Situationally-Oriented Databases: The Concept of Managing XML Data Based on Dynamic DOM-Objects

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: August 28, 2013; Accepted: September 28, 2013;
Published: December 9, 2013. UDC 004.65
Subject MSC mathematical classification: 90C35, 90C39
Subject ACM computer classification: I.6.5

Abstract

The article considers the concept of XML-data management in a dynamic model of the web application, as is often required to specify the processing operations of XML-documents requires the server and reduce the complexity of data processing and issuing them to the user. Manual programming is required. Standard data processing operations is proposed to implement the interpreter. The basis of the situation-oriented embedded database library functions, the current state of the memory, the memory associated data, the interpreter embedded dynamic models SOBD. It is proposed equipped situationally-oriented database linguistic means of processing and interpreting the data source XML. Processing operations implemented in the programming language interpreter. Processing of XML-documents is based on the specification of dynamic DOM-objects in the model of DOM-elements to create them in the process of interpretation the model, in which the data is loaded, and then filtered or purified. Finished DOM-objects are created when the state becomes the current model, the interpreter of dynamic models automatically downloads their data and deletes DOM-object entire when the state is no longer be current. Data for the
formation of DOM-objects can be processed by two methods of copying and cutting. To load the data provides for the use of external data source, loading of functions, downloading from another DOM-object, load filtering XML. Separately, we consider a method of transformation of DOM-objects. An example of using dynamic DOM-objects in a web application based on situation-oriented database. An example of the transformation of the data receiver finished user interface. Understand the specific data model in the DOM-processed objects. We discuss the scope of DOM-objects. In the example shown submodels of different levels, macro complex data sources.

**Keywords:** Dynamic model; interpretation; situationally-oriented databases; XML; DOM; XSLT.

1 Introduction

Currently in the field of Information Technology is actively developing an approach to the development of information systems based on the direct use of models of high-level abstractions (Model-based, model-driven engineering) (Gusarenko, 2012; Gusarenko & Mironov, 2013). In line with this approach can be considered and the idea of situation-oriented databases, in which data is in XML format are associated with states of internal dynamic model and can be processed in the context of its current states. Situationally-oriented databases can serve as the basis of Internet applications, providing a dynamic form content in accordance with the situation. In this regard, important question is the organization of processing XML-data associated with the current state of the dynamic model discussed in this article.

2 Architecture of Situationally-Oriented Database

In the situationally-oriented database includes the following components (Figure 2.1):

- HSML (HSM Library) — library of dynamic models, which contains a set hierarchical models of situational HSM (Hierarchical Situational Models) as a hierarchy of transition graphs with a finite number of states (Finite State Model);
- CSM (Current State Memory) — current state of the memory that stores information about the current state of dynamic models;
- ADM (Associated Data Memory) — associated data memory that stores the XML data, correlated (associated) with the different states of the dynamic model;
- AFL (Associated Functions Memory) — library associated functions to store the data processing functions, correlated with the states of the dynamic model;
- HSMI (HSM Interpreter) — interpreter of a dynamic model that, in response to an external query Q generates R a response by processing a dynamic model of the HSM from HSML by monitoring its current state stored in CSM, processing associated data from the ADM and perform associated functions of the AFL.

So if situationally-oriented database is used by the Web server as the basis for Internet applications, as an input query Q serves a set of parameters obtained with the URL (for example, the parameters form in POST), and as a result of R - return HTML code to be sent to clients. Request parameters specify a dynamic model processed and the need to change its current state. In Fig. 2.2 shows an example of a dynamic model of
HSM. Root condition of sta:S0, symbolizing model in general has children of three elements: two definitions of the XML documents from the ADM and a submodel of HSML. The first document, doc: X1, a file «X1.xml», and the second, doc:X2, - file «X2.xml»), both of the folders «XML» ADM. Submodel sub: M sets the parent state sta:S0 two substates: sta:S1 and sta: S2. Child elements of the state sta:S1 and sta:S2 contain, first, the elements of the transitions jmp:S2 and jmp:S1, providing the current state of change, and secondly, the elements of action-act: A1, and act: A2, provided for specific actions in the corresponding states.

