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

A. S. Gusarenko
Department of Computer Science and Robotics
Ufa State Aviation Technical University
Ufa, Russia
e-mail: valter-hartman@mail.ru

V. V. Mironov
Department of Computer Science and Robotics
Ufa State Aviation Technical University
Ufa, Russia
e-mail: mironov@list.ru

Abstract[1]

Management of the XML data associated with states of dynamic model in situational-oriented databases is considered. The concept of dynamic DOM objects which are automatically created, loaded, destroyed in the course of change of current states of dynamic model is offered. Means of dynamic DOM objects specification by means of the DOM elements associated with states of dynamic model are offered. Specifications of loaded and saved XML contents by means of data sources and data receivers defined in dynamic model are discussed.

1. Introduction

Now in the field of information technologies the approach to development of information systems on the basis of direct use of models of high level of abstraction (Model-based, model-driven engineering) [1–3] actively develops. Situational-oriented databases, in which XML data associated with states of the built-in dynamic model are processed in a context of model current states, can be considered in the tideway of this approach [4, 5]. Situational-oriented databases can form a basis of the internet applications providing dynamic formation of content according to current situation [6–8]. In this connection the organization of processing of the XML data associated with current states of dynamic model, discussed in this article are important.

In situational-oriented database includes the following components (Fig. 1):

- HSML (HSM Library) is the library of dynamic models containing a set of the Hierarchical Situational Models in the form of transition graphs hierarchy with finite states number (Finite State Model);
- CSM (Current States Memory) is the memory of current states of dynamic models;
- ADM (Associated Data Memory) is the memory of the XML data associated with states of dynamic model;
- AFL (Associated Functions Memory) is the library of the data processing functions associated with dynamic model states;
- HSMI (HSM Interpreter) is the interpreter of dynamic model which in response to external query Q forms answer R by processing of some dynamic model HSM from HSML on the basis of tracking of HSM current states kept in CSM, processing the associated data from ADM and executing the associated functions from AFL.

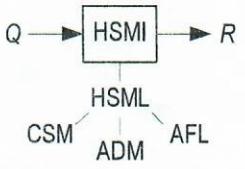

Fig. 1. Architecture of situational-oriented database

When using situational-oriented database on a Web server as a basis for the internet application, the query Q is the set of parameters received together with URL (for example, as form parameters in the POST mode), and the result R is the answer HTML code sent to a client. Query parameters specify a processed dynamic model and the necessity of change of its current states.

On Fig. 2 it is given an example the dynamic HSM model. The root state *sta:S0* symbolizing model as a whole, contains three child elements: two definitions of XML documents from ADM and one sub-model from HSML. The first document *doc:X1* is the *X1.xml* file, and the second one *doc:X2* is the *X2.xml* file, both from the XML folder in ADM. The sub-model *sub:M* defines two states *sta:S1* and *sta:S2* which are child states for a parent state *sta:S0*. In turn child members of states *sta:S1* and *sta:S2* contain (1) transition elements *jmp:S2* and *jmp:S1* providing change of a current states, and (2) action ele-

[1] **Proceedings of the 14th international workshop on computer science and information technologies CSIT'2012, Ufa – Hamburg – Norwegian Fjords, 2012**

ments *act:A1* and *act:A2* providing executions of certain actions in the appropriate states.

In the course of interpretation of action elements the AFM functions associated with them are executed to provide processing of XML documents from ADM. In particular, DOM objects creation, XML documents loading, contents XSL transformation, transformation result sending as query result can be provided. Processing of the associated data, thus, is programmed in the associated functions from AFM being outside of HSM that complicates understanding of logic of data processing. Specifications of processing of XML documents can be distributed on various elements of HSM. For example, in some state XML document loading is specified; in one of the sub-states of this state XML document modification is specified; in other sub-state saving of changes is specified; in the third sub-state formation of output result for sending to the client is specified. It is necessary to analyze jointly some functions associated with several states to understand logic of this processing. Therefore would be the useful if the logic of processing of XML documents from ADM could be specified explicitly in HSM.

