Dynamic DOM-objects in situational-oriented databases: lingware and knoware of data sources

Golovkin Y. B.
Ph. D., Associate Professor. dep.
ACS,
FSEE HPE Ufa State Aviation
Technical University
450000, Ufa, st. Karl Marks, h. 12
phone +79174290226,
e-mail: valter-hartman@mail.ru

Gusarenko A. S.
postgraduate, junior researcher.
FSEE HPE Ufa State Aviation
Technical University
450000, Ufa, st. Karl Marks, h. 12
phone +79174290226,
e-mail: artyomgusarenko@gmail.com

Mironov V. V.
Doctor of Technical Sciences, prof.
dep. ACS, FSEE HPE Ufa State Aviation Technical University
450000, Ufa, st. Karl Marks, h. 12
phone +79174290226,,
e-mail: diplomastudents@mail.ru

Submitted: December 1, 2013; Accepted: December 4, 2013;
Published: December 9, 2013. UDC 004.65
Subject MSC mathematical classification: 90C35, 90C39
Subject ACM computer classification: I.6.5

Abstract

We solve the problem of development algorithmic support of situation-oriented databases for processing the data sources of various sizes required algorithms to implement the methods of processing XML-documents in SODB. Demonstrates the general case of processing, cleaning and filtering document XML, implemented in an interpreter. Proposed two approaches for processing XML, the first - stream processing, and the second - cache processing with loading the document into memory. In the model of embedded DOM-elements bound to a state of dynamic model in which the attributes are introduced specifying the processing and loading of XML-data. Terms entered data for the developed algorithms. An algorithm - head interpretation DOM-element model for the processing of DOM-elements. An algorithm for the interpretation of the elements sources, for processing nested items sources inside the DOM-elements. An algorithm for stream processing DOM-objects, that is, using the tools of XMLReader. The algorithm of the cached DOM-objects processing to use of technology DOM. For this type of software modules developed on the platform interpreter PHP.
The article shows the representation of the model developed for the processing of sources SODB. This representation - XML text for use in an information system. This model implements the example with students and taking subjects, when at the request of the student are given all of the items and the evaluation of the student, or vice versa, for the selected object displays all students to their scores. To handle files Student.xml, Subject.xml, Acadach.xml processing methods are used cached when the document is not loaded into memory, and sent to the stream, where it can be seen on each element individually to find the right, under this approach a return to the passed elements is not possible to return the required re-direct it to a stream. This approach helps to process a large volume of documents. The second approach - Cache processing with loading the whole document, this method is more convenient for small documents and return to the previous document nodes is possible.

**Keywords:** Web application, database, dynamic model; NoSQL; XML; DOM; PHP.

### 1 Introduction

Modern web applications that provide dynamic content to users based on the information stored on the web servers, determine the need for new approaches to the organization of the database. As an alternative to the traditional (relational) databases are actively developing XML-oriented databases, including within the movement NoSQL (Strauch & Kriha, n.d.; Kaur & Rani, 2013; Benzaken, 2013; Takahashi, 2013; Pokorný, 2013; He & Zhai, 2013).

In (Gusarenko, 2012; Gusarenko & Mironov, 2013) has been proposed and has evolved the concept of situation-oriented databases (SODB) intended for use in dynamic web applications. In SODB data format XML (Su-Cheng & Lee, 2009) are associated with states of internal dynamic model. For the dynamic model tracked its current state, and the data is processed in the context of states. This article deals with the automatic generation of XML-data associated with the states of the dynamic model in the linguistic and algorithmic plans.

