JSON Documents Processing Using Situation-Oriented Databases

Valeriy Mironov[1], Artem Gusarenko[1], Nafisa Yusupova[1], Yuriy Smetanin[2]

[1]Ufa State Aviation Technical University, Computer Science & Robotics Dept., Karl Marks st., 12, 450008 Ufa, Russia;
{mironov, gusarenko}@ugatu.su, yussupova@ugatu.ac.ru

[2]Russian Foundation for Basic Research, Head of Information Technologies & Computing Systems Dept., Leninsky Prospekt st., 32a, 119334 Moscow, Russia;
smetanin@rfbr.ru

Abstract: Situation-oriented databases provide processing of documents from heterogeneous data sources under the control of a hierarchical situational model. This article discusses the problem of processing database documents in JSON format, along with XML. Two implementation approaches are discussed: (1) on the fly JSON to XML document conversion and using Document Object Model for processing XML, and (2) loading the JSON document into an associative/indexed array followed by applying the template engine. The database interpreter works with external heterogeneous data extracted from files, databases, archives, web services, data is processed using virtual documents. Examples of processing JSON documents received from a web service are analyzed. Data from the San Francisco Open Data web server is used as the JSON test source. Query in Socrata Query Language used for JSON data extraction is presented. The implementation of approaches in the research situation-oriented database prototype based on Hypertext Preprocessor is considered.

Keywords: hierarchical model; situation-oriented databases; NoSQL; databases; model-driven development; open data

1 Introduction

Web applications that are widely used now, in the course of performing their functions, should support the interaction, on the one hand, with users and other applications, on the other hand, with a variety of local and remote data sources. Traditionally, relational databases were used as data sources, which caused the problem of mapping [1] relational data to data [2] in other formats, such as XML [3]. However, Now the range of data formats used has significantly expanded, in

particular, thanks to the success of the NoSQL [4] movement (such as Key-Values databases, JSON-oriented document databases, Graph databases) and Polyglot Persistence [5]. In this regard, new approaches to the flexible processing of heterogeneous data in web applications are needed. The development of modern web applications is strongly influenced by new approaches such as Service-Oriented that estimated in work [6] in aspect of characteristics of Model-Driven Architectures which use domain-specific languages [7]. Recently, the experience of creating situation-oriented web applications has become relevant [8], applicable for NoSQL in model-driven [9] database development such as SODB together with Big Data technologies [10].

The ideas put forward in the framework of these approaches have had multiple impacts, including on Situation-Oriented Databases (SODB) [11]. SODB is a new approach to developing web applications, driven by the embedded Hierarchical Situation Model (HSM) [12]. It is based on the uniform processing of heterogeneous data [1]. The SODB approach novelty in the field of heterogeneous data integration is the concept of virtual documents that are mapped to real data in various formats (Fig. 1). When a virtual document is declared in the HSM model, it's mapping to real data is specified. After that, the processing of this virtual document can be specified with the reference to its declaration. The advanced invariance principle assumes that changing the mapping [2] specifications should not affect the specifications for processing virtual documents.

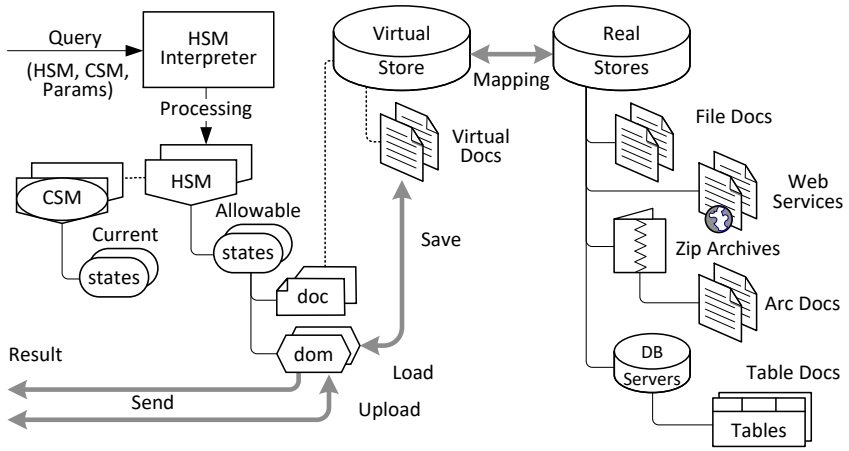


Figure 1
SODB architecture

Initially, SODB focused on processing virtual documents in XML format and on the using of appropriate XML technologies such as XSLT [11]. Recently, for XML has appeared a competitor such as JSON. Due to its simplicity and laconism, JSON has become popular and is actively used along with XML when exchanging data between web servers [3] and between the browser and the server.

