Advances in Intelligent Systems Research, volume 174

Proceedings of the 8th Scientific Conference on Information Technologies for Intelligent Decision Making Support (ITIDS 2020)

Graphic Documents Parametric Personalization for Information Support of Educational Design Using Situation-Oriented Databases

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

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

Gayaz Tuguzbaev
Computer Science & Robotics Dept
Ufa State Aviation Technical University
Ufa, Russia
tuguzbaev.g@ugatu.su

***Abstract*—This article discusses the personalization of graphic documents intended as blanks for educational design. The blanks of design documents are filled in with design personal data in order to reduce the complexity of the subsequent design. The so-called parametric personalization is considered, in which the parameters of the figures that are already placed in the blank of the project document are adjusted. A document template is created manually using a graphical editor. Pre-marking of personalization labels is performed when creating a template. Subsequently, computer processing of the template is performed, in which the personalization labels in the template are replaced with personal data from the database. Templates are processed using situation-oriented databases that allow integrating heterogeneous data under the control of a highly abstract hierarchical situational model. The concept of virtual documents that are mapped onto heterogeneous external data is used in the situational model to specify access to data. Features of the graphic formats VDX, VSDX, ODG and FODG are considered in the aspect of mapping virtual documents. It requires both mapping entirely to an XML file for VDX and FODG formats, as well as to some XML files in a ZIP archive for VSDX and ODG formats. The construction of situational models to ensure the personalization of graphic templates is considered. The XML file of the document or part of the document is loaded into the DOM object. A search in the XML tree of nodes containing identification tags is specified using XPath expressions. The found tags are replaced by the corresponding values from the personal database. The results obtained are used in practice in the process of educational design of conceptual and logical database models at the Faculty of Informatics. There is a decrease in the complexity of the routine part of the project, achieved through personalization.**

***Keywords*—Personalized document, situation-oriented database, hierarchical situational model, virtual document, VDX, VSDX, ODG, FODG**

I. Introduction

The construction of graphic documents (engineering and infographics) is a mandatory requirement in the process of educational design in many technical and socio-economic subject areas [1–3]. Popular tools for creating vector graphics, diagrams, flowcharts are office graphic editors, such as proprietary Microsoft Office Visio and the free OpenOffice Draw, LibreOffice Draw, etc. They are widely used in the educational process as graphic design tools in various educational disciplines [4–7].

For the implementation of the educational project, students are usually given blanks of graphic documents based on which they create solutions based on their individual tasks. This reduces the complexity of the routine part of the design process. Workpieces are usually performed manually in a graphical editor.

Personalization training, including project activities students – current modern trend [8–10]. Personalization of graphic documents makes it possible to release students from routine activities during instructional design. The possibility of this is due to the emergence of open graphic formats based on XML, allowing software processing of documents [9]. Personalization consists in the fact that the data stored in the database and reflecting the specific features of the project being carried out are programmed into the procurement of project documents (Fig. 1). It can be:

- specifications and details of the project performed by variant;
- personal data of the developer, which must be present in the document being developed;
- the results of the previous stages of the project, which should be present in the document being developed or may serve as the basis for the development of the current stage.

Personalized blanks are formed in two steps: 1) template development; 2) template customization. At the first stage, a blank template with preliminary marking of personalization points is developed in a graphical editor environment. At the second stage, software processing of the template is performed, in which personalization points are found in the template and data from the personal database is placed in them.

It should be noted two levels of personalization and, accordingly, two levels of difficulty of the task, which applies not only to graphic, but also to other types of documents (text, multimedia, etc.).

- *Parametric*. Parameters of figures that are already placed in the blank template are adjusted. For example, the text

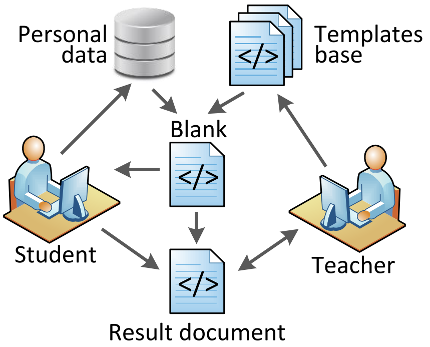


Fig. 1. The use of personalized blanks of graphic documents in the process of educational design

