Advances in Intelligent Systems Research, volume 174
Proceedings of the 8th Scientific Conference on Information Technologies for Intelligent Decision Making Support (ITIDS 2020)

RESTful Web Services Development in Situation-Oriented Databases

Valeriy Mironov
Computer Science & Robotics Dept
Ufa State Aviation Technical University
Ufa, Russia
mironov@ugatu.su

Artem Gusarenko*
Computer Science & Robotics Dept
Ufa State Aviation Technical University
Ufa, Russia
gusarenko@ugatu.su

Nafisa Yusupova
Computer Science & Robotics Dept
Ufa State Aviation Technical University
Ufa, Russia
yussupova@ugatu.ac.ru

***Abstract*—New features RESTful web services development, envisaged in the situation-oriented databases (SODB), are considered. SODB is a heterogeneous data integrator driven by an embedded hierarchical situational model (HSM), in which virtual documents are mapped onto heterogeneous physical data. Microservices and microservice architecture are discussed in terms of the benefits of scaling and modifying web applications. Elements of the situational model that define HTTP-request processing are considered in terms of access to request properties and attached data based on the concept of virtual documents. An HTTP-request processing design pattern developed to work with relational database tables for which data sources are external web services is discussed. The structure of the HSM-model, which performs selective processing depending on the type of request, is discussed in detail for HTTP-request methods GET and POST. An example of the practical implementation of microservices based on SODB for the problem of monitoring student views of educational videos on YouTube is given.**

Keywords—situation-oriented database, embedded dynamic model, SOA, web service, microservice, RESTful, HTTP-request, HSM, XML, JSON, YouTube API

I. INTRODUCTION

RESTful Web Services (or simply REST) are web services built on the principles of Representational State Transfer – an architectural style of interaction between components of a distributed web application. Service Oriented Architecture (SOA), of which REST is a variation, has become popular and widely used in web applications due to its high scalability and manageability [1]. The current trend in SOA development is the use of microservices which are characterized by extreme isolation and granularity in order to increase modifiability [2].

The benefits of SOA are the reason why many modern information technologies strive to include support for services and microservices in their functionality [3]. In this article, such support is considered in relation to situation-oriented databases (SODB) [4]. SODB is a project for the creation and development of an information processor controlled by a highly abstract situational model for processing heterogeneous data [6]. As a computer program, SODB is an interpreter of hierarchical situational models (HSM), presented as a hierarchy of submodels [7]. Each submodel defines a set of possible states and transitions between them. Actions associated with the state are executed when the state becomes current. Heterogeneous data processing in the SODB is based on the concept of virtual documents [8]. A virtual document is a special action that sets up a mapping to real data in a physical storage. Virtual documents are loaded into data processing objects, processed there and uploaded to other virtual documents. There, a uniform processing of heterogeneous data is achieved.

The issues of using REST were considered by the authors in articles [1]. This article is devoted to new developed capabilities in terms of building microservices based on SODB.

II. SERVICES AND MICROSERVICES

Monolithic application architecture. The monolithic architecture, according to which web applications have traditionally been built, is efficient and convenient with a small number of business process service functions [9]. When additional tasks arise, the costs of scaling, testing increase, and difficulties arise in modifying and expanding functionality. Emerging problems lead to preference service-oriented architecture [10]:

- *Style violation when adding features.* When expanding the application, problems arise in understanding the program code, since the changes are knocked out of the accepted style of building the application.
- *Complicated testing and scaling.* When making additions, the application must be tested as a whole, and not as a separate part. When scaling an application, you must copy it to the new nodes, introducing unjustified redundancy.
- *Link to technology and programming languages.* Monolithic architecture is demanding on language tools. When expanding the application, you must focus on one programming language, framework, virtual machine etc.

Microservice architecture. The new style of application development governs the creation of a separate program (microservice) to serve a separate limited function of the business process, and not the entire complex of tasks [11]. The application turns into a set of services, each service performs a limited automated function [12].