In this regard, there was an implementation problem of the JSON data processing functions in the SODB.

2 Problem Discussion

The SODB architecture, presented in Fig. 1, explains the principles of processing heterogeneous documents [12]. The basis of the SODB is the hierarchical situation model [6] HSM. HSM specifies a hierarchical set of allowable business process states, conditions for state transitions, actions provided in the current states.

The HSM Interpreter executes the HSM model on external queries, identifying the current states and performing the actions associated with the current states. The sets of current states (CSM – Current State Model) are saved between the interpretation cycles.

There are two types of elements for processing data in the HSM: (1) the doc element specifies a virtual XML document, and (2) the dom element specifies a DOM object (XML Document Object Model) for loading and processing a virtual document. A virtual XML document can be mapped on a real document in the form of a local XML file, a remote XML web service, a ZIP archive containing compressed XML files, a relational database that stores XML documents. Similarly, real JSON document can exist in the form of a local JSON file, remote JSON web service, ZIP-archive with JSON-packed files, relational database, storing JSON-documents.

The conceptual model [7] for processing a virtual XML document is shown in Fig. 2 *a*. Here is a fragment of the HSM model, including the state sta, which contains three child elements: (1) doc is a definition of a virtual document, (2) dom is a virtual document processing object, and (3) wdg is a widget that forms the HTML-code for sending to the user's browser. A real XML document that maps a virtual document is loaded into the DOM object. Here, it is processed, for example, in the form of XSL transformation in accordance with the specified XSLT stylesheet. The result of the processing is used to generate a view in the client's browser using a widget.

Two approaches can be proposed to implement this scheme in the case where a virtual document is mapped to a JSON document: (1) JSON to XML transformation, and (2) processing JSON in special objects.

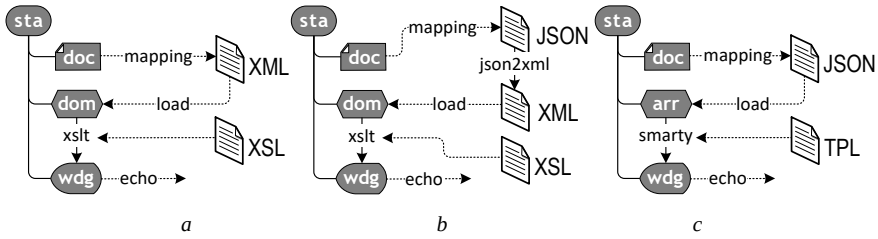


Figure 2
Models of XML / JSON processing in SODB

2.1 JSON to XML Transformation

The model for processing a JSON document with transformation to XML format is shown in Fig. 2 *b*. The conversion is done on the fly before loading the document into a DOM object, further processing is performed just like processing an XML document. The advantage of this approach is due to the large capabilities of XSLT with respect to JSON. Some difficulties originate if the result of processing should be in JSON format. An additional XSL transformation is required to reverse the conversion of XML to JSON in this case.

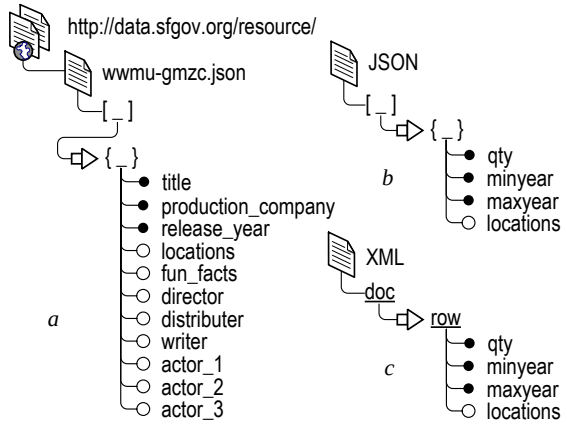
2.2 JSON Processing Objects

The JSON document processing model using JSON processing objects is shown in Fig. 2 *c*. In this case, the JSON document is loaded into a special object for processing. Object arr is an associative array (an abstract data type composed of a collection of 'key, value' pairs), which provides access to particular elements of the JSON document. A suitable template engine, such as Smarty, can be used to handle an associative array and form the widget's code. The capabilities of the template engine are more modest than XSLT, but they are usually sufficient to solve a wide range of transformation tasks.