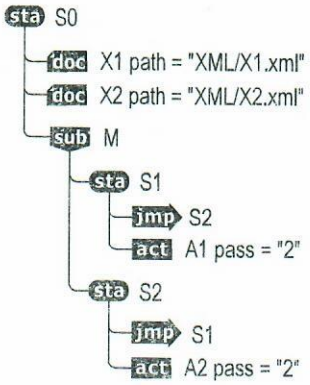

Fig. 2. Diagram of dynamic model including definitions associated with XML documents

DOM objects. DOM (Document Object Model) [9] is the well-known platform independent object-oriented interface for access to the documents XML, XHTML, HTML. The technology of processing of XML data on the basis of DOM provides DOM object creation, external XML document loading, document manipulation, document saving. Considering great opportunities for DOM technology, its flexibility and wide use, we will try to solve the considered problem on the basis of DOM.

Thus, we would like to give HSM developers opportunity to specify the DOM objects associated with current states of dynamic model. We would like, that during HSM interpretation according to these specifications DOM objects were automatically created. The appropriate XML data shall load automatically in DOM objects from ADM. DOM objects shall become available to processing in other parts of model corresponding to current states. The necessary XML contents of DOM objects shall remain timely, and unnecessary contents shall be deleted. Thus we would like to give opportunity to specify in HSM conversion of the XML data containing in DOM objects. For example, the developer could specify for some state of model that XML data of the associated DOM object shall be transformed according to a certain XSLT style sheet. Objects with similar behaviour we will call *dynamic DOM objects*.

2. The dynamic DOM objects concept

DOM elements. So, during interpretation of a dynamic model the DOM objects associated with states of a dynamic model and available to processing from other parts of model shall be created automatically. For this purpose we will provide in dynamic model special DOM elements within which sources of XML data can be specified. Respectively in the course of interpretation of dynamic model we will provide the appropriate processing of the specified DOM elements: DOM objects creation and loading when parent states become current; DOM objects remove when parent states cease to be current. It will allow to process the XML data associated with current statuses, addressing to DOM objects from the functions given in actions of a current state. We expect that labor input of programming as the routine of preparation and loading of DOM objects is laid to the interpreter thus shall decrease.

For implementation of this idea it is necessary conceptual studies of a number of questions: how to create DOM objects; how to load XML data; in what states DOM objects are available to processing; how to delete DOM objects and how to save in ADM their XML contents.

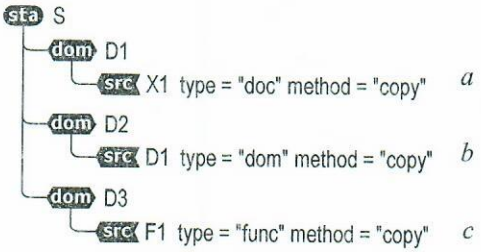

Fig.3. Assignment of XML data sources

XML data sources. Some options of XML data loading in the DOM objects created in case of interpretation of DOM elements can be offered: from an external XML file; from other DOM object of model; as result of some function, etc. On Fig. 3 it is shown, how various options of data sources are represented on the model diagram. The symbol of dom represents a DOM element, a name and attributes of an element are specified to the right of the symbol (the DOM elements *dom:D1*, *dom:D2*, *dom:D3* are specified on Fig. 3). The symbol of data source src is attached to a DOM element as a child

member and specifies loading of XML data in the DOM object generated by a DOM element. In turn, attributes of a data source element specify, XML data from where undertake.

Loading from external XML file. Fig. 3, *a* shows a fragment of the model diagram on which for a DOM element the data source in the form of the XML document from ADM is specified. The attribute *type = "doc"* specifies that it is necessary to load the XML document from ADM, and the data source name *X1* refers to *doc:X1* definition. The attribute *method = "copy"* says that contents of the document *doc:X1* shall be copied entirely in the created DOM object. Thus, when processing the element *dom:D1* the DOM object *D1* will be created, and when processing element *src:X1* the file *X1.xml* will be loaded into the DOM object *D1*.