This approach improves scalability and testability, facilitates the use of various languages and technologies, and increases modularity [13]. The application consists of

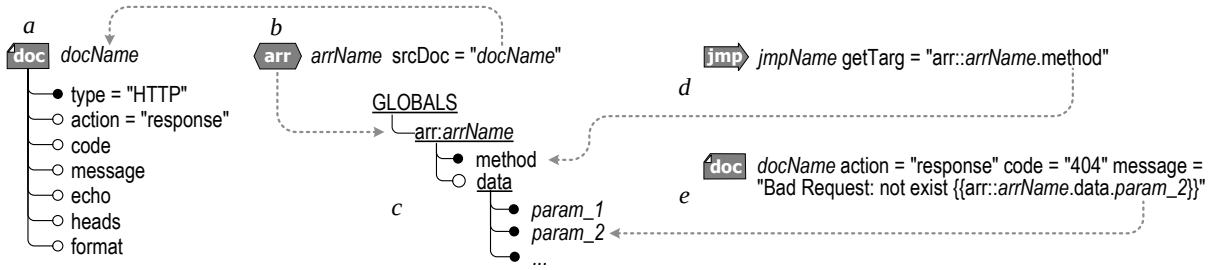

Fig. 1. HSM elements for HTTP-Request processing: *a* —virtual document of HTTP-type; *b* — object of processing data of type "associative array"; *c* — structure of the HTTP-Request property array; *d* — using HTTP-Request properties to transition to a new state; *e* — using HTTP Request properties with injection.

modules that are independent of each other and may not consider the existence of each other. When adding functionality, a new service is being developed. This makes it easy to test, build, experiment without interfering with the rest of the application, without repeatedly building the application. In these conditions, the organization of the relationship between services and data exchange becomes an important task. Microservice architecture allows developers to work independently on individual services [14].

As a rule, a microservice has a separate database that is independent of other services and serves the functions (business logic) of this service [15]. The microservice API (Application Program Interface) is created to communicate with the outside world through external requests, commands, CRUD-operations with the database (Create, Read, Update, Delete) [16].

Client programs that send requests to the service and receive the processing result are created to use the service. Special protocols and extensions are used to implement interaction with services. For example, RESTful services are based on the Internet protocol HTTP. The client program sends the service a web request (HTTP Request) of one of the types (Request Method; usually it is GET, POST, PUT, DELETE), attaching data to it in the form of a list of parameters or a file. The GET method is intended to receive the content of the resource from the service; The POST method is used to transfer user data to the resource; The PUT method is used to load the resource; DELETE method deletes a resource. Having received the HTTP Request, the service can find out its method and extract the attached data. The client receives a response from the service (HTTP Request Response), which contains a status code characterizing the success of the request, and data (usually in JSON, XML or HTML format) [17].

III. HTTP-REQUEST PROCESSING IN SOBD

For HTTP-Request processing, the service should be able to find out the Request Method and other data associated with the request. In addition, the service should be able to generate and send an HTTP Request Response with the corresponding status code and the resulting data. Both the first and second specified functions are performed using a virtual document of a new type of `HTTP`, designed to solve this problem. The corresponding elements of the HSM model and their interaction are illustrated in Fig. 1.

Fig. 1, *a* represents the HSM element of a virtual document `doc:docName`, where `docName` is the name of the element that the programmer sets. The required attribute `type = "HTTP"` specifies the type of document. The optional `action = "response"` attribute instructs to form and send an HTTP-Request Response. The optional `code`, `message`, `heads`, `echo`, and `format` attributes specify information for Response: the status code that explains the message, Response headers, HTML data, and the data format in general.

Fig. 1, *b* represents an HSM associative array element (a kind of data processing object) `arr:arrName`, where `arrName` is the name of the element specified by the programmer. It is designed to process a virtual document loaded into it [19]. The `srcDoc` (source document) attribute indicates a virtual document to be loaded into an associative array. In this case, it refers to the virtual `doc:docName HTTP`-document.