Both approaches discussed above were implemented in the HSM Interpreter research prototype. Interpreter is written in PHP [3] and is used as an engine for the server part of the web application. The performance of the approaches has been tested on several test cases, two of which are detailed below.

3 PHP-based Examples

Consider two examples of workable HSM models that illustrate the technique of applying both approaches.



d

Qty	Range	Locations
54	1935--2010	
27	1941--2014	Golden Gate Bridge
22	1936--2017	City Hall
19	1938--2015	Fairmont Hotel (950 Mason Street, Nob Hill)
14	1995--2017	Treasure Island
10	1961--2008	Chinatown
10	1936--2015	Coit Tower
10	1956--2000	St. Peter & Paul's Church

Figure 3
Web service JSON data source example

3.1 Used JSON Data Source

Data from the SF OpenData web server, which provides a variety of information about San Francisco, including in JSON format, was used as the JSON test source. Namely, Film Locations service was used (a filming locations listing of movies shot in San Francisco). The structure and composition of the facts about the films and locations provided by the data source are shown in Fig. 3 *a*.

Let us need summary data, not just facts. For example, suppose for each `location` you need to calculate the number of facts (`qty`) and their time range (`minyear, maxyear`) (Fig. 3 *b*). Equivalent representation in XML-format is given on Fig. 3 *c*. Thus, we want to obtain the result in the form presented in Fig. 3 *d*. This problem can be solved in two ways: (1) to download fact-document in the original format Fig. 3 *a* and convert it to the desired format by XSL transformation, and (2) to receive the document in the desired format Fig. 3 *b* from the web service by sending a request to the server. In the tests we use the second way.

The SF OpenData service allows you to formulate GET queries using the SoQL – SQL-like query language. To solve our problem, a query was constructed, which provides grouping by attribute locations, calculation of statistical indicators for groups by means of aggregate functions count, min, max, restriction and sequencing of output results:

```
$select = locations, count (*) as qty,
        min (release_year)   as minyear,
        max (release_year)   as maxyear
$group = locations
$having = qty > 9
$order = qty desc
```

Before sending to the server, the query must be coded in accordance with the rules for setting parameters in HTTP GET requests.

3.2 JSON to XML Transformation Example

In Fig. 4 is an example of the HSM model, which involves processing a JSON document based on JSON to XML transformation. In the state `sta:JSON-Processing` two virtual multi-documents are specified.

Multi-document `doc:SFGov` is provided for downloading JSON-data. It is mapped to web services SF OpenData [8]. The type attribute indicates the JSON-format documents. `Entry`-element `ent:FilmLocs` defines a separate virtual document in the multi-document, mapped to the Film Locations in San Francisco Web service [8]. The `path` attribute specifies a resource, and the get attribute specifies an SoQL query that is coded in accordance with the rules for creating GET parameters.

Multi-document `doc:TmpDoc` is provided to demonstrate the possibility of local saving of downloaded data. It serves to store data in a relational database on a MySQL server. The `action` attribute instructs to establish a connection with MySQL using the default parameters. Element `ent:putDoc` specifies a virtual document that is stored in the cell of table `xmldocs` in the line with the given value of the identifier id in column doc. Element `ent:getDoc` is specified a virtual document extracted from the same cell of the table `xmldocs`.

In the root state a submodel [9] `sub:proc` is envisaged, providing processing of virtual documents. The internal state `sta:proc` is intended to handle under normal conditions and states `sta:loadErr` and `sta:saveErr` – when an error occurs.

In the state `sta:proc` first element `dom:Films` creates and loads a DOM-object. The `srcDoc` attribute refers to the virtual document `ent:FilmLocs` in the multi-document `doc:SFGov`. Therefore, a corresponding JSON document is downloaded from the corresponding web service (see Fig. 3 *b*). This document is converted on the fly into XML format before loading into the DOM object. During the conversion, the JSON array element (enclosed in square brackets) wraps itself in tags `doc`, and the JSON object element (bounded by curly brackets) turns into tags `row` (see Fig. 3 *c*). The `onLoadErr` attribute instructs to jump to the `sta:loadErr` state if errors are detected during the load process. Presence of the attribute `saveDoc` means that after processing the contents of the DOM object must be stored in the corresponding virtual document. Attribute refers to a virtual document `ent:putDoc` of the multi-document `doc:TmpDoc` therefore XML-based representation JSON-document will be saved in the corresponding MySQL table cell. The `onSaveErr` attribute instructs to jump to the `sta:saveErr` state if error is detected during data saving.