In the process interpretation of the elements-stock run the associated function from the AFM, which can include the processing of XML documents from the ADM. For example, can be provided for the creation of DOM-objects, load them in the XML documents, XSL transformation of the contents, the output result of the transformation as a result of the query. Processing of associated data, therefore, is programmed to associate features from the AFM, outside of HSM, making it difficult to understand the logic of data processing. Specification of processing XML documents can be distributed over different distances, with HSM. For example, in a given state of loading XML document, and its modification specified in one of the substates of this state, save the changes in another, and an output result in the third. To understand the logic behind this processing, it should be analyzed together several functions associated with multi-state. In this regard, it would be helpful if the logic of processing XML documents from the ADM can be explicitly specify in the HSM.

![Architecture of situationally-oriented database](image1)

**Figure 2.1.** Architecture of situationally-oriented database

![Diagram of dynamic model including definitions associated with XML documents](image2)

**Figure 2.2.** Diagram of dynamic model including definitions associated with XML documents
**DOM-objects.** DOM (Document Object Model) — it is platform-independent object-oriented interface to access documents XML, XHTML, HTML, are now receiving widespread. The technology of processing XML data based on DOM provides for DOM object, loading it entirely external XML document (or create a new document "from scratch" programmatically), the manipulation of the document as a tree, save the resulting document in the external environment.

Taking into account the opportunity, flexibility, the prevalence of DOM, it seems appropriate to solve the problem under consideration, first of all, this technology. The considered problem. Therefore, it is advisable give developers the ability to specify HSM DOM objects associated with the current states of the model. During the interpretation of the HSM according to these specifications, the interpreter should automatically create a DOM objects, load them in the XML data from the ADM, making them accessible to the processing from other elements of the model corresponding to the current state, save the modified XML data in the ADM, delete objects when they are no longer required. It should be provide the ability to specify at the level of HSM transformation XML data contained in DOM objects. It should be provide the ability to specify at the level of HSM transformation XML data contained in DOM objects. For example, a developer can specify the model that in some state of XML data is loaded into a DOM object, subject to XSL transformation according to a certain XSLT stylesheet. Objects with such behavior is called dynamic DOM-objects.

3 The concept of Dynamic DOM objects

3.1. Dynamic creation of DOM objects

So, the interpretation of the dynamic model can be automatically created DOM objects that contain XML data associated with the current states of the model and available for processing from other model elements associated with the current state. For this purpose a special model of DOM elements with the specified sources for these XML data. Accordingly, in the process of interpretation models envisage handling of DOM elements: the creation of the corresponding DOM objects and loading them into XML data from various sources, when the parent becomes the current state, removal of DOM objects when the parent states are no longer current. This allows to handle XML data associated with the current state, referring to the DOM objects of the procedures specified in the elements of the current state-actions. This should decrease the complexity of programming, as a routine preparation and loading DOM objects placed on the interpreter.

Implementing this idea requires a conceptualization of issues that will be created as a DOM objects and loads XML data about the availability of DOM created objects to be processed in different states, the removal of objects and DOM saving in the ADM of the contents of XML. It may be prompted several options for filling the data DOM objects created in the interpretation of DOM elements: from an external XML file, from another DOM object model, as a result of a function, etc. In Fig. 3.3 shows how the different variants of the data sources are represented in the diagram model.
Figure 3.3. Assignment of source XML-data from ADM (a), as another DOM-element (b), as a result of function (c)

DOM element is represented by the symbol $\text{dom}$, the right of it, as usual, the name of the element (in this case, DOM elements $\text{dom:D1}$, $\text{dom:D2}$, $\text{dom:D3}$). Symbol of the data source $\text{src}$, attached to the DOM element as a child element that specifies the features of loading data into XML DOM object generated by the DOM element. The attributes source element, in turn, specify, where are the XML-data.

**Load from external XML file.** In Fig. 3.3, a diagram shows a fragment of the model on which to DOM element is specified a data source as XML document from ADM. The attribute type = "doc" indicates that the XML document of the ADM, as the data source name «X1» - a reference to the definition of doc: X1. The attribute method = "copy" means that the content of the document doc: X1 will be fully copied into the newly created DOM object. When processing dom:D1 will be created DOM object «D1», and in the processing of src:X1 according to the doc:X1 it will be downloaded the file X1.xml.