Loading from another DOM object. Fig. 3, *b* shows a fragment of the model diagram on which for a DOM element the data source in the form of the reference to other DOM element is specified. To it points the attribute *type = "dom"*, i. e. the data source name *D1* refers to *dom:D1* definition. It is supposed that by the time of element *src:D1* processing the DOM element *dom:D1* is already processed, i. e. the DOM object *D1* is created and loaded. The interpreter addresses to DOM object *D1* and copies its XML contents in new DOM object *D2*.

Loading from a function. Fig. 3, *c* shows a fragment of the model diagram on which for a DOM element the data source in the form of the function returning as result a line of XML data is specified. To it points the attribute *type = "func"*, i. e. the data source name *D1* refers to function from AFL. In the course of processing the interpreter calls the specified function and loads returned result into the created DOM object.

Loading with filtering. In actual practice often it is required not only to copy XML data from data source, but also to execute data transformation. For example, often it is required to filter data, i. e. to load into DOM object a certain part of the data, satisfying to some conditions. On Fig. 4 the filtration for the data source in the form of the XML file from ADM is illustrated. In the dynamic model on Fig. 4, *a* in a root state *sta:S* are defined both XML document *doc:X1*, and the element *dom:D3* in which *doc:X1* loading with filtering is provided. The data source *src:X1* has the attribute *method = "cut"* specifying that a certain sub-tree will be derived from the initial XML document. The additional element, field and value attributes are intended for the specification of a condition of filtering. The *element* attribute contains XPath-expression which defines a set of nodes in a XML tree one of which will be a root of loaded sub-tree. The *field* attribute contains XPath-expression which defines in sub-tree a checked XML element or attribute. The *value* attribute contains demanded value. In this example it is required to load into DOM object sub-tree, beginning in the element *e1* which *k1* attribute matters "*123*". On Fig. 4, *b* the model of the XML data *doc:X1* in the graphic notation [11] is provided. The root XML element *E0* can contain some child XML elements *e1*, each of which contains the *k1* key attribute (identifier) and the *a1* (non-key) attribute. Thus, XPath-expression *'/E0/e1'*, specified in the *element* attribute of a source *src:X1*, addresses a set of all XML elements *e1*; the *field* attribute addresses in *e1* the *k1* XML attribute; the *value* attribute sets for it required '*123*' value. During *dom:D3* processing the interpreter addresses to *src:X1* source, touches the XML elements *e1*, finding single at what the *k1* XML attribute has required '*123*' value, and loads the appropriate sub-tree in DOM object. As a result in DOM object *D3* the XML data appropriate to model provided on Fig. 4, *c* will be loaded.

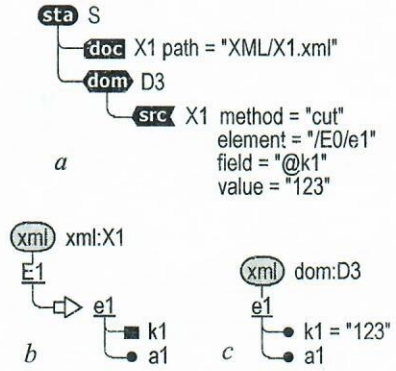

Fig. 4. DOM object loading with XML data filtering

Loading with transformation. Generally conversion of XML data when loading can be considered as XSL transformation of the data source XML document, and for a solution the XSLT technology can be used (Fig. 5). In this case the data source element contains the following attributes: the *method = "xslt"* attribute specifying need of XSL transformation; the *stylesheet* attribute referring to the used style sheet; the *params* attribute containing the list of global parameters for transmission to the style sheet.

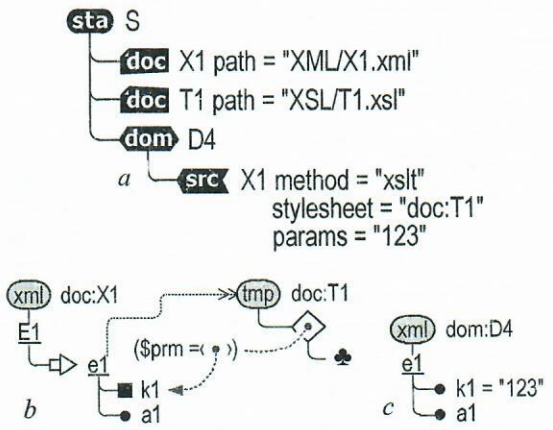

Fig. 5. DOM object loading with data transformation

Processing a data source element, the interpreter reveals need of XSL transformation, loads the style sheet, transfers global parameters, executes transformation and loads result of transformation in DOM object.