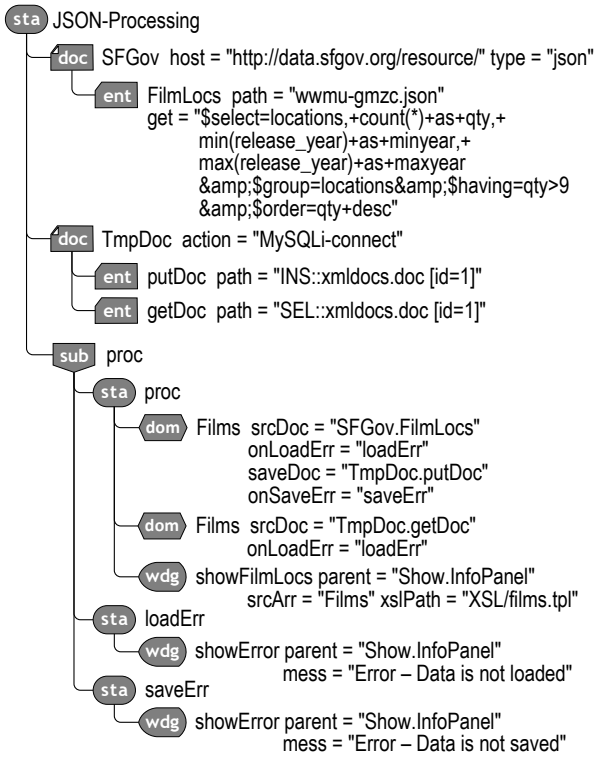


Figure 4
HSM model examples based on JSON to XML transformation

The second `dom:Films` element reloads the DOM object. The `srcDoc` attribute refers to the `ent:getDoc` virtual document in the multi-document `doc:TmpDoc`. Therefore, the XML-document will be loaded from the cell MySQL table that has been saved earlier. (Note that the same content is loaded into the DOM object. This is done only to demonstrate the technique of loading an XML document from the MySQL database.)

The `onLoadErr` attribute prescribes go into a state `sta:loadErr` if errors are found in the load process.

Widget-element `wdg:ShowFilmLocs` generates HTML-code snippet to display the result in the client browser (see Fig. 3 *d*). The attribute `srcDom` refers to `dom:Films`, i.e. the HTML code of the widget is formed based on the content of this DOM object. The attribute `xslPath` prescribes to generate HTML-code by XSL Transformation of XML-content in accordance with the style sheet file `XSL/films.xsl`. The attribute parent points to the parent widget, the HTML code of which should be embedded with the resulting HTML code of this widget, i.e. it is assumed: (1) That in some state that is a parent of the state `sta:proc`, widget-element `wdg:Show` is defined, and (2) that the HTML code generated by the widget `wdg:Show`, contains an identifier attribute `id = "InfoPanel"`.

In the states `sta:loadErr` and `sta:saveErr` widgets `wdg:ShowError` generate an error message when loading and saving data [10]. These messages are inserted into the same point in the HTML of the parent widget, into which the result is inserted in the absence of errors.

3.3 JSON Processing Objects Example

In Fig. 5 is an example of the HSM model, which involves processing a JSON document based on JSON processing objects. Syntactically, this model is very like the previous one (see Fig. 4), it solves the same problem, but in a different way. Definitions of virtual documents `doc:SFGov` and `doc:TmpDoc` have not changed, except that the document `doc:TmpDoc` maps to the table `jsondocs`. The main difference is that `arr`-elements are used instead of the `dom`-elements.

`Arr`-elements are provided in the HSM-model specifically for JSON documents. When interpreted, they generate `Arr` objects – multidimensional arrays designed to load and process JSON documents during interpretation of the HSM model. When loaded, each JSON object is represented as an associative array, and the JSON array is represented as an indexed array.

The first `arr:Films` element creates and loads an `Arr`-object. The attribute `srcDoc` refers to a virtual document `ent:FilmLocs` in multi-document `doc:SFGov`. Therefore, a JSON document is downloaded from the web service and loaded into the `Arr`-object. As before, the `onLoadErr` attribute prescribes

to go to the `sta:loadErr` if errors are found in the boot process. The presence of the attribute `saveDoc` means that after the processing, the contents of the `Arr`-object must be saved in the `ent:putDoc` of the multi-document `doc:TmpDoc`. Therefore, `arr:Films` content is converted back into a JSON document that is stored in the MySQL table cell. The attribute `onSaveErr` prescribes go into a state `sta:saveErr` if there are errors in the save process. The second `arr:Films` element reloads the `Arr` object. The attribute `srcDoc` refers to virtual document `ent:getDoc` in the multi-document `doc:TmpDoc`. Therefore, the JSON document will be extracted from the MySQL table cell in which it was previously saved. The attribute `onLoadErr` prescribes go into a state `sta:loadErr` if errors are found in the load process.