**Load from function.** In Fig. 3.3, the diagram shows a fragment of the model on which to DOM element is specified the data source as a function that returns a string as the result of XML-data. As indicated by the attribute type = "func", i.e., the data source name «F1» - a reference to the function of the AFL. During processing, the interpreter calls the specified function, and loads the result of its performance XML line created in the DOM object.

**Load from another DOM object.** In Fig. 3.3 b shows a fragment of the diagram model, which for the DOM element is specified the data source as a reference to another DOM element. As indicated by the attribute type = "dom", i.e., the data source name «D1» - a reference to the definition of dom: D1. Assumed that the at the time of processing element src: D1 DOM element dom: D1 has already been processed by the interpreter, i.e. created and loaded DOM object «D1». The interpreter accesses the DOM object «D1» and copies the contents of the XML DOM a new object «D2».

**Load, provided filtering XML data.** In real conditions may require not just copy the XML data from the source data, and perform with a particular conversion. So often you want to filter the data, i.e., the load in the DOM object only the part of the data that meets certain conditions. In Fig.3.4 illustrates the filtering of the data source as XML-file from the ADM. In the dynamic model shown in Fig. 3.4, and at the root of the state of sta:S set definition XML document doc: X1, stored in the ADM, and DOM element dom: D3, which provides for downloading a document doc: X1 with filtration. Data Source src:X1 has an attribute method = "cut", indicating that the source of the
XML-document will be "cut" some subtree. Additional attributes «element», «field» and «value», set the filtering condition. Attribute «element» includes XPath-expression that defines a set of element nodes in XML-tree, one of whom will upload the root of the subtree. Attribute «field» contains XPath-expression that specifies a subtree validated XML-element or attribute. Attribute «value» contains the desired value. In this example, you want to load in the DOM object subtree, starting in a «e1>>, the attribute «k1> which has the value "123". Fig. 3.4 b shows the data model of XML-doc: X1 in a graphical notation. The root element of XML-<E0> can contain multiple child XML-elements «e1>>, each of which contains the attributes of «k1» (ID) and «a1». Thus, XPath-expression «/ E0/e1>>, specified in the attribute «element» the source src:X1 in Fig. 3.4, and, addresses the set of all elements of the XML-«e1>>; attribute «field» addresses in the "e1» XML-attribute «k1>>, and the attribute of «value» for it sets the desired value of "123." During the processing element dom:D3 interpretation refers to the power commutator src:X1, XML iterates the elements of «e1>>, searching for the only one with the XML attribute «k1> has the desired value of "123", and loads the corresponding subtree in the DOM object. As a result, the DOM object «D3» will be loaded XML data corresponding to the model shown in Fig. 3.4 b.

3.2. Load, providing transformation of XML data

In a more general case transformation of XML data loading can be regarded as XSL transformation of XML document and data source for the solution of applied technology XSLT (Figure 3.5). Element of a data source in this case contains an attribute method = "xslt", which informs the interpreter the need for XSL transformation of data, as well as the attribute <stylesheet>, referring to using this style sheet, and the attribute «params», which contains a list of values for global parameters transfer of the style sheet. During the processing of the data source determines the need for an interpreter XSL transformation, load style sheet, passes it global parameters, performs the transformation of XML data source and loads the result into DOM created object. In this example, by using XSLT solved the same problem uploading with the filtration as in the previous example (see Fig. 3.4). Conceptual model of XSL transformation XML source data (Fig. 3.5, b) is presented in a graphical notation. The transformation involves finding an element in the source document «e1>>, which has the at-tribute «k1» equal to the value of global parameter «prm», and the issuance of the element found with the subtree as a result of transformation.