Fig. 1, *c* represents the structure of an associative array in `arr:arrName` loaded from the `doc:docName` virtual `HTTP`-document. The global array `arr:arrName` contains a tree of elements of type *key–value*. The `method` element contains the value of Request Method. The `data` element contains a nested array of parameters passed from request. Other HTTP Request properties, such as request headers, are made available in a similar way.

Fig. 1, *d* illustrates the use of the HTTP Request properties from the `arr:arrName` array to control the transition to a new state. The `jmp:jmpName` element (`jmpName` is the element name specified by the programmer) makes a transition to a new state, the name of which is extracted from the `arr:arrName` array. The `getTarg` attribute refers to the method element of the `arr:arrName` array, whereby the `HTTP` Request Method value is used as the name of the new state for the transition.

Fig. 1, *e* illustrates another way of using the `HTTP` Request properties from the `arr:arrName` array. The `doc:docName` element with the `action = "response"` attribute instructs to form and send a Request Response with the status code 404 and an explanatory message specified in the `message` attribute. The {{...}} construct located in the body of the explanatory message is an injection that inserts the parameter value extracted from the `arr:arrName` array into the message.

The above features (along with other data processing tools in HSM) allow you to successfully develop microservices for processing HTTP Request.

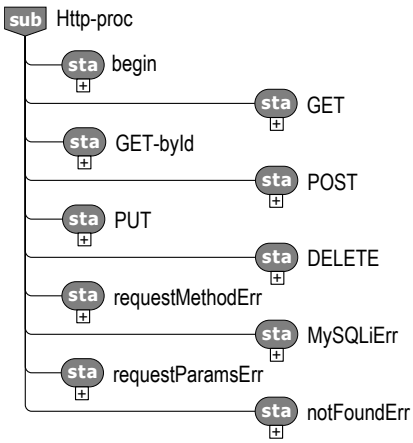

Fig. 2. Submodel states `sub:Http-proc` HTTP Request processing

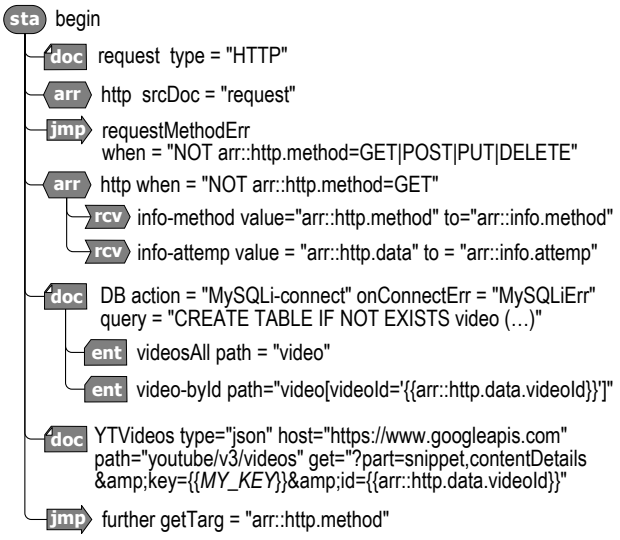

Fig. 3. Initial state of submodel `sub:Http-proc`

IV. HTTP REQUEST PROCESSING PATTERN

The `Http-proc` pattern of a typical HTTP Request handler focused on CRUD processing of relational database tables for which an external web service serves as a data source has been developed. The MySQL relational database table containing video information and the YouTube API web service providing YouTube video information are used as an example in the pattern.

The general structure of the pattern is shown in Fig. 2–6 using graphical HSM notation [22]. Here, the submodel is disclosed at the state level. The structure of individual states is detailed below in Fig. 3–6.