### 2 Architecture of Situationally-Oriented Database

This article is further work (Gusarenko, 2013; Gusarenko & Mironov, 2013; Mironov & Gusarenko, 2013) in which the concept of the dynamic DOM objects automatically created and filled by a XML content in the course of interpretation of a dynamic model of SODB was developed. According to this concept DOM objects are bound to statuses of a dynamic model; DOM objects are created, boot, used for XML contents processing when appropriate statuses of a dynamic model become current. Unlike known approach where this functionality is reached at the expense of programming of appropriate functions in the subprogrammes stocks associated with statuses of a dynamic model, here:

1) at elements statuses of a dynamic model as child DOM elements at which, in turn, child elements sources set loaded XML data, and child elements receivers — saved XML contents are provided;

2) during interpretation of a dynamic model by the interpreter automatic creation of DOM objects for cur-rent statuses of model and loading of XML data with possible
conversion, and also automatic deleting DOM objects in case of change of current statuses is executed.

As a whole the concept is directed on that by development of a dynamic model to allow to specify flexibly in the declarative form the XML data required in these or those situations, and methods of their receiving from different sources, relieving of programming of appropriate functionality. In this article questions of implementation of this concept are considered.

**Key ideas.** In fig. 1 process of formation of XML contents of DOM object on the basis of information from data sources is illustrated. Each element source refers to a certain XML document, the document of the source which is storing in memory of associated data (ADM). It is used by a source as a basis for formation of XML data – result of the source intended for loading in DOM object. The result of a source is located in the parent document corresponding to XML contents of an element, parent in relation to an element source.

Sources can be nested each other. If the element source is directly nested in a DOM element, its parent document are DOM-object XML contents and if the element source itself is nested in other element source, its parent document is the result of a parent source.

In the elementary case the result of a source can represent a source XML-document (Case, 2006; Dekeyser, Hidders & Paredaens, 2004; Jea, Chang & Chen, 2009) contents. In more difficult situations it can be demanded to derive only certain, required XML elements (wanted) from the source document, to filter them according to the given conditions (cond), to clear their internal contents of ignored elements (ignory).

As a result required XML elements can represent a XML fragment which doesn't have a root element that can demand to conclude them in a XML element – a root (root).

In the elementary case the result of a source is located in a root element of the parent document as a child member. In more difficult situation it can be demanded to find a set of target elements (targs) in the parent document and to duplicate result of a source in each of them. XML technologies provide two basic approaches to document handling:

- stream processing – the XML document is processed by a flow in process of reading its nodes from the file. The document thus takes place in a random access memory not entirely, and parts. It allows to process very large volumes of data in the unidirectional mode. Instruments of stream processing: XMLReader and XMLWriter;

---

**Figure 2.1.** Formation of XML contents of DOM object on the basis of information from data sources
● the cached processing – before processing the XML document entirely boots in the structured look (in the form of a tree) in a random access memory of the computer. It gives the chance of flexible document handling of limited scope. Instruments of the cached processing – the DOM technology.

Both of these approaches can be applied when processing nodes in the data source document (Rose, Malaika & Schloss, 2006).

In implementation process of dynamic DOM-objects the linguistic support discussed below and including language means of the specification of data sources for loading, created DOM-objects was studied. Such means provide possibility of the job of data sources as on the chart of a dynamic model, in the graphic look oriented on perception by the person, and in text form, intended for perception by the interpreter.

Recognizing that routine of preparation and loading of dynamic DOM-objects it is laid to the interpreter, the appropriate algorithmic support discussed below and including algorithms of processing of those elements of a dynamic model which specify DOM-objects and data sources for them is developed. It was considered that necessary DOM-objects in model are set by means of DOM-elements, and elements – data sources can be nested (to form hierarchy), it caused the recursive nature of algorithms.

Check of operability of algorithmic support is realized by programming of the developed algorithms on the basis of language of server scenarios RNR with the subsequent testing on examples.

3 Linguistic means of the specification of data sources

The dynamic model of SODB is set by means of the formal HSM language (Hierarchical Situational Models) in the form of the graphic chart having the equivalent XML representation. Dynamic DOM objects are set as a part of model by means of DOM-elements and the elements enclosed in them – sources and data sinks. Syntax of these elements is explained by means of syntax charts on which the following designations are used: the meta character >> ; means "by determination"; OR meta character – an option choice (one of options shall be selected); the portable lines attached to an element from below – attributes of an element or its child members; dark circle on the portable line – mandatory single-digit attribute; light circle – optional single-digit attribute; light arrow triangle – optional many-valued attribute or a multiple child member; by italic type values of substitution are designated.