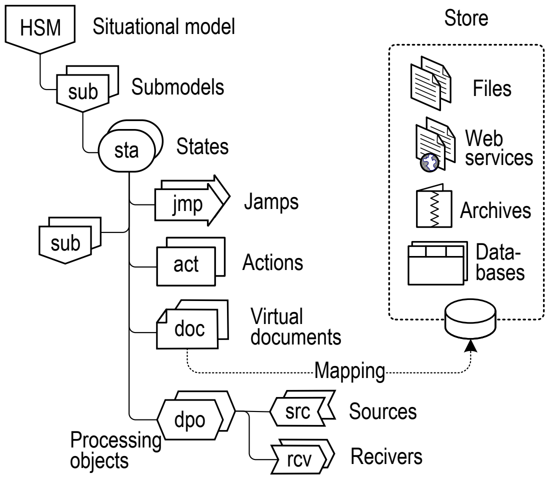


Fig. 2. The hierarchical situational model structure of a situation-oriented database

associated with the figure is entered, or parameters are set that determine its size and position.

- *Structured*. New pieces are placed in the blank, which were not originally there. For example, graphic fragments and figures, developed at the previous stages and necessary as a basis for developing the graphics of the current stage, are introduced. Note that this is a more complex task, requiring more complex software manipulations with the template and graphic fragments extracted from the database.

Here we consider mainly the problem of parametric personalization. Another feature of the approach under consideration is that the solution to the personalization problem is performed within the framework of situation-oriented databases – an integrator of heterogeneous data, on which the authors work.

II. Situation-Oriented Databases and Hierarchical Situational Model

Situation-oriented databases are a project that provides the integration of heterogeneous data based on the information processor controlled by an integrated highly abstract hierarchical situational model [11].

The Hierarchical Situation Model (HSM) sets the functioning of a situation-oriented database. To do this, the HSM is cyclically processed by the HSM interpreter, which monitors its current states and stores them between processing cycles. HSM has two equivalent presentation forms: graphic, which defines the hierarchical structure of elements and is oriented to human perception, and text, which uses XML-like syntax and oriented to programmatic processing by the interpreter. It uses the graphical form of HSM models.

In Fig. 2 shows the general structure of the HSM model. The HSM model as a whole is a collection of `sub` submodels organized in a hierarchy. Each submodel contains many `sta` state elements, which, in turn, can contain submodels as nested ones, as well as elements such as `jmp` transitions and `act` actions. Special types of shares are also provided: virtual `doc` documents and `dpo` processing objects.

Current state management. Interpretation of the HSM model is performed cyclically or upon request within a session. The first cycle of processing a submodel within a session begins with the initial state of this submodel. During the cycle, the current state of the submodel can be changed using transition elements. The current state of the submodel at the end of the cycle is saved and used to start processing in the next session loop. When processing the current state, actions specified by stocks and related to the processing of external data can be performed.

Virtual document concept. Access to external data is specified in the situational model using virtual documents mapped on heterogeneous real data [12–15]. The `doc` elements define virtual documents and determine the actual data to which these real documents are mapped. In Fig. 2 shows the types of external data available for display: local and remote files, web services, archives, relational databases [16–21]. Data processing is performed using processing objects. `Src` source elements load data into memory based on mapping specifications. The actual processing of the downloaded data can be specified in `src` sources and `rcv` receivers. The output of processing results is also carried out using `rcv` receiver elements based on virtual document mapping specifications. Thus, in order to process some data, it is necessary in the HSM model, firstly, to set a virtual document displayed on the corresponding initial data, and secondly, to set up a processing object, determine the loading of virtual document data into it, processing the downloaded data and uploading the processing results.