A. Submodel states

The `sub:Http-proc` submodel contains sample states that correspond to typical HTTP Request processing situations:

- The `sta:begin` state is for the initial processing of Request. Virtual documents that are displayed on both the HTTP Request and other processed physical data are declared and verified in this state.
- The states `sta:GET`, `sta:POST`, `sta:PUT`, `sta:DELETE` are intended to continue processing the Request in accordance with the Request Method (GET, POST, PUT, DELETE methods, respectively).
- The `sta:GET-byId` state is designed to specifically handle the situation using the GET method, if required in the `sta:GET` state. Similar states for the POST, PUT, DELETE methods are omitted in Fig. 4).
- The states `sta:requestMethodErr`, `sta:MySQLiErr`, `sta:requestParamsErr`, `sta:notFoundErr` are designed to handle various situations related to errors and exceptional conditions. Typically, Response generation with the appropriate return code and message is performed in such states.

Transitions between states are set both explicitly, using `jmp`-elements, and implicitly, using the corresponding attributes in other elements [23].

B. Initial State

The initial state of `sta:begin` is detailed in Fig. 3. The `doc:request` virtual document provides access to the HTTP Request properties through the `arr:http` associative array. The `jmp:requestMethodErr` transition element checks the validity of the used Request Method (if the method does not match the set of valid ones, it will go to the error handling state `sta:requestMethodErr`).

The `arr:info` associative array contains information returned as a Response in response to an HTTP Request. In the `sta:begin` state, information about the original request (method and parameters) is entered into this array using the receiver elements `rcv:info-method` and `rcv:info-attempt`.

The `doc:DB` virtual document is a pattern for mapping to a MySQL relational database using the MySQLi module [15]. It specifies a connection to the database and an implicit transition to `sta:MySQLiErr` state in case of a connection error. In addition, the optional `query` attribute explicitly sets the SQL query that must be sent to the database after a successful connection. In this case, an SQL query is given to create a table in the database, if there is no such table (the query is not complete).

The `ent:videosAll` nested entry-element sets the display to all rows in the `video` table. The `ent:video-byId` entry element defines the display on one row of the `video` table, in which the `videoId` identifier field is equal to the specified value. The specified value is the value that is passed in the `videoId` HTTP Request parameter. Injection {{...}} provides the extraction of the value of the `videoId` parameter from the `arr:http` array and substitution into the database query string.

The `doc:YTVideos` virtual document illustrates mapping to an external web service for receiving JSON format data using the GET method. In this case, the display on the YouTube API V3 Videos service was set to obtain information about the video by its identifier [24]. Injection {{...}} extracts the `videoId` value from the `arr:http` array

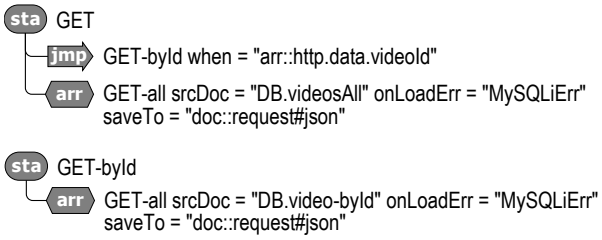

Fig. 4. Submodel states `sub:Http-proc` for GET-processing

and substitutes it into the web service request string as the `id` parameter.

The `jmp:further` transition continues processing in the next state, the name of which matches the name of the HTTP Request method.

C. GET processing states

The `sta:GET` and `sta:GET-byId` states are detailed in Fig. 4. The `sta:GET` state serves a situation where information about the entire database table is requested, while the `sta:GET-byId` state serves a situation where information about only one of the table rows is required. These situations are distinguished by the absence / presence of the `videoId` parameter as part of the HTTP Request [25]. The `jmp:GET-byId` element checks for the presence of this parameter as part of the HTTP Request and enters the `sta:GET-byId` state in this case.