![Figure 3.4](image)

**Figure 3.4.** Example loading of DOM object with filtering XML data
3.3. Load, providing merging of XML data

In a more sophisticated case might need to load an object in the DOM XML data obtained by converting multiple source XML documents. For example, the general information contained in one document is required to supplement the detailed information from another document. In Fig. 3.6 illustrates the source, providing XML merge documents. The model of the first of the merged documents containing general information and acting as a parent, is shown in Fig. 3.6, and (see Fig. 3.5). The model of the second of the merged document containing detailed information and acting as a child, is shown in Fig. 3.6 b. Model of the resulting document loaded in the DOM object shown in Fig. 3.6 in. The corresponding fragment of the dynamic model, providing merger of the two documents when loading DOM object «D3», is shown in Fig. 3.6, the element has an attribute of the source method = "merge", communicating with the interpreter, that are loaded into DOM XML object data obtained by merging two XML documents. These documents set the attributes perentDoc (parent document) and childDoc (child of the document). Other attributes define XPath expression, specifying merge features:

- parentElement points in the parent document, the set of XML-elements to which the subtrees will be attached, taken from the child document;
- parentField indicates for each element of the previous sets (parentElement) value used for the identification affixed subtree;
- childElement indicates in a child document, the set of elements which, together with their subtrees are used to copy to the parent document;
- childField indicates for each element of the previous sets (childElement) value used for the identification affixed subtree.

During the processing element of the source interpreter refers to the parent document, and for each of its XML-type element parentElement finds in the child document corresponding to it element of the XML-type childElement such that the values and parentField childField match. The corresponding element found with his subtree is copied to the parent document as a child of the processed XML-element parentElement.

3.4. Dynamic deleting of DOM objects

The principle, which underlies the dynamic removal of DOM objects, is, DOM object that is automatically created when processing a DOM element of the current state of the dynamic model should be automatically removed when a state is no longer current.
This demand can be simply implemented by epilogue processing conditions under the general interpretation of the algorithm. Epilogue processing is executed by the interpreter each time before changing the current state and consists in that the interpreter recursively perform the subtree condition has ceased to be current, with a special feature (flag) epilogue mode. Using this attribute, child elements, the state may provide in the course of processing epilogue any actions that must be done before changing the current state. Thus, the processing elements in the DOM epilogue mode, the interpreter should automatically delete the corresponding DOM objects.

3.5. Save XML data of DOM-objects

The contents of the XML DOM objects can be changed in the process of interpretation (e.g., following a request from the user), then you may need to save them in the ADM. Consequently, it is necessary to provide the ability to set the HSM procedures save the contents of DOM objects.

**Element receiver.** Because the saving procedure is reversed with respect to the procedures loading DOM object model, we introduce an element of "sink." For his images in the chart model we use the symbol  \( \text{rcv} \) (as for a source, but with the opposite direction and mark \( \text{rcv} \)). Element receiver is illustrated in Fig. 3.7, where it is defined by means of maintaining XML DOM data objects dom:D1, dom:D2 and dom:D3 in the target XML document doc:X1, doc:X2, doc:X3 in the ADM. The attribute «method», as in the case of the data source tells the interpreter what data DOM object will be stored: method = "copy" means that you must keep all XML contents of the whole; method = "cut" - the need to save XML subtree of the DOM object; method = "xslt" - it is necessary to save the output XSL transformation of the contents of DOM object. Attribute «target» defines a way to combine the stored data to those already in the target document: target = "create" means that you must re-create the XML contents of the destination document (if the file already exists, it is replaced by the created file); target = "insert" - that you want to insert XML stored data in the target document tree (set the path to the parent element of the target document); target = "update" - it is necessary to replace XML persisted data for a subtree of the target document (set the path to the subtree).
3.6. Scope DOM objects of model

We consider question as what elements dynamic model of "scope" (i.e., available for processing) DOM objects generated DOM-elements. The answer depends, first, the number of the interpretation of the passage, and second, the relative position in the DOM model element and this element.

The influence of passage interpretation. In this paper, we are oriented on the general case multi pass dynamic interpretation of the model, when an interpreter is interpreting the cycle several times through a dynamic model handles recursive traversal of the tree of the current model. If at first (mostly) pass the interpreter performs the current state and creates a DOM objects associated with the current state, then the next (additional) passes - only the items of fixed current conditions. Consequently, the relative positions of the elements affects the scope of only the main pass, and the additional DOM passes all the objects created in the first pass, are available ("scope") from any element of the current state. The influence of the mutual arrangement of the elements. At the first pass, the interpreter creates a DOM-objects handling of the DOM elements. These DOM scope objects of the model elements that are processed by the interpreter after creation, and therefore not available of those that are processed to create them. Consequently, the scope is determined by the order of processing elements during the model interpretation. In the dynamic model of the set of elements associated with a certain condition, ordered, in interpreting the current state of the tree model elements are processed in the order they appear. We consider an element of a dynamic model, for which there are other elements of the brothers, associated with the same condition preceding it ("elder brothers") and following it ("junior brothers"). Accordingly, the elder brothers are not available, and the younger - scope objects created by these elements.