**DOM element.** The syntax chart of a DOM element is given in fig. 2. By determination this element on the graphic chart is represented the character containing a tag of "dom", with attributes and nested elements.

The DOM element can contain the following optional attributes specifying the initial XML document for loading in generated DOM object:

- **doc** – sets the XML document in the form of the link to a doc-element (in a dynamic model the doc-element with the appropriate name, containing in-information on finding of the necessary XML file shall be set);
- **path** – sets the XML document by explicit specifying of a way to the XML file.
If attributes are absent, after DOM object result instead of loading of the initial XML document the root XML element with a DOM-element name is created.

The DOM element can contain some child members – sources and data sinks:
- the element of data source sets XML result which will be placed in the generated DOM object (in a root element or in the specified elements beforehand the loaded initial XML document) as a child member (it is considered below);
- the element of a data sink sets XML result which will be placed in the document ADM or in a flow of output data as the response to request (in this article it isn't considered).

The element of data source represents the element of a dynamic model placed in a DOM element as the descendant (a nested element). It includes a set of the attributes which are setting properties of a source, and can contain, in turn, child elements sources (the recursive determination).

The syntax chart of an element-source is given in fig. 3. The element source is set by the character from well-aimed "src" to the right of which the source name is entered.

Attributes of an element source:
- targs – the optional attribute specifying in a parent XML document a set of target XML elements, will be attached to each of which as a child member XML result of a source. In the absence of attribute the XML result is attached to a root element of the parent document;
- Document_source – the mandatory construction which is setting the document from which XML contents undertake. This construction represents one of three attributes:
  - path – sets the source document by explicit specifying of a way to the XML file;
  - doc – sets the document of a source in a type of the link to a doc-element (in a dynamic model the doc-element with the appropriate name, containing in-formation on finding of the necessary XML file shall be set);
  - dom – sets XML contents of DOM object which is generated by a DOM element with the entered name as the document;
Figure 3.2. Syntax chart element source

- root – the optional attribute ordering in addition to conclude XML result of a source in tags with the given name (it can be convenient when the XML fragment which doesn't have a root element) is de-rived from the document of a source;
- wanted – the optional many-valued attribute setting a name list of XML elements, found in the source document for result formation (the list of the values separated by gaps as it is accepted in XML). In the absence of attribute the result is created of a root XML element of the document of a source;
- ignory – the optional many-valued attribute setting a name list of XML elements in the document of a source which should be ignored when forming result;
- cond – the optional many-valued attribute setting the list of conditions with which shall satisfy required a source XML document element (from the list in the wanted attribute) to get to result. In the absence of attribute all found XML elements get to result from the wanted list. Syntax of the job of conditions is discussed below;
- instrument – the optional attribute, ordering to use for reading the document of a source of XMLReader – a method of stream (not cached) access. In the absence of a default attribute the method of cached access in case of which the source document entirely is located in buffer DOM object is applied.

For process control of filtering XML elements of data source the algorithm is provided the many-valued cond attribute. It contains the list of the conditions partitioned by gaps in a look

\[
\text{term == term,}
\]

where the left term sets checked value in a required XML element of a source, and right – in a target XML element of the parent document. Each term contains a prefix and a basis separated by a colon. The prefix specifies to the interpreter, what method to select values in case of check of an appropriate condition. The following prefixes are provided:
● att: – value of attribute of a XML element is checked, the basis sets an attribute name;
● ele: – value child XML element is checked, the basis sets member name;
● xpath: – the value addressed by XPath-expression is checked, the basis sets expression;
● var: – variable value of the dynamic model which value is stored in memory of a current status (CSM) is checked, the basis sets a variable name;
● glb: – value of global variable of the environment of interpretation is checked, the basis sets a name of global variable;
● pst: – Post value - a variable value from client request is checked, the basis sets a Post-variable name;
● ses: – value of the sessional (session) variable environment of interpretation is checked, the basis sets a name of a sessional variable.