In the state `sta:GET`, the associative array `arr:GET-all` is populated with the rows of the videos table through the virtual document `doc:DB.videosAll` (attribute `srcDoc`). In case of an error during the operation with the database, the transition to the `sta:MySQLiErr` state is performed (attribute `onLoadErr`). If the document is successfully loaded, its saving to the `doc:request` virtual document is executed (attribute `saveTo`). Saving in the `doc:request` document means that the contents of the associative array is returned as an HTTP Request Response. The option `#json` in the `saveTo` attribute instructs to form a Response as a JSON document. Thus, one `arr:GET-all` element defines three operations: loading a document; error handling; saving a document.

In the state `sta:GET-byId`, the associative array `arr:GET-all` is handled in the same way. The difference is that only one row of the `videos` table corresponding to the requested video is loaded and returned as a Response.

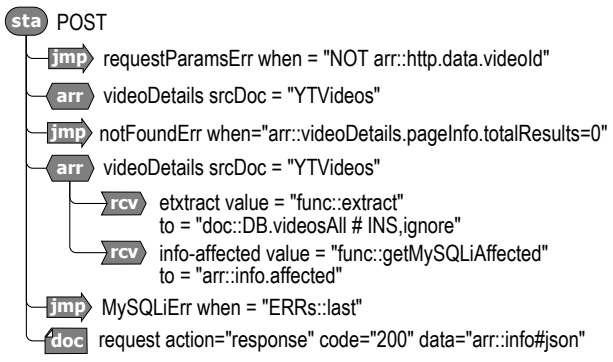

Fig. 5. Submodel states `sub:Http-proc` for POST-processing

D. POST, PUT, DELETE processing states

The `sta:POST` state serving the update situation of the database table is detailed in Fig. 5. A video identifier is needed to serve this situation. Therefore, the `jmp:requestParamsErr` element checks for the corresponding parameter in the HTTP Request. If the parameter is absent, then it enters the `jmp:requestParamsErr` state to handle the error.

Next, the `doc:YTVideos` virtual document containing information about the required video is loaded into the `arr:videoDetails` associative array. There may be a situation when an incorrect video identifier is transmitted with the HTTP Request. This situation is checked in `jmp:notFoundErr` by checking the `totalResults` field in the loaded document. If the field value is zero, then the error state is `sta:notFoundErr`.

Next, the document loaded in `arr:videoDetails` is processed using the `rcv:extract` and `rcv:info-affected` elements. The first element extracts the necessary information about the video and loads it as a new row into the database table. The `value` attribute indicates that you need to use the `extract` function to generate the information. The `to` attribute indicates that the generated information should be sent to the virtual document `doc:DB.videosAll`. The option `#INS, ignore` means that sending to the database is performed by the `INSERT IGNORE SQL` command. The `rcv:info-affected` element identifies the number of rows that were inserted in the table and writes this value to the `arr:info` virtual array. This is necessary because the `INSERT IGNORE` SQL command ignores the insert operation for cases where a record with this identifier is already in the table.

The `jmp:MySQLiErr` element checks for errors when performing operations with the database, in which case it enters the state `sta:notFoundErr`. If there are no errors, the normal completion of the `sta:POST` state processing is performed using the `doc:request` element. The `action = "response"` attribute instructs interpreter to perform an HTTP Request Response, and the `code` and `data` attributes specify the return code and returned data, respectively. The value of the `data` attribute instructs to return the contents of the associative array `arr:info` in JSON format.

The internal structure of `sta:PUT` and `sta:DELETE` states is very similar to `sta:POST`, so these states are not considered in detail.

E. Error Handling States

The states `sta:requestMethodErr`, `sta:MySQLiErr`, `sta:requestParamsErr`, `sta:notFoundErr`, serving special situations when errors are detected, are detailed in Fig. 6. Processing completion (HTTP Request Response) is performed in these states using the `doc:request` elements with the `action = "response"` attribute. The `code` attribute specifies the return code. The `message` attribute sets the returned message into which data about the detected error is inserted using the injection {{...}}.

