergraph hyperedge.

Case 2 shows a metagraph edge which may be represented as a special case of a metavertex containing source and destination vertices. This case is also isomorphic to the hypergraph hyperedge. The metagraph edge is represented as a predicate with the name "Edge". The source and destination vertices are represented as predicate atom parameters.

Case 3 also shows metagraph edge which fully complies with the formal definition of undirected edge including direction flag parameter.

Case 4 shows an example of a directed edge. The direction flag parameter is also used. The source and destination vertices may be represented as predicate atom parameters (case 4.1) or as predicate key-value parameters (case 4.2).

Case 5 shows example of metavertex mv_2 which contains three nested vertices v_1, v_2 and v_3 joined with undirected edges e_1, e_2 and e_3. Edges are represented with separate predicates that are nested to the metavertex predicate. Case 6 is similar to case 5, but edges e_1, e_2 and e_3 are directed.

An attribute may be represented as a special case of a metavertex containing name and value only. Case 7 shows a simple numeric attribute representation. Case 8 shows an example of vertex v_1 containing a numeric attribute and a reference attribute that refers to metavertex mv_2. The attribute is represented as a predicate with the name "Attribute".

Thus we have defined a predicate representation of all the main elements of a metagraph data model. In the next section we will use this representation for real-world example definitions.

III. EXAMPLES

In this section we will apply our proposed model to address such RDF limitations as reification limitation and N-ary relations limitation. Despite the fact that these two limitations are very similar, their differences are recognized in the RDF community. The root of both limitations is the absence of the emergence principle in the flat graph RDF model.

A. Reification limitation example

Reification is used to define RDF statements about other RDF statements. According to the RDF Primer [6]: 'the purpose of reification is to record information about when or where statements were made, who made them, or other similar information (this is sometimes referred to as "provenance" information)'. Thus, reification is considered as an auxiliary technique to "log" provenance information about statements.

RDF contains reified triple construction to describe reification in the following form:

```
StatementID subject predicate object
```

Consider the example of the complex statement: ‘James noted that Paul noted at 4 p.m. that John arrived in London’. In the reified triples form, this example may be represented as follows:

```
1.StatementID_1 John arrived_in London
2.StatementID_2 StatementID_1 has_author Paul
3.StatementID_3 StatementID_1 has_time “4p.m.”
4.StatementID_4 StatementID_2 has_author James
5.StatementID_5 StatementID_3 has_author James
```

In statements 2 and 3, StatementID_1 is used as the subject. Statements 2 and 3 contain provenance information about author and time of statement 1. Statements 4 and 5 contain provenance information about the author of statements 2 and 3.

The RDF graph form of this example is shown in fig. 3.

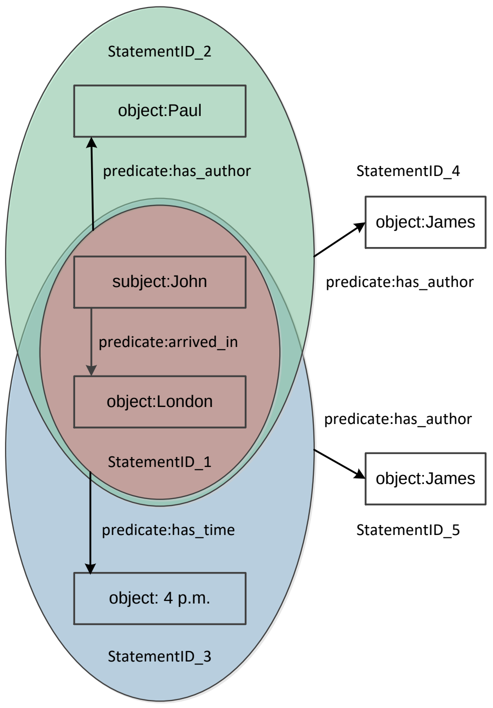


Fig. 3. Example of RDF reification

In fig. 3 statements 1, 2, 3 are highlighted, whereas statements 4 and 5 are not highlighted in order not to confuse visualization of the figure. Fig. 3 shows that a reified triple may be considered as a metavertex but in very restrictive form, containing only one subject, predicate and object.

The problem shown in this example is emergence loss because of artificial splitting of the whole situation into a few RDF statements. Statements 4 and 5 are represented by separate RDF statements, but they would more intuitively be represented by a single unit containing the whole situation.

The metagraph approach helps to represent this example in a more natural and holistic way. From the metagraph point of view, this example contains three nested situations:

- Situation 1. John arrived in London.
- Situation 2. Paul noted at 4 p.m. situation 1.
- Situation 3. James noted situation 2.

Each situation is represented by a metavertex as shown in fig. 4. Attribute “has_time=4 p.m.” may be binded either to edge “noted” or to metavertex “Situation 2” (fig. 4 shows both cases).