In Fig. 3.8 shows part of a dynamic model that contains a hierarchy of three states, sta:S0, sta:S1 and sta:S2, which are associated three DOM element, dom:D0, dom:D1 and dom:D2, and six shares, act: A0, ..., act: A6. The action act: A0 is processed on the second pass, the interpretation of the cycle, the other - on the ground. Of the shares act: A0, processed on the second pass, to see all DOM objects created in the first pass: dom:D0, as well as the dom:D1, if the current state of the sta:S1, or dom:D2, if the current state of the sta:S2.

**Figure 3.7.** Variants of data receivers to store content from DOM-objects
Of the remaining shares that are processed on a first pass, only the scope DOM objects that are created prior to processing. Thus, the object dom:D0 can be seen from the shares act:A2, ..., act:A6; dom:D1 - shares of the act: A4; dom:D2 - of the action act:A6.

3.7. Processing content of DOM-objects

Thus using DOM elements associated with the states of the dynamic model, while the main (first) iteration of the loop created by the interpretation of DOM-objects containing XML-data. These objects, ultimately intended to produce output data sent to the external (relative to the situationally-oriented database) (Strauch & Kriha, n.d.) environment. Such processing of the resulting DOM-objects, usually performed during additional passes of interpretation. In Fig. 3.9 illustrates the use of the DOM object on the second pass, the interpretation of the fragment to generate HTML-Code. It is assumed that the interpretation of the model runs on a web server, and the result is sent to the client browser via HTTP. This element of the receiver named rcv:Echo means the standard output the resulting data and its attribute, pass = "2" - the fact that the element is processed by the interpreter on the second pass. The resulting HTML code generated by the XSL transformation of the contents of the XML DOM object «D3». And although the details are hidden in the text of the transformation stylesheet doc:T3, the overall picture is quite clear from the model.

4 Profound Example

4.1. User interface

To illustrate the use of dynamic DOM-objects we consider Internet-based application that is intended for display to the user information about the student as well as students and passing of items.

Data about students, subjects and submission are stored as XML-files stud.xml, predm.xml and sdacha.xml, placed in a folder associated XML data situationally-oriented database. Fig. 4.2 conceptual model of these documents and test instances of their contents.
Document stud.xml (Fig. 4.2, a) on the root element "Students" can contain several nested "student" with the attributes of the "code" (code for the student) and "name" (last name, first name of the student), and the attribute "code "is an identifier. A copy of this document (see Fig. 4.2 g) contains information about the three students: Ivanov, Petrov and Sidorov. Similarly, the XML-document predm.xml (Fig. 4.2b) on the root element "object" can contain several nested "object" with the attributes of the "code" (code of the object) and "titles" (the name of the object), and the attribute "code "is an identifier. Model instance of this document (see Fig. 4.2 d) contains information about three things: "systems theory", "mathlogic" and "cryptography". 

In this example it is assumed that the same raw data (Jea, Chang & Chen, 2009; O’Connor, Merriman & Nelson, 2010) by the user is required to submit to the web browser in two ways: with a focus on students and their achievement in the subjects or subjects with a focus on the passing of these items and students. Fig. 4.3 shows the 4-screen forms that must be generated online application to perform these functions. Form, and displays the situation "StudentsList." Here is a button "to the subject" provides a link to the form used, displaying the situation "SubjectsList." Radio buttons placed on the left of the names of students who are used to select the situation (one of the students). Button "of student performance" causes a transition to a form to display in a situation of "StudentSelected" performance of the selected student. Form used in the same way shows the situation "SubjectsSelected" list items. Here is a button "to students" provides a transition to the form and displaying situation "StudentsList." Radio buttons placed on the left of the names of objects are used to select one of the items. Button 'achievement in the subject "causes a transition to a form of g to show progress on the selected subject. Form in displays information (Dejanovic, Milosavljevic, Perisic, & Tumbas, 2010; Chen & Huck, 2001; Benzaken, 2013; Kaur, & Rani, 2013; Han, 2014) about the selected student's academic achievement in the form of unnumbered list by the student subjects with an indication of the estimates. The button "To the list of students" is used to return to the form and, as a button "to the subject" - to go to the form used. Form g in a similar way displays information about the progress of the selected object in the form of an unordered list of students who have passed the subject, showing the estimates obtained. The button "To the list of things" used to return to the form used and the button "to students" - to go to the form of a. Requires that in appropriate situations online application is sent to the client browser form, as well as host and handled accordingly the impact on the user controls.
Figure 4.2 XML-source data such as: a-c – conceptual model; d-e – model test items