V. Practical Example

Based on the presented pattern, a complex of microservices for monitoring the viewing of educational videos posted on YouTube was developed as part of research on the creation of a university digital educational environment. The viewing monitoring subsystem collects information about the comments that students made when watching videos, and on this basis generates analytical reports for both teachers and students. Microservices operate under the control of a research prototype of an interpreter of hierarchical situational models, which is hosted on the university's web server PHP platform [28–30].

Lower-level microservices provide CRUD operations with tables of the relational database of the viewing monitoring subsystem. Each microservice manages its own table. The main task of the microservice is to provide the ETL process for filling out its table (Extract–Transform–Load) using the POST method. In addition, the microservice provides operations for fetching (GET), updating (PUT), and deleting (DELETE) data. The composition of the database of the viewing monitoring subsystem is shown in Fig. 7, where the names of the microservices match the names of the tables.

The tables `videos`, `tries`, `playlists`, `playlist_video` are filled with information that the corresponding microservices retrieve from the YouTube web service through the API V3 (Application Program Interface):

- In the `videos` table, YouTube's information about educational videos that are monitored is recorded.
- The table `tries` recorded information about YouTube comments posted registered students when watching videos.
- The table `playlists` recorded YouTube information about playlists, views which are monitored.
- The table `playlist_video` YouTube recorded information on the composition playlists, views which are monitored (of which the movie is a playlist).

The tables `students`, `groups`, `prog_playlist` are filled with information that microservices receive from students, teachers or retrieve from the faculty database:

- The `students` table contains information about students who are entered by the microservice at the registration of each student.
- The `groups` table contains information about student groups corresponding to one study program. Microservice enters group information from the faculty database.
- The `prog_playlist` table contains information about playlists, videos of which students should watch when mastering the academic discipline as part of the study program.

Upper-level microservices use a database supported by lower-level microservices to generate analytical reports based on viewing monitoring. Reports targeted at both students and teachers are generated by microservices. The report intended for the student contains indicators of his activity as the lower level of granularity for individual videos, as well as aggregated indicators for playlists and the academic discipline. The report intended for the teacher includes additional aggregated indicators of activity at the level of student groups, study programs, etc.

To date, the viewing monitoring subsystem has been used in the educational process for more than one year, confirming its efficiency and usefulness. At the stage of development and implementation in the educational process, the proposed approach demonstrated the following advantages:

- Compact HSM model that performs the functions of processing HTTP Request. This follows from a higher level of abstraction of the declarative HSM model compared to traditional procedural programming languages. A smaller amount of program code is required to set the desired functionality, including for HTTP Request processing.
- Ease of pairing with existing applications, which is a consequence of the use of microservice architecture. For example, it was possible to quite simply connect the microservice of the students table with the previously developed application for registering students, without making any significant changes to it.

VI. Conclusions

Thus, for the hierarchical situational model of HSM used in situation-oriented databases, HTTP Request processing capabilities are provided. These capabilities allow you to both access the HTTP Request properties and generate an HTTP Request Response in the style used by HSM when processing other types of data (based on the concept of a virtual document and data processing objects). This allows you to

```
sta requestMethodErr
    doc request action = "response" code = "405"
        message = "Method Not Allowed: {{arr::http.method}}"
sta MySQLiErr
    doc request action = "response" code = "500"
        message = "Internal Server Error: {{ERRs::last.message}}"
sta requestParamsErr
    doc request action = "response" code = "404"
        message = "Bad Request: missing required params"
sta notFoundErr
    doc request action = "response" code = "404"
        message = "Bad Request: not exist {{arr::http.data.videoId}}"
```