The textual representation of fig. 4 is shown below:

```
Metavertex(Name=Situation3,
  Vertex(Name=James),
  Metavertex(Name=Situation2,
    Attribute(has_time,"4 p.m."),
    Vertex(Name=Paul),
    Metavertex(Name=Situation1,
      Vertex(Name=John),
      Vertex(Name=London),
      Edge(Name=arrived_in, vS=John, vE=London,
        eo=true)),
    Edge(Name=noted, vS=Paul, vE=Situation1, eo=true,
      Attribute(has_time,"4 p.m."))),
  Edge(Name=noted, vS=James, vE=Situation2, eo=true))
```

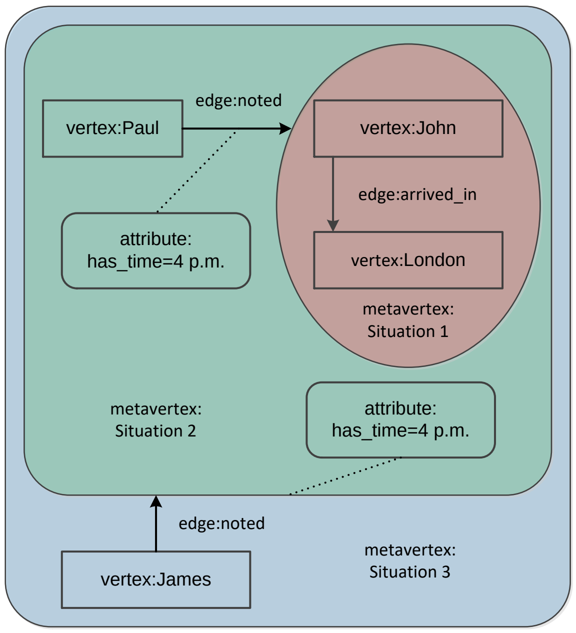


Fig.4. Metagraph representation of RDF reification

This considered example shows that the metagraph approach allows representing reification without emergence loss, keeping each nested situation in its own metavertex.

B. N-ary relationship limitation example

An N-ary relationship is a situation where a predicate combines several subjects or objects, or has nested predicates. Such a situation is a problem from an RDF point of view. To address this problem, the W3C Working Group Note was published [2].

Consider the example of the complex statement: ‘John arrived to London at 4 p.m. by train in order to meet his classmates James and Paul'. This is a typical example of an N-ary relationship as shown in fig. 5. Both problems shown in fig. 5 cannot be represented by a pure RDF triplet model.

The "Problem_arrived" is that the predicate "arrived_to" has nested predicates "has_time" and "by_transport". According to [2] we are adding a supporting subject to "Problem_arrived" representing an instance of a relation.

The "Problem_meet" is that the predicate "to_meet" has two objects "James" and "Paul". According to [2] we have several ways to solve this problem. We may use the list construct of RDF or we may join object "James" and "Paul" into the classmates group. We do the latter in this example.

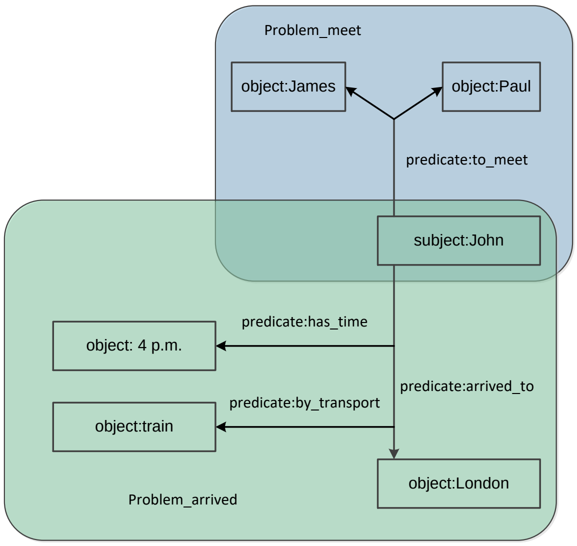


Fig. 5. Example of N-ary relationship

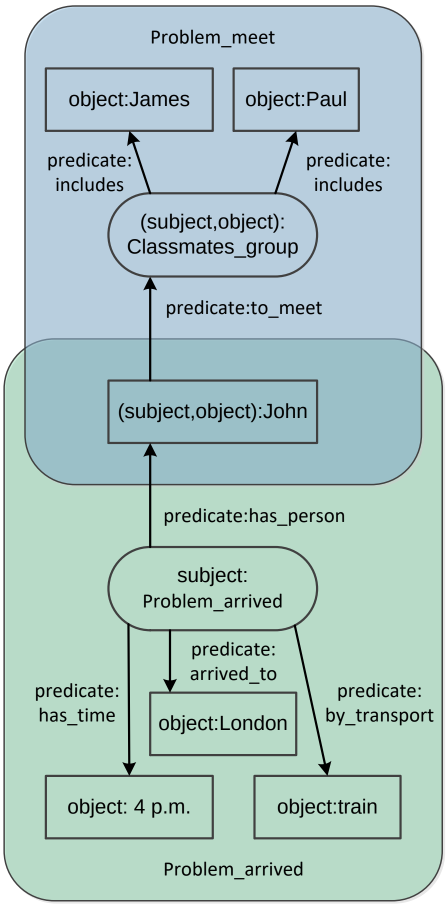


Fig. 6. RDF representation of N-ary relation example

The solution is shown in fig. 6. We have added supporting vertices "Classmates_group" and "Problem_arrived", which are shown in rounded boxes. In predicate "to_meet" the "Classmates_group" is an object while in predicate "includes" it is a subject. In predicate "has_person", "John" is an object while in predicate "to_meet" he is a subject.