Figure 4.3. Examples of screen forms for reporting on students, subjects and submission of a - a list of students with a choice of the individual student; b - a list of items to select a single item; c - the list of selected student donations; d - a list of donations to the chosen subject

4.2. Dynamic model

To implement the functions of Internet-based applications using a dynamic model (Panach, Juristo & Pastor, 2013) presented in Fig. 4.4. The model has a three-level hierarchy of states: the state of the root level 1 sta:Students Objects; two states of the 2nd 
level, placed in sub-model sub: Students Objects; four states of the 3rd level (Pokorný, 2013), placed in two sub-models: sub: Students and sub: Items.

**Root condition.** State sta:Students Objects is the root state of the model. It declared the associated data: three documents XML, four styles of transformation XSL, library of macros, and then the immersion in the sub-model of the next level of the hierarchy.

**Submodel second level.** Submodel sub: Students Objects defines two states, sta: Students and sta: Items that determine the current mode of presentation. In a state of sta: Students focus on the data of students, and be able to sta: Items - on items. State transitions are provided jmp-elements that become active when the user clicks the buttons "to the subject" and "To students," respectively (actual pressing interpreter defines an array of POST, obtained from the client, along with URL). Depending on the current state of this sub-model (Batory, 2006; Takahashi, 2013) div-elements provide immersion in the sub-model sub: Students or sub: Items.

![Diagram of the dynamic model of internet-applications](Image)

*Figure 4.4. Dynamic model of internet-applications*
Submodels of the third level. These submodels define the state of the lower level of the hierarchy: sta:StudentsList and sta:StudentSelected in submodel sub:Students and sta:SubjectsList and sta:SubjectsSelected in submodel sub:Items. Change the current state (navigation) within the sub-models provided jmp-elements that are activated when the user presses the buttons "Student achievements", "Return to students" and "achievement in the subject", "Back to subjects".

4.3. Data sources

DOM-elements. In cell-states of lower-level sub-models contain internal dom-elements, which retrieves information needed in appropriate situations: in a state of sta:StudentsList - all students; able to sta:SubjectsList - all subjects are able to sta:StudentSelected - on items handed selected student and obtained in this assessment; able to sta:SubjectsSelected - about the students who have passed the selected item, and received at the same estimates. DOM-elements dom:StudentsList and dom:SubjectsList loaded from simple data sources from ready for made XML-documents stored in memory and associated data declared in the root condition. In contrast, DOM-elements dom:StudAcadachiev and dom:SubjAcadachiev use complicated-WIDE data sources, providing a composition of several XML-documents (Sudarsan & Gray, 2006; Rose, Malaika & Schloss, 2006; Su-Cheng & Lee, 2009; Tang, Yu, Wong & Li, 2008) from the associated data memory, depending on the parameters - the user has selected a student or subject.

Att-elements. To store the parameters defining the student or the item selected by the user to be able to sta:StudentSelected and sta:SubjectsSelected provided att-elements att:codeSt ("Student Code") and att:codeSb ("code object"), creating an element's attributes-state in memory of the current state. Creating Attributes and download their values occurs at the stage of interpretation prolog's states (mode = "prolog"). On the interpretation of the cycle at which the transition from the sta:StudentsList a state sta:StudentSelected or state sta:SubjectsList in state sta:SubjectsSelected. Source element using the values of the selected student IDs or object that is retrieved from the array POST, obtained from the user with the URL (method = "post").