Fig. 6. Submodel states `sub:Http-proc` for errors handling

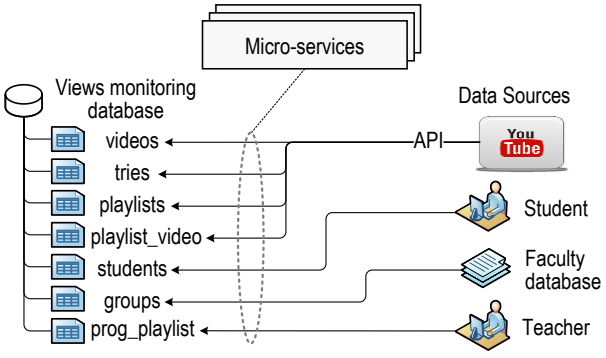

Fig. 7. Database tables and their data sources

build RESTful microservices based on situation-oriented databases. The practical use of this approach for monitoring the viewing of educational videos has confirmed its efficiency. At the same time, the advantages of the approach based on SODB and microservices, expressed in the compactness of the HSM model and ease of pairing with existing applications, have been demonstrated.

ACKNOWLEDGMENTS

This paper was supported by Russian Foundation for Basic Research grant No 19-07-00682.

REFERENCES

[1] V. V. Mironov and A. S. Gusarenko, 'Using of RESTful-Services in Situationally-Oriented Databases', Vestnik UGATU, vol. 19, no. 1 (67), pp. 232–239, 2015.

[2] 'What's a (micro)service - part 1?' [Online]. Available: http://chrisrichardson.net/post/microservices/general/2019/02/16/whats-a-service-part-1.html. [Accessed: 06-May-2019].

[3] 'What are microservices?', microservices.io. [Online]. Available: http://microservices.io/index.html. [Accessed: 06-May-2019].

[4] V. V. Mironov, A. S. Gusarenko, and N. I. Yusupova, 'Situation-Oriented Databases: Current State and Prospects for Research', Vestnik UGATU, vol. 19, no. 2 (68), pp. 188–199, 2015.

[5] 'Microservices Pattern: Microservice Architecture pattern', microservices.io. [Online]. Available: http://microservices.io/patterns/microservices.html. [Accessed: 06-May-2019].

[6] A. S. Gusarenko and V. V. Mironov, 'Smarty-objects: Use Case of Heterogeneous Sources in Situation-Oriented Databases', Vestnik UGATU, vol. 18, no. 3(64), pp. 242–252, 2014.

[7] V. V. Mironov, A. S. Gusarenko, and N. I. Yusupova, 'The Invariance of The Virtual Data in The Situationally Oriented Database When Displayed on Heterogeneous Data Storages', Herald of Computer and Information Technologies, no. 1(151), pp. 29–36, 2017.

[8] V. V. Mironov, A. S. Gusarenko, and N. I. Yusupova, 'Structuring virtual multi-documents in situationally-oriented databases by means of entry-elements', SPIIRAS Proceedings, vol. 4, no. 53, pp. 225–242, 2017.

[9] 'Microservices Pattern: A pattern language for microservices', microservices.io. [Online]. Available: http://microservices.io/patterns/. [Accessed: 06-May-2019].

[10] D. Dixon, 'PHP Microservices — Creating A Basic Restful Crud API', Medium, 02-Jan-2019.

[11] J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner, 'From monolith to microservices: A classification of refactoring approaches', Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11350 LNCS, pp. 128–141, 2019.

[12] D. Taibi, V. Lenarduzzi, and C. Pahl, 'Architectural patterns for microservices: A systematic mapping study', presented at the CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science, 2018, vol. 2018-January, pp. 221–232.

[13] M. Cavallari and F. Tornieri, 'Information systems architecture and organization in the Era of MicroServices', Lecture Notes in Information Systems and Organisation, vol. 24, pp. 165–177, 2018.

[14] R. Petrasch, 'Model-based engineering for microservice architectures using Enterprise Integration Patterns for inter-service communication', presented at the Proceedings of the 2017 14th International Joint Conference on Computer Science and Software Engineering, JCSSE 2017, 2017.

[15] A. S. Gusarenko, 'Improvement of Situation-Oriented Database Model for Interaction with Mysql', Journal of Instrument Egineering, vol. 59, no. 5, pp. 355–363, 2016.