In an example in the Fig. 5 it is shown, how by means of XSLT the loading task with filtering (as in the previous example, see Fig. 4) is solved. The conceptual model of XSL transformation of XML data (Fig. 5, *b*) is provided in the graphic notation [11]. Templates of transformation specify element *e1* search at which the *k1* attribute is equal to value of the global *prm* parameter, and output of the found element together with its sub-tree as result of transformation.

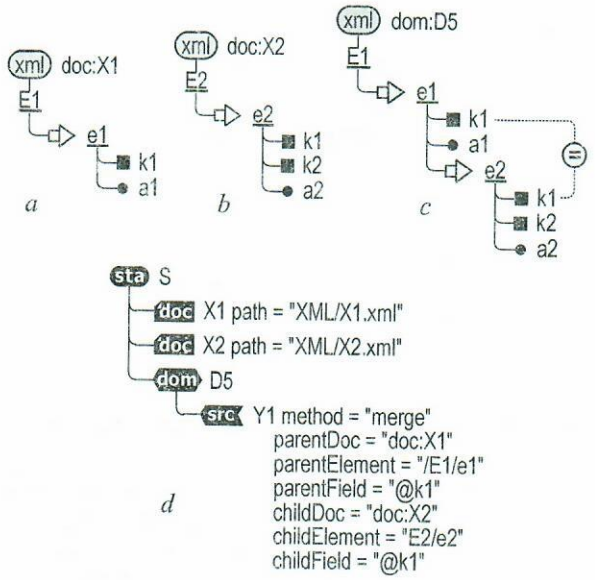

Fig. 6. DOM object loading with data merging

Loading with XML data merging. In more difficult case it can be demanded to load in DOM object the XML data received from several XML documents. For example, for entity set the general information, containing in one document, it is necessary to add the detailed information containing in other document. In the Fig. 6 such data source providing merge of XML documents is illustrated. The model of the first of merged documents which contains general information and represents itself as the parent document, is provided on the Fig. 6, *a*. The model of the second of the merged documents, containing detailed information and representing itself as child, is given on Fig. 6, *b*. The model of the resultant document loaded into DOM object, is given on Fig. 6, *c*. The appropriate fragment of a dynamic model specifying merge of two documents in loading process of DOM object *D5*, is given in the Fig. 6, *d*. The data source element has the attribute *method = "merge"* specifying that XML data loaded in DOM object shall be created by a way of merge of two XML documents. These documents are set by the *parentDoc* attribut (the parental document) and *childDoc* attribute (the child document). Other attributes set XPath expressions specifying features of merge of documents:

- *ParentElement* specifies a set of XML elements to which the sub-trees taken from the child document will be attached in the parent document;
- *ParentField* specifies memberwise from the previous set (*parentElement*) the value used for identification of the attached sub-tree;
- *ChildElement* specifies in a child document, the set of elements which, together with their sub-trees are used to copy to the parent document;
- *ChildField* specifies for each element from the previous set (*childElement*) the value used for identification attached sub-tree.

During source element processing the interpreter addresses to the parent document and for each its XML element of the *parentElement* type finds in the child document all corresponding to it a *childElement* type XML element such that *parentField* and *childField* values match. The found elements together with the sub-trees are copied in the parent document as children of the processed XML element *parentElement*.

Saving of XML data of DOM objects. XML contents of DOM objects can be changed in the course of interpretation (for example, according to request of the user), and then can be demanded to save changes in ADM. Therefore, it is necessary to provide means of saving of contents of DOM objects.