Each element source, in turn, can contain one or several internal ales-cops-sources creating child XML-elements as a part of result of a source. Depth of an enclosure isn't restricted.

Graphic representation of a dynamic model (and in particular, DOM-objects) in the form of the chart is used as means of evident design. For use in an information system on the basis of the chart the equivalent text representation in the form of a XML code processed by the interpreter is built.

The fragment of a dynamic model containing a DOM-element, in text form has the following structure:

<dom:Name_DOM-element doc="Name_doc-element">
  <src:Name_source doc="Name_doc-element" … >
    …
    <src:Name_source doc="Name_doc-element" … >
      …
    </src:Name_source>
  </src:Name_source>
</dom:Name_DOM-element>,

where elements and attributes register according to XML syntax.

4 Algorithmic support

4.1. Head algorithm DOM-element interpretations

Algorithmic support here is understood as set of algorithms according to which the interpreter executes processing of a fragment of the dynamic model containing a DOM-element. DOM-elements settle down in a dynamic model as child members of elements statuses therefore their processing is executed in a context of processing of appropriate parent statuses.

The flowchart of head algorithm of processing (interpretation) of a DOM-element – DOMProc – is given in fig. 4. The "hierarchical" method of representation of the flowchart used hereinafter has differences from the traditional. The input in the next unit
The called algorithm begins (unit 1) with receiving a name processed DOM element (unit 2) and checks of existence of DOM object with such name (unit 3). In case of absence the DOM-object is created with a root element (unit 4). Then (unit 5) existence at a DOM-element of the doc or path attribute is checked and in this case bootstrap loading in DOM object of the relevant document (unit 6) is executed. For DOM object the pointer on its root element (unit 7) is created. Further in a cycle are processed (the unit 8) all child members of a DOM element (sources and data sinks). Whether for each child member it is checked, is a source (unit 9), and in this case for it the algorithm of processing of data source (unit 10) is caused, whether is it the receiver (unit 11), and in this case for it the algorithm of processing of a data sink (unit 12) is caused. By a call of algorithm of processing of a source or the receiver pointers are transferred to a root element of DOM object and to a processed element (source / receiver). Upon termination of a cycle (the unit 8) algorithm comes to the end and executes return to invocation point (unit 13). The called algorithm begins (unit 1) with receiving a name processed DOM element (unit 2) and checks of existence of DOM object with such name (unit 3). In case of absence the DOM object is created with a root element (unit 4). Then (unit 5) existence at a DOM element of the doc or path attribute is checked and in this case bootstrap loading in DOM object of the relevant document (unit 6) is executed. For DOM object the pointer on its root element (unit 7) is created. Further in a cycle are processed (the unit 8) all child members of a DOM element (sources and data sinks). Whether for each child member it is checked, is a source (unit 9), and in this case for it the algorithm of processing of data source (unit 10) is caused, whether is it the receiver (unit 11), and in this case for it the algorithm of processing of a data sink (unit 12) is caused. By a call of algorithm of processing of a source or the receiver pointers are transferred to a root element of DOM object and to a processed element (source / receiver).

![Figure 4.1. Head algorithm of interpretation DOM elements](image-url)
Upon termination of a cycle (the unit 8) algorithm comes to the end and executes return to invocation point (unit 13).

4.2. Algorithm of interpretation element source

The flowchart of algorithm of processing of an element source is given in fig. 5. The algorithm (unit 1) begins with targs attribute verification of presence at a processed element source (unit 2) to define a method of formation of an array of target elements of the parent document. In the presence of attribute the array of target elements is created in a search result in the parent document of a set of elements with the names given in the targs attribute (unit 3). Otherwise the set of target elements includes the single root element of the parent document (unit 4).