[16] L. H. N. Villaça, L. G. Azevedo, and F. Baio, 'Query strategies on polyglot persistence in microservices', presented at the Proceedings of the ACM Symposium on Applied Computing, 2018, pp. 1725–1732.

[17] J. Donham, 'A domain-specific language for microservices', presented at the Scala 2018 - Proceedings of the 9th ACM SIGPLAN International Symposium on Scala, co-located with ICFP 2018, 2018, pp. 2–12.

[18] V. V. Mironov, A. S. Gusarenko, and N. I. Yusupova, 'Situation-oriented databases: document management on the base of embedded dynamic model', in 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. Moscow, Russia, 2016, pp. 238–247.

[19] V. V. Mironov, A. S. Gusarenko, and N. I. Yusupova, 'Integration of Virtual Multidocument Mappings into Real Data Sources in Situational-Oriented Databases', Applied Informatics, vol. 13, no. 3(75), pp. 47–60, 2018.

[20] A. S. Gusarenko and V. V. Mironov, 'Heterogeneous Document Sources in Situationally-Oriented Databases', Vestnik UGATU, vol. 19, no. 4, pp. 124–131, 2015.

[21] V. V. Mironov, A. S. Gusarenko, and N. I. Yusupova, 'Displaying virtual XML-documents on MySQL tables in the situation-oriented databases, "distributed approach"', Journal of Information Technologies and Computing Systems, no. 1, pp. 77–89, 2017.

[22] A. Mitrovic, V. Dimitrova, A. Weerasinghe, and L. Lau, 'Reflective experiential learning: Using active video watching for soft skills training', presented at the ICCE 2016 - 24th International Conference on Computers in Education: Think Global Act Local - Main Conference Proceedings, 2016, pp. 192–201.

[23] V. V. Mironov, A. S. Gusarenko, R. R. Dimetriev, and M. R. Sarvarov, 'The Personalized Documents Generating Using DOM-objects in Situation-Oriented Databases', Vestnik UGATU, vol. 18, no. 4 (65), pp. 191–197, Jun. 2014.

[24] E. Poche, N. Jha, G. Williams, J. Staten, M. Vesper, and A. Mahmoud, 'Analyzing User Comments on YouTube Coding Tutorial Videos', presented at the IEEE International Conference on Program Comprehension, 2017, pp. 196–206.

[25] H. Silva and I. Azevedo, 'Instructional videos and others on Youtube: Similarities and differences in comments', presented at the CSEDU 2017 - Proceedings of the 9th International Conference on Computer Supported Education, 2017, vol. 1, pp. 418–425.

[26] Y. Berkunskyi, K. Knyrik, T. Farionova, and T. Smykodub, 'Using microservices in educational applications of IT-company', presented at the 2017 IEEE 1st Ukraine Conference on Electrical and Computer Engineering, UKRCON 2017 - Proceedings, 2017, pp. 1208–1211.

[27] V. V. Mironov, A. S. Gusarenko, and N. I. Yusupova, 'Stream handling large volume documents in situationally-oriented databases', International Scientific Journal INDUSTRY 4.0. Scientific Technical Union of Mechanical Engineering "INDUSTRY 4.0", vol. 3, no. 5, pp. 240–244, 2018.

[28] S. Hoque and A. Miranskyy, 'Online and Offline Analysis of Streaming Data', presented at the Proceedings - 2018 IEEE 15th International Conference on Software Architecture Companion, ICSA-C 2018, 2018, pp. 68–71.

[29] S. Hoque and A. Miranskyy, 'Architecture for analysis of streaming data', presented at the Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018, 2018, pp. 263–269.

[30] A. S. Gusarenko, 'Model for creating documents in OFFICE OPEN XML format based on situationally-oriented databases', Applied Informatics, vol. 10, no. 3 (57), pp. 63–76, 2015.