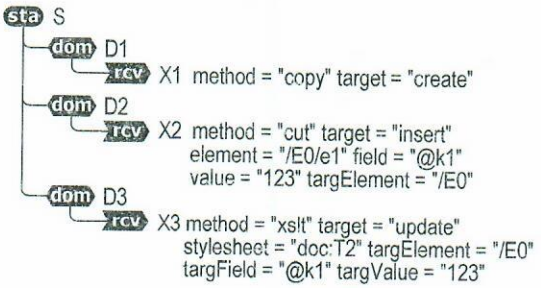

Fig. 7. Data receivers for DOM object content saving

Data receiver. As saving of DOM object contents is reverse in relation to DOM object loading, the appropriate element of model called *data receiver* was provided. For receiver representation on the chart of model the character rcv is used. (It same as the source character, but has an opposite direction and *rcv* label). The element is illustrated on Fig. 7 where it provides saving of contents of DOM objects *dom:D1*, *dom:D2* and *dom:D3* in target XML documents *doc:X1*, *doc:X2*, *doc:X3* respectively.

As well as in data source, the *method* attribute specifies. which data of DOM object are required to be saved: the attribute *method = "copy"* means that is necessary to save

XML contents entirely; the attribute *method = "cut"* orders to save a certain XML sub-tree from DOM object; the attribute *method = "xslt"* commands to save result of XSL conversion of contents of DOM object.

The *target* attribute specifies rules of placement of saved data in the target document. For example, the attribute *target = "create"* means that is necessary to rewrite contents of the target document (if such document already exists, it shall be replaced with the created one). The attribute *target = "insert"* orders to insert saved XML data into a tree of the target document (the path to a parent member of the target document shall be specified). The attribute *target = "update"* commands to replace with saved XML data some sub-tree of the target document (the path to this sub-tree shall be set).

Visibility of DOM objects in model. Let's discuss visibility of DOM objects: from what parts of a dynamic model the DOM objects generated by these or those DOM elements are available to processing. The response depends, first, on number of pass of interpretation, secondly, from a relative positioning in model of a DOM element and a point of processing of the appropriate DOM object.

Dependence on interpretation pass. Here we assume possibility of multi-pass interpretation of a dynamic model when the interpreter several times processes a dynamic model by the recursive bypass of the current tree of model within an interpretation cycle. On the first (main) pass of interpretation changes of current states are fixed also the DOM objects associated with current states are created. On the subsequent (additional) passes elements of the fixed current statuses are only processed. Therefore, the relative positioning of elements in model influences visibility only on the main pass, and on additional passes all DOM objects created on the first pass, are available ("are visible") from any element of a current state.

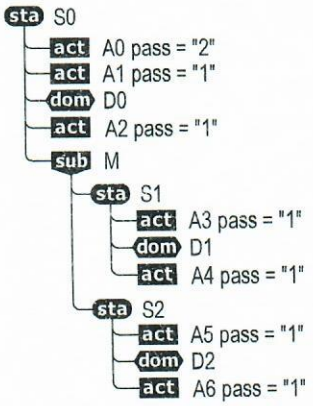

Fig. 8. Visibility of DOM objects in dynamic model

Influence of a relative positioning of elements. On the main pass the interpreter creates DOM objects when processing the corresponding DOM elements. The DOM object is visible from those elements of model which are processed after processing of the appropriate DOM element, and are respectively invisible from those elements which are processed before DOM element processing. Therefore, visibility is defined by order of processing of elements of model during interpretation. In a dynamic model the set of the elements associated with some state, is arranged; in the course of interpretation of a tree of current statuses these elements are processed as their following. Let's consider some element of a dynamic model which has siblings associated with the same status. It can be preceding siblings or the following siblings. Respectively, the considered element isn't visible to preceding siblings and is visible to following siblings.

Fig. 8 shows the fragment of a dynamic model containing hierarchy from three states (*sta:S0*, *sta:S1*, and *sta:S2*) with which three DOM elements (*dom:D0*, *dom:D1*, and *dom:D2*) and six actions (*act:A0*, ..., *act:A6*) are associated. The action *act:A0* is processed on the second pass of interpretation, and remaining actions are processed on the first one. In *act:A0* all DOM objects created on the first pass are visible: *dom:D0* irrespective of a current state; *dom:D1* if a current state is *sta:S1*; *dom:D2* if a current state is *sta:S2*. Remaining actions are processed on the first pass therefore from them only those DOM objects which are created earlier are visible. So, the object *dom:D0* is visible from the *act:A2*, ..., *act:A6*; the object *dom:D1* is visible from the *act:A4*; the object *dom:D2* is visible from the *act:A6*.