After that cyclic processing on each target element (unit 5) is executed. Root attribute existence at a processed element source (unit 6) is checked and in this case the child member at a processed target element which will be used further instead of a target element (unit 7) is created. Instrument attribute existence = by "XMLReader" (the unit 8), a setting method of processing of nodes of the document of a source is checked. Depending on it the algorithm of stream processing – SrcReaderProc (unit 9) or the cached processing – SrcDOMProc (unit 10) is caused. Upon termination of a cycle (unit 5) the algorithm comes to the end and executes return to defiant algorithm (unit 11).

4.3. Algorithm of stream processing elements of the document of a source

The flowchart of algorithm of stream processing of elements of the document of a source is given in fig. 6. The algorithm (unit 1) begins operation with initialization of the tool of stream input of XMLReader (unit 2) that provides opening of a flow of input for sequential reading of tags of the document of a source. Further in a cycle (unit 3) is executed stream reading a node behind a node from the source document until the document won't end. For processing the nodes being XML elements (unit 4) are selected. It is checked, whether there is at an element source a wanted attribute (unit 5), and whether in this case the processed node is checked, is required a source XML document element (units 6-11). Check begins with reset of a flag of coincidence (unit 6).

After that in a cycle (unit 7) names from the list given in the wanted attribute get over, are compared to a name of a processed node (the unit 8) and in case of coincidence is set a flag (unit 9). Upon completion of a cycle (unit 7) is checked flag activation of coincidence (unit 10) and if the flag isn't set (coincidence isn't recorded), processing of this node (unit 11) compulsorily comes to the end and transition to following processing (in the unit 3) is executed. If the processed node is recognized required, it is checked on execution of conditions of filtering which are set in the cond attribute in the form of conditions on at-tributes of a processed node (units 12-14). Such conditions can be checked directly, without reading of the following nodes of the document from an input flow. For this purpose in a cycle conditions of this kind are derived from the cond attribute (unit 12) and checked for a processed node (unit 13). In case of detection of an outstanding condition processing of this node compulsorily comes to the end (unit 14).
and transition to following processing (in the unit 3) is executed. If the processed node meets conditions on attributes, it is checked on execution of other conditions of the filtering given in the cond attribute (units 15-18). Such conditions require the appeal to other nodes of the document from an input flow.

**Figure 4.2.** Algorithm of stream processing elements of the document of a source

**Figure 4.3.** Algorithm of the cached processing elements of the document of a source
For this purpose the processed element (together with the internal contents) is derived in a cache (unit 15). Further in a cycle conditions of this kind are derived from the cond attribute (unit 16) and checked for the processed node being in a cache (unit 17). In case of detection of an outstanding condition processing of this node compulsorily comes to the end (unit 18) and transition to following processing (in the unit 3) is executed.

If the processed node successfully passed all checks, it joins (Tang, Yu, Wong, & Li, 2008; Han, 2014; Kudrass & Conrad, 2002; ) as a child member a processed target element of the parent document (unit 19).

Further cleaning of the associated child member from the internal elements which names are set in the ignory attribute (units 20-22) is executed. For this purpose names from the ignory list are viewed in a cycle (unit 20) and for each of them cyclic search of internal elements (unit 21) with deleting (unit 22) is executed. After the processed node of the document of a source is checked, connected to a target element and is cleared, for it internal data sources (units 23, 24) recursively are processed. The cycle on elements sources, child in relation (Batory, 2006) to processed (unit 23) is for this purpose executed. For each internal source the algorithm of its processing (the unit 24, see fig. 5) recursively is caused, and as a target element the processed node is transferred.

On the last step of processing of a node wanted attribute existence (unit 25) again is checked and in case of its absence (unit 24) forced completion of an operation cycle of the document is executed. It becomes that it will in this situation be restricted to processing of a root element of the document of a source. Upon termination of a main loop (unit 3) the algorithm comes to the end and executes return to defiant algorithm (unit 27).

4.4. Algorithm of the cached processing elements of the document of a source

The flowchart of algorithm of the cached processing of elements of the document of a source is given in fig. 7. As a whole it is close to the flowchart of stream processing though it is slightly simpler in a type of great opportunities for the cached processing of XML documents.