Complex data sources DOM-objects. In order not to clutter up the diagram model information not directly related to navigation, complex data sources are presented in the form of macros (as indicated by the attribute mcrLib), which is rendered in a macro-macro-libraries (see Fig. 4.5). During the interpretation of macro expansion are inserted from the library to a dynamic model. Consider the data source for dom:StudAcadachiev (data source for dom:SubjAcadachiev has a similar structure). It has three levels of nested elements sources: at the top level - the element src:StudentPassing that uses nested src:ScoreStudents, which, in turn, used the element src:StudentsList. Let's start with the innermost element src:StudentsList.

Source element src:StudentsList retrieves data from the XML-document xml: StudentsList as a subtree (method = "cut"), beginning in the XML-element "student". In this case, we seek the element "student" who has the attribute "code" (field = "@code") is the value stored in the element att:codeSt memory of the current state of the dynamic model (value = "att:codeSt").
In other words, the source retrieves XML-data (Herskovits & Chen, 2008; Kudrass & Conrad, 2002; Nassis, Dillon, Rajagopalapillai & Rahayu, 2006; Sladic, Milosavljevic, Konjovic & Vidakovic, 2011) corresponding to the selected student. Next, using other sources of data are supplemented by information about the passing and the student subjects. Source element src:ScoreStudents merges two XML-trees (method = "merge"). The parent tree is formed already considered internal power-cussed src:StudentsList (parent = "internal"). Subsidiary tree is a document xml: Marks (childDoc = "Marks") defined in the root condition. Condition of the merger - the equality of the student code of the element "student" of the parent tree (parentElement = "student" parentField = "@code") and the student code of an element "renting" a child of the tree (childElement = "Passing / Passing" childField = "@codeSt"). As a result, information about the selected student is supplemented by information regarding the completion of this student. Each passing must be supplemented by information on the passed subject (called the subject), which is achieved in the top-level source.

Source element src:StudentPassing merges two XML-trees (method = "merge"). The parent tree is formed already considered internal power-cussed src:ScoreStudents (parent = "internal"). Subsidiary tree is a document xml: SubjectsList (childDoc = "SubjectsList") defined in the root condition. Condition of the merger - the equality of the object code for the element "passing" of the parent tree (parentElement = "subject / Passing" parentField = "@codeSb") and the object code for the element 'subject' child tree (childElement = "items / item" childField = "@code ").

As a result, information on each deposit of the selected student is supplemented by information about exams at the subject. Downloadable XML-data Fig. 4.6 a, b shows the conceptual model of XML-data (He & Zhai, 2013; Case, 2006; Dekeyser, Hidders, & Paredaens, 2004) generated by complex sources to be loaded into DOM-objects dom:StudAcadachiev and dom:SubjAcadachiev. Fig. 4.6, c and d are given copies of the XML-data for cases where the user selects the first student (Ivanov, code «s01») or the first item ("Systems Theory", the code "P01").
4.4. Formation results for output to the client browser

The resulting HTML code is sent to the client in response to the input request, should ensure the formation of a web browser window images corresponding to the current state of the dynamic model (see. Fig. 4.3). HTML-code is formed on the second pass, the interpretation of the dynamic model and consists of fragments generated as a result of the interpretation of the individual elements of current conditions at different levels of the hierarchy (Gusarenko & Mironov, 2013; Mironov & Gusarenko, 2013). So, on the second level of the hierarchy in the sub-model sub: Students Subjects elements-action act: Students and act: Objects provide output header "Students" and "Objects". In the same sub-model elements btn:Objects and btn:Students form HTML-button "to the subject" and "the students" at the top of the image. Further, on the third level of the hierarchy in the sub-model sub: Students and sub: Items using elements receivers rcv:Echo by XSL-transformation of the content DOM-objects in the middle of the image formed lists: students selected student donations, items, donations to selected subject. In the same sub-model elements btn:StudAcadachiev, btn:Students, btn:SubjAcadachiev, btn:Objects formed HTML-button "student's academic achievement", "To students 's achievement in the subject", "To subject" at the bottom of the image.