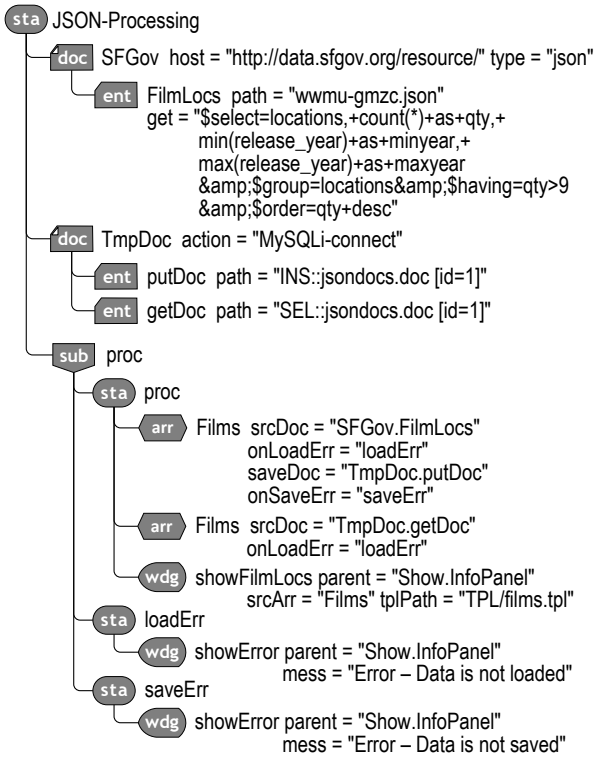


Figure 5
HSM model examples based on JSON processing objects

Widget element `wdg:ShowFilmLocs` generates HTML-code snippet to display the result in the client browser (see Fig. 3 *d*). The attribute `srcArr` refers to `arr:Films`, i.e. HTML-code of the widget is formed on the basis of the content of this `Arr`-object. The `tplPath` attribute instructs to generate the HTML code from the `Arr`-object content using the Smarty template engine per the template

file `TPL/films.tpl`. The attribute `parent` indicates that the resulting HTML code of this widget should be embedded in the parent HTML code of the widget `wdg:Show`.

In states `sta:loadErr` and `sta:saveErr` the `wdg:ShowError` widgets generate an error message when loading and saving data. These messages are inserted into the same point in the HTML of the parent widget, into which the result is inserted in the absence of errors.

4 The Results Discussion

`Two approaches`. Now DBMS can work with JSON-documents in the sense that it can maps virtual documents for real data in JSON-format. When using the first approach, internal processing of JSON documents is performed, as before, in XML format. When using the second approach, internal processing is performed in the format of multidimensional associative / index arrays. If the first approach allows using the power of XML technologies, then the second approach is more familiar for JSON documents.

`High abstraction`. As in the case of XML, JSON specifications are set at a high level of abstraction. Essential aspects of JSON documents are specified in the HSM model declarations, and technical details such as using an algorithm for converting JSON to XML format, creating objects and loading JSON documents into them, etc. are hidden from the developer and executed by the interpreter. For example, the HSM model developer does not need to explicitly specify which of the two approaches to use for processing the JSON document. Based on the context, the interpreter chooses either transformation into an XML-format if the document needs to be loaded into the DOM-object, or conversion to a multidimensional array if the document needs to be loaded into the Arr-object.

`Uniformity`. With both approaches to providing JSON functionality, the structure of the HSM models remains unchanged. The structure is the same as in the case of using XML-documents, the differences appear only in the attributes. Thus, a successful linguistic notation has been found both for specifying virtual documents and for processing them: using a doc-element, a virtual multi-document is defined and its mapping to real data is specified, and using dom/arr-elements, a uniform processing of the virtual document is specified.