The algorithm (unit 1) begins operation with cache preparation in the form of DOM object with the XML document (O’Connor, Merriman & Nelson, 2010; Chen & Huck, 2001; Herskovits & Chen 2008; Nassis, Dillon, Rajagopalanpillai, & Rahayu, 2006; Sladic, Milosavljevic, Konjovic & Vidakovic, 2011; Sudarsan & Gray, 2006) of a source loaded in it (unit 2). Thus, if the DOM object to which the element source refers, already exists, it is used as a cache; otherwise the new DOM object in which the source document boots is created.

Further the array of links (unit 3) to required elements of the document of the source which names are set in the wanted attribute of an element source (in case of absence at a wanted attribute element source in an array is created the single link to a root element of the document of a source) is skidded. Further by a cycle on an array of links (unit 4) is checked each required element (units 5-13). First of all, execution of conditions of filtering which are set in the cond attribute (units 5-7) is checked.
For this condition in a cycle are derived from the cond attribute (unit 5) and are checked for a processed element (unit 6). In case of detection of an outstanding condition element processing compulsorily comes to the end (unit 7) and transition to following processing (in the unit 4) is executed. If the processed node successfully passed checks, it joins as a child member a processed target element of the parent document (the unit 8).

Further cleaning of the associated child member from the internal elements which names are set in the ignory attribute (units 9-11) is executed. For this purpose names from the ignory list are viewed in a cycle (unit 9) and for each of them cyclic search of internal elements (unit 10) with deleting (unit 11) is executed. After the processed node of the document of a source is checked, connected to a target element and is cleared, for it internal data sources (units 12, 13) recursively are processed. The cycle on elements sources, child in relation to processed (unit 12) is for this purpose executed. For each internal source the algorithm of its processing (the unit 13, see fig. 5) recursively is caused, and as a target element the processed node is transferred. Upon termination of a main loop (unit 4) the algorithm comes to the end and executes return to defiant algorithm (unit 14).

4.5. Implementation on the PHP platform

Implementation of the developed algorithmic support was carried out as a part of the interpreter of dynamic models of SODB on the PHP platform. The PHP-version of algorithms of interpretation of DOM elements, elements sources and elements receivers in the form of the PHP functions, caused the friend from the friend, and also from other functions of the interpreter is developed.
The standard opportunities of the stream and cached XML processing provided in PHP 5 (the libxml and libxslt libraries supporting extensions of DOM, XMLReader, XMLWriter) were used. The modular structure of the software in respect of functions of dynamic DOM objects is explained in fig. 8.

The listing given below illustrates a code of the dynamic models used in case of testing of the developed software. The dynamic model is made according to the chart used on a conceptual phase of dynamic DOM objects (Dejanovic, Milosavljevic, Perisic & Tumbas, 2010; Panach, Juristo & Pastor, 2013). It services the web application, displaying to the user of convergence about students and subjects studied by them, and includes three levels of hierarchy, 3 submodels, 5 statuses. For formation of output pages 4 DOM objects (in listing are highlighted in bold type), which XML content is dynamically created of 3 initial documents depending on the current situation are used.

**Listing.** Test dynamic model with DOM elements and elements sources

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sta:Students-Subjects xmlns:sta = 'HSM' xmlns:jmp = 'HSM' xmlns:sub = 'HSM' xmlns:act = 'HSM' xmlns:mnu = 'HSM'
xmlns:dom = 'HSM' xmlns:btn = 'HSM' xmlns:use = 'HSM' xmlns:inp = 'HSM' xmlns:div = 'HSM' xmlns:var = 'HSM'
xmlns:txt = 'HSM' xmlns:sec = 'HSM' xmlns:src = 'HSM' xmlns:rcv = 'HSM' xmlns:doc = 'HSM' xmlns:att = 'HSM'>

<doc:ListStudents path="XML/stud.xml" />
<!-- XML-documents are in use -->
<doc:ListSubjects path="XML/predm.xml" />
<doc:Progress path="XML/sdacha.xml" />
<doc:ListStudents path="XSL/stud.xsl" />
<doc:ProgressStudent path="XSL/uspevstud.xsl" />
</sta:Students-Subjects>
```

**Figure 4.5.** Modular structure of the software of dynamic DOM objects
<div:Students-Subjects section = "sec:StudentsSubjects"/>

<!-- Immersing в Submodel Students-Subjects -->

<sec:StudentsSubjects>
  <sub:Students-Subjects> <!-- Submodel Students-Subjects -->
    <sta:Students> <!-- State Students -->
      <act:Students pass="2"> <!-- Action -->
        <rcv:echo method="label" value="Students"/>
      </act:Students>
    </sta:Students>
    <jmp:Subjects> <!-- transition by press a button -->
      <btn:Subjects caption="To subjects"/>
    </jmp:Subjects>
    <div:Students section = "sec:StudentsSubjects"/>  <!-- Immersing в Submodel Students -->
  </sub:Students-Subjects>
  <sta:Subjects> <!-- State Subjects -->
    <act:Subjects pass="2"> <!-- Action -->
      <rcv:echo method="label" value="Subjects"/>
    </act:Subjects>
    <jmp:Subjects> <!-- transition by press a button -->
      <btn:Subjects caption="To students"/>
    </jmp:Subjects>
    <div:Subjects section = "sec:StudentsSubjects"/>  <!-- Immersing в Submodel Subjects -->
  </sta:Subjects>
</sec:StudentsSubjects>

<sub:Students> <!-- Submodel Students -->
  <sta:ListStudents> <!-- State ListStudents -->
    <dom:ListStudents> <!-- DOM-element ListStudents -->
      <src:ListStudents doc="ListStudents" instrument="XMLReader"/>
      <rcv:echo pass="2" method="xslt" styleSheet="doc:ListStudent"/>
    </dom:ListStudents>
    <jmp:SelectedStudents targs="SelectedStudents"> <!-- Transition by press a button -->
      <btn:SelectStudent caption="Select student"/>
    </jmp:SelectedStudents>
  </sta:ListStudents>
  <sta:SelectedStudents> <!-- State SelectedStudents -->
</sub:Students>
Variable in memory current state

DOM-element ProgressStudent

DOM-element ListSubjects

DOM-element ProgressInSubject

DOM-element SelectedStudent

DOM-element DeliveryInSubject
Conclusion

In this article results of development of linguistic and algorithmic support of data sources as a part of the dynamic DOM objects which are set in dynamic models of situation-dependent of oriented databases are provided.

The developed linguistic support in the form of syntax charts of DOM elements and nested elements sources as a part of elements statuses of a dynamic model differs existence of attributes for the job of conditions of selection and filtering elements of the document source connected to elements of the parent document during loading in DOM object.

The developed algorithmic support in the form of flowcharts of algorithms of interpretation of DOM elements and elements sources differs that depending on the specification of a dynamic model two modes of processing of nodes of documents of data sources – stream and cached are provided.

Algorithmic support is implemented in the form of PHP-modules as a part of the interpreter of dynamic models situation-dependent the oriented database. Test examples of dynamic models for the web application showed operability of offered approach to the organization of dynamic DOM objects.

Further researches are supposed to be continued in the direction of research of performance measures, first of all, time of filling of dynamic DOM objects a XML content of data sources.

Acknowledgments

The paper carried out as part of research supported by the RFBR grant 10-07-00167-a.

References


presented at the Proc. 8th Workshop of winter school postgraduates and young researchers, Ufa: USATU.


Mironov, V. V., & Gusarenko, A. S. (2013). Profiling algorithm of situationally-oriented database based on dynamic DOM-objects (Vol. 1, pp. 115–118). Presented at the Proc. 8th Workshop of winter school postgraduates and young researchers, Ufa, Russia: USATU.


Takahashi, K. (2013). Design and evaluation of lifelog mashup platform with NoSQL database (pp. 133–139). Presented at the Proc. of 15th International Conference on Information Integration and Web-Based Applications and Services, iiWAS’2013, Vienna, Austria.