Since we do not use reification, this may be represented in the RDF triple form "`subject predicate object`" as follows:

```
1. Problem_arrived has_person John
2. Problem_arrived arrived_to London
3. Problem_arrived by_transport train
4. Problem_arrived has_time "4p.m."
5. John to_meet Classmates_group
6. Classmates_group includes James
7. Classmates_group includes Paul
```

As in the reification example, the problem here is in emergence loss due to the artificial splitting of the situation. The "Problem_arrived" vertex is added not because it describes the situation in a natural way, but because it is required to keep a consistent triplet structure. In a large RDF graph, many supporting vertices may obscure meaningful understanding the situation.

As in the reification example, the metagraph approach helps to represent this example in a more natural and holistic way as shown in fig. 7.

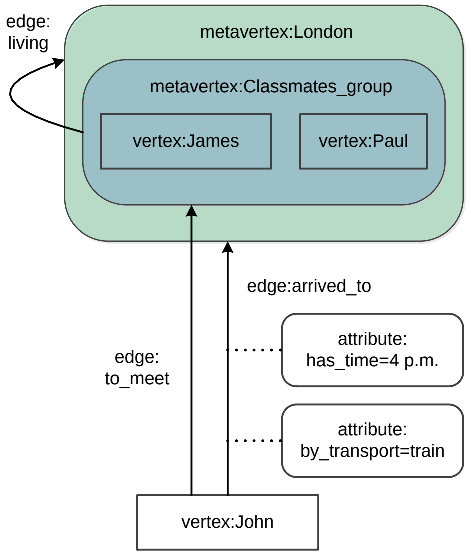


Fig. 7. Metagraph representation of N-ary relation example

The "Problem_arrived" is solved by binding attributes "has_time=4 p.m." and "by_transport=train" to the edge "arrived_to". The "Problem_meet" is solved by using metavertex "Classmates_group" which includes vertices "James" and "Paul".

The implicit knowledge about "Classmates_group" living in London may be shown either by the edge "living" or by inclusion of metavertex "Classmates_group" into metavertex "London" (fig. 7 shows both cases).

The textual representation of fig. 7 is shown below:

```
Metavertex(Name=London,
  Metavertex(Name=Classmates_group,
    Vertex(Name=James),
    Vertex(Name=Paul),
    Edge(Name=living, vS=Classmates_group, vE=London,
      eo=true)))
Vertex(Name=John)
Edge(Name=to_meet, vS=John, vE=Classmates_group,
  eo=true)
Edge(Name=arrived_to, vS=John, vE=London, eo=true,
  Attribute(has_time,"4 p.m."),
  Attribute(by_transport, train))
```

This considered example shows that the metagraph approach allows the representation of N-ary relations without emergence loss, keeping each nested situation in its own metavertex.

Summing up the examples section, it should be noted that the metagraph model addresses RDF limitations in a natural way without emergence loss. Proposed textual representation of the metagraph allows clear and emergent description of examined problems.

IV. Research Results and Conclusions

The research presented in this paper shows that in spite of the advantages of RDF modelling, it has poor capabilities for representing complex situations.

Complex situations representation requires more complex graph data modelling, requiring supporting the emergence principle of general systems theory.

The flat graph model and near flat hypergraph model do not allow implementation of the emergence principle. Here, the metagraph model is proposed to implement this principle.

Along with the formal description of the metagraph model, the textual predicate representation is also proposed.

The given examples show that the metagraph model is suitable for representing complex situations without emergence loss, and the proposed textual representation gives clear and emergent description of complex situations.

Nowadays, graph database systems supporting emergent graph models are only in their infancy. There is no doubt that developing such databases and query languages for emergent graph models will become essential research and practice challenges.

References

[1] D. Allemang, J. Hendler, Semantic Web for the working ontologist: effective modeling in RDFS and OWL – 2nd ed. Elsevier, 2011, 369 p.

[2] Defining N-ary Relations on the Semantic Web. W3C Working Group Note 12 April 2006. [Online]. Available: http://www.w3.org/TR/swbp-n-aryRelations/

[3] A. Basu, R. Blanning, Metagraphs and Their Applications. Springer, 2007, 174 p.

[4] E. Samohvalov, G. Revunkov, Yu. Gapanyuk, "Metagraphs for describing semantics and pragmatics of information systems," in Herald of Bauman Moscow State Technical University, vol. 1(100), 2015, pp.83-99.

[5] Vitaly I. Voloshin. Introduction to Graph and Hypergraph Theory. Nova Science Publishers, Inc., 2009, 287 p.

[6] RDF Primer. W3C Recommendation 10 February 2004. [Online]. Available: https://www.w3.org/TR/2004/REC-rdf-primer-20040210/