Mapping on graphic documents. In relation to the problem under consideration, it should be borne in mind that at present, two categories of formats can be used for XML-based graphic document files:

- a document in the form of a single ("flat") XML file. A typical example is the VDX format, an XML file that defines a Visio diagram in DatadiagramML [22]. This format is used in versions of Microsoft Visio 2010 and earlier, while it remains valid due to the wide distribution of these versions. Another example is the FODG (Flat ODG) format, used, for example, in the LibreOffice Draw editor;

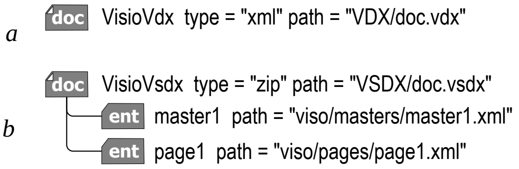

a
doc
VisioVdx type = "xml" path = "VDX/doc.vdx"
b
doc
VisioVsdx type = "zip" path = "VSDX/doc.vsdx"
ent
master1 path = "viso/masters/master1.xml"
ent
page1 path = "viso/pages/page1.xml"

Fig. 3. Defining a virtual document in the HSM model with mapping on a Visio document in formats VDX (*a*) and VSDX (*b*)

- a document in the form of a ZIP archive, in the folders of which XML files are packed that specify various aspects of the image. An example is the VSDX format - Visio chart in accordance with the standard OPC (Open Packaging Conventions) [23]. This format is used in versions of Microsoft Visio 2013 and later. Another example is the format ODG (Open Document Graphics), conforming to standard ODF (Open Document Format) [24] and used in the editors Apache OpenOffice Draw and LibreOffice Draw.

Depending on the category of the format used, software processing of graphic documents should be organized differently: in the first case, the virtual document must be displayed directly on the XML file, and in the second, on the XML files packed in a ZIP archive. In the first case, you need to know the internal structure of the XML file, and in the second, the internal structure of both the ZIP archive of the graphic document and the XML files of interest in it.

Fig. 3 illustrates the task of a virtual document displayed on an external graphic document for these two cases. In option *a*, the `doc:VisioVdx` virtual document is mapped to the `doc.vdx` file located in the VDX folder, and in option *b*, the `doc:VisioVsdx` document to the `doc.vsdx` file located in the `VSDX` folder. In this case, in the second option, the mapping is defined as mapping to a ZIP archive and two paths to access the contents of the archive are specified in it using the attached elements `ent:master1` and `ent:master2`: to file `master1.xml`, placed in folder `visio/masters`, and to file `page1.xml`, placed in folder `visio/pages`.

Fig. 4 illustrates the job in the HSM model of processing a virtual document. The processed virtual document is loaded into the processing object using elements `dom:VisioVdx-Proc` and `dom:VisioVsdx-Proc`. In this case, the processing object is a DOM object (Document Object Model – a “software-independent interface” that provides access to the XML document tree and allows the program to modify XML-document) [25]. The loaded document is specified using the `srcDoc` attribute - links to the virtual document. In the case of the VDX format (Fig. 4, *a*), the entire XML file of the Visio document is loaded into the DOM-object, and in the case of the VSDX format (Fig. 4, *b*), the XML file from the archive of the Visio document (in this case — file `page1.xml`). The `saveDoc` attribute, which refers to the same virtual document, instructs, upon completion of processing, to save the modified XML file in the ZIP archive in the same place.

Processing a graphic template in a “flat” XML file format is easier than in a ZIP archive format. The XML file, being fully loaded into the DOM object and personalized in it, can be immediately sent to the addressee (student). The ZIP archive must be processed in parts, the processed parts must be returned to the archive in the same place so as not to violate the integrity of the document. In these conditions, you should make a copy of the original template, subject it to sequential personalization in parts, and then send it to the addressee. For example, difficulties may arise if several parts of a document are to be processed together.

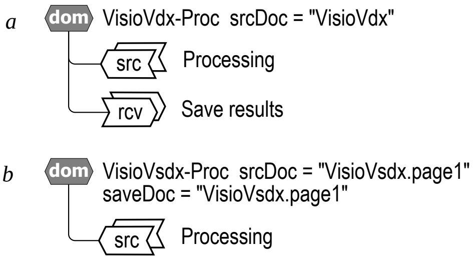

a
dom
VisioVdx-Proc srcDoc = "VisioVdx"
src
Processing
rcv
Save results
b
dom
VisioVsdx-Proc srcDoc = "VisioVsdx.page1"
saveDoc = "VisioVsdx.page1"
src
Processing

Fig. 4. The general scheme of processing a virtual document in the HSM model for formats VDX (*a*) and VSDX (*b*)

The actual processing in Fig. 4 is not detailed, only the elements performing it are indicated - sources and receivers. Details are explained below.

III. Preparation of Templates and Their Processing During Parametric Personalization