`Invariance`. The above invariance principle is observed within the same approach, i.e., changing the address of the JSON document and the physical store will require changes to the specifications of the doc-element and will not require changes to dom/arr elements that reference this doc-element. At the same time, the principle of invariance is violated when moving from the JSON to XML method to the JSON processing objects method, since it will require you to replace the

dom-elements with arr-elements and change the attributes that is, you need to make changes to the document processing specifications. In the future, it is supposed to investigate the possibility of unification so that in the HSM-model one could use some generalized Document Processing Object element, which, depending on the context, can be interpreted as a dom-element or as an arr-element.

`Implementation`. The above functionality of virtual multi-documents is implemented in the research prototype of the HSM interpreter as part of the engine for managing web applications based on DBMS. The prototype of the HSM interpreter, written in PHP language, as well as test HSM-models, other programs, and data of Web applications are located on the webserver http://hsm.ugatu.su. When a client accesses the root script of a web application, the HSM interpreter runs, which executes a cycle of interpretation of the corresponding HSM model. In the process of interpretation, the corresponding CSM-model is corrected and the resulting HTML-code sent to the client. The built-in features and add-ons of the PHP-platform are actively used for access to local files, archives, relational databases [4], remote web servers.

`Effect`. Using the proposed approach reduces the complexity of programming web applications based on DBMS, compared with traditional "manual" programming in the languages of server-side scripts such as PHP. This is achieved due to a higher level of abstraction of the HSM model when the set of routine operations specified in the declarative form when determining virtual multiloquent is assigned to the HSM-model interpreter.

Conclusions

Being an information processor for processing documents from heterogeneous data sources, situationally-oriented databases need the functionality of processing JSON documents. The approaches discussed above make it possible to solve this problem by transforming the JSON document either into an XML format on the fly or into an associative-index array. In the second case, it was required to provide in the situational model special arr-elements, which generate associative-index arrays for loading and processing JSON documents during interpretation. A comparative consideration of both approaches in a practical example of processing the document retrieved from the web service showed a high uniformity of the situational models used for this. JSON-functionality is implemented in the prototype of the HSM interpreter on the PHP platform as part of the control mechanism of web applications based on SODB.

Acknowledgment

This work is supported by RFBR grant 19-07-00682.

References

[1] Janga, P., Davis, K. C.: Mapping heterogeneous XML document collections to relational databases. Lecture Notes In Computer Science, Vol. 8824, 2014, pp. 86-99

[2] Cobo, M. J. et al.: A relational database model for science mapping analysis. Acta Polytechnica Hungarica, 12 (6), 2015, pp. 43-62

[3] Amanatidis, T., Chatzigeorgiou, A.: Studying the evolution of PHP web applications. Information and Software Technology, 72, 2016, pp. 48-67

[4] Varga, V. et al.: Conceptual design of document NoSQL database with formal concept analysis. Acta Polytechnica Hungarica, 13 (2), 2016, pp. 229-248

[5] Fowler M., Sadalage P. J., NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, 2013

[6] Ameller, D., Burgués, X., Collell, O., et al.: Development of service-oriented architectures using model-driven development: A mapping study. Information and Software Technology, Vol. 62, 2015, pp. 42-66

[7] Kosar, T. et al.: Domain-Specific Languages: A Systematic Mapping Study. Information and Software Technology, 71, 2016, pp. 77-91

[8] Xuanzhe Liu et al.: Data-driven composition for service-oriented situational web applications. IEEE Transactions on Services Computing, Vol. 8, Iss. 1, 2015, pp. 2-16

[9] Agh, H., Ramsin, R.: A pattern-based model-driven approach for situational method engineering. Information and Software Technology, 78, 2016, pp. 95-120

[10] Osvaldo, S. S., Jr. et al.: Developing software systems to Big Data platform based on MapReduce model: An approach based on Model Driven Engineering. Information and Software Technology, 92, 2017, pp. 30-48

[11] Mironov, V. V., Gusarenko, A. S., Yusupova, N. I.: Situation-oriented databases: document management on the base of embedded dynamic model. CEUR Workshop Proceedings (CEUR-WS.org): Selected Papers of the XI International Scientific-Practical Conference Modern Information Technologies and IT-Education (SITITO 2016) Vol. 1761, P. 238-247, Moscow, Russia, 2016, pp. 238-247

[12] Mironov, V. V., Gusarenko, A. S., Yusupova, N. I.: The invariance of the virtual data in the situationally oriented database when displayed on heterogeneous data storages. Vestnik Komp'iuternykh i Informatsionnykh Tekhnologii [Herald of Computer and Information Technologies], No. 1 (151) 2017, pp. 29-36 (In Russian)