DOM objects contents processing. So, DOM objects are created during the main pass of interpretation by means of DOM elements. These objects are finally intended for formation of output data which go to external environment in response to input request. Such resultant processing of DOM objects is, as a rule, carried out during additional passes of interpretation.

Fig. 9 illustrates DOM object use on the second pass of interpretation for generation of a fragment of the HTML code. It is supposed that interpretation of model is carried out on the Web server, and the result goes to the client browser according to the HTTP protocol. The name rcv:Echo means that the receiver is the standard stream of resultant data. The attribute "pass = "2" means that the element is processed by the interpreter on the second pass. The resultant HTML code is formed by XSL transformation of XML contents of DOM object *D3*. And though details of transformation are latent in the style sheet *doc:T3*, the overall picture is quite clear from a dynamic model.

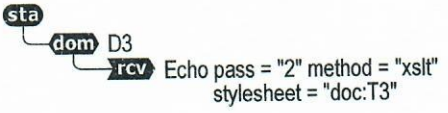

Fig. 9. Using DOM object for HTML code generation

3. 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 is based on a binding of DOM objects to states of a dynamic model. DOM objects are created, loaded, processed, when the appropriate states of a dynamic model are current. In a known approach it is reached by labor-consuming programming of the functions called in action elements, associated with states of a dynamic model.

The offered concept has the following features:

1) The DOM elements specifying DOM objects can be bound to states of a dynamic model; the data source elements specifying loaded XML data, as well as the data receiver elements specifying saved XML contents, can be bound to DOM elements.

2) Automatic creation of DOM objects, as well as loading of XML data with required conversion is executed during interpretation for current states of a dynamic model.

As expected, implementation of the concept will allow developers of dynamic models to specify the XML data required in these or those situations, as well as methods of obtaining XML data from different sources in the declarative form.

Further researches in this direction are connected to program implementation of the offered conceptual decisions in the form of the appropriate additions in structure of a dynamic model HSM and in algorithms of the interpreter of dynamic models HSMI.

Acknowledgments

The research is supported by the Russian Fond of Foundation Research grant № 10-07-00167-a.

References

1. Model Based Systems Engineering [Electronic Resource]. URL: http://mbse.gfse.de.
2. Model Based System Development [Electronic Resource]. URL: http://www.ru.nl/mbsd.
3. Model-Driven Engineering [Electronic Resource]. URL: http://en.wikipedia.org/wiki/Model-driven_engineering.
4. Mironov V. V., Юсупова Н. И., Shakirova G. R. Situational-oriented databases: The concept, architecture, XML-implementation // Vestnik UGATU: scientific journal of Ufa state aviation technical university. 2011.V. 14, No 2 (37). P. 233–244.
5. Mironov V. V., Yusupova N. I., Shakirova G. R. Situational-oriented databases: external representation based on XSL // Ibid. 2011 V. 14, No 4 (39). P. 200–209.
6. Mironov V. V., Malikova K. E. Internet-applications based on built-in dynamic models: The idea, concept, security // Ibid. 2009. V. 13, No 2 (35). P. 167–179.
7. Mironov V. V., Malikova K. E. Internet-applications based on built-in dynamic models: The architecture, data structure, interpretation // Ibid. 2010. V. 14, No 1 (36). P. 154–163.
8. Mironov V. V., Malikova K. E. Internet-applications based on built-in dynamic models: elements management of user interface // Ibid. 2011 V. 14, No 5 (40). P. 170–175.
9. Document Object Model (DOM) [Electronic resource]. URL: http://www.w3.org/DOM.
10. Simple API for XML [Electronic resource]. URL: http://en.wikipedia.org/wiki/Simple_API_ for_XML.
11. Mironov V. V., Yusupova N. I., Shakirova G. R. Hierarchical data models: concept and implementation based on XML. Moscow: Mashynostroenie, 2011. 453 p.