XSL-transformation. Of particular importance to us is the formation of fragments of HTML-code by XSL-transformation of XML DOM-content of dynamic objects. This transformation in the dynamic model (see. Fig. 4.3) provides data receivers rcv:Echo DOM-object dom:StudentsList, dom:SubjectsList, dom:StudAcadachiev and dom:SubjAcadachiev. Transformation is performed on the second pass, the interpretation of the dynamic model in accordance with style sheets specified in the attributes of «xsl» rsv-relevant elements. Fig. 4.6 shows the conceptual model of such a transformation, the following notation (Gusarenko, 2013). In each model on the right shows a conceptual model of XML-data undergoing transformation, and to the left - model XSL-template (Template), used to produce the resulting HTML-output data. Applied the so-called boosts conversion, during which the processor transformation crawls source XML-tree and applies the soot-corresponding patterns associated with its nodes.

List of students (see Fig. 4.6 a) is formed on the basis of the document xml:StudentsList (see. Fig. 4.1, a, r) is loaded in the dom:StudentsList. Provided the only pattern associated with the XML-element "student"; This template will be executed for each student (Ivanov, Petrov, Sidorov, as shown in Fig. 4.1 g) represented in transitional tree. Each time you run the template will generate HTML-element <INPUT> with the attribute type = "radio", which is in the client's browser displayed as radio buttons. Attribute name = "codeSt" specifies the name of the button on which you can check its status in the array POST, and the attribute value, which is set to code the student to determine with the selected student. Conventional design creates for the very first element <INPUT> attribute checked = "checked", specifies the range of the first student default. Following the element <INPUT> displays the name of the student and the HTML-element <BR>, while providing transitioning to start a new line.

List of items (see Fig. 4.6 b) is formed in a similar manner on the basis of the document xml:SubjectsList (see. Fig. 4.1b, d), loaded in the dom:SubjectsList.
Figure 4.5. Content Model DOM-objects "StudAcadachiev" and "SubjAcadachiev" formed complex data sources: a, b - a conceptual model; c, d - Model instances for the case of selecting the first student and the first subject

Figure 4.6. Conceptual model transformation XML DOM-content objects in the formation of HTML-code that is sent to the client browser: a - creating a list of students; b - creating a list of objects; c - creating a list of selected student donations; d - creating a list of donations to the chosen subject
List of the selected student donations (see Fig. 4.6 in) is formed on the basis of XML-data (see. Fig. 4.5, and c) have been added to dom:StudAcadachiev. Transformation is performed using two templates, one of which is associated with XML-element "student", and the second - with XML-element "Submit".

The first pattern is triggered only once, at the root processing element; it generates a header row containing, in bold name of the selected student and starts bypassing the internal elements of the root element. The second pattern is triggered when processing each passing selected student; it displays HTML-element <LI>, which forms in the browser string bulleted list. Each row displays the name of the passed subject matter and the resulting score, separated by a dash. List of donations for the selected object (see Fig. 4.6 g) is formed in a similar manner on the basis of XML-data (see. Fig. 4.5, b, d) have been added to dom:SubjAcadachiev.

**Conclusion**

In this paper, at a conceptual level (excluding the features and capabilities of the implementation of the medium) is proposed and investigated a possible approach for manipulating XML data in a situationally-oriented databases, based on dynamically created DOM objects.

The concept of dynamic DOM objects based on the DOM binding sites in the states of the dynamic model: DOM objects are created, loaded, used to handle XML content as the corresponding states of the dynamic model are ongoing and are deleted after the change of the current state. In one approach, this is achieved by programming functions in the cell-actions associated with states of the dynamic model.

The difference lies in the concept that 1) the elements of a dynamic model of states as a child provides DOM elements, which, in turn, child elements, define the sources of downloadable XML data and child elements Receivers saves the contents of the XML 2) during the interpretation of the dynamic model of the interpreter runs automatically create DOM objects for the current states of the model and loading XML data with a possible transformation, and automatic removal of DOM objects by changing the current state.

Implementation the concept of a dynamic model allows the developer to specify a flexible declarative XML data required in certain situations and how to get from various sources, sparing him the need for this programming functionality of the corresponding. Further research in this area are related to software implementation of proposed conceptual solutions in the form of the additions to the structure of a dynamic model of HSM and interpreter in the algorithms of dynamic models HSMI.

**Acknowledgments**

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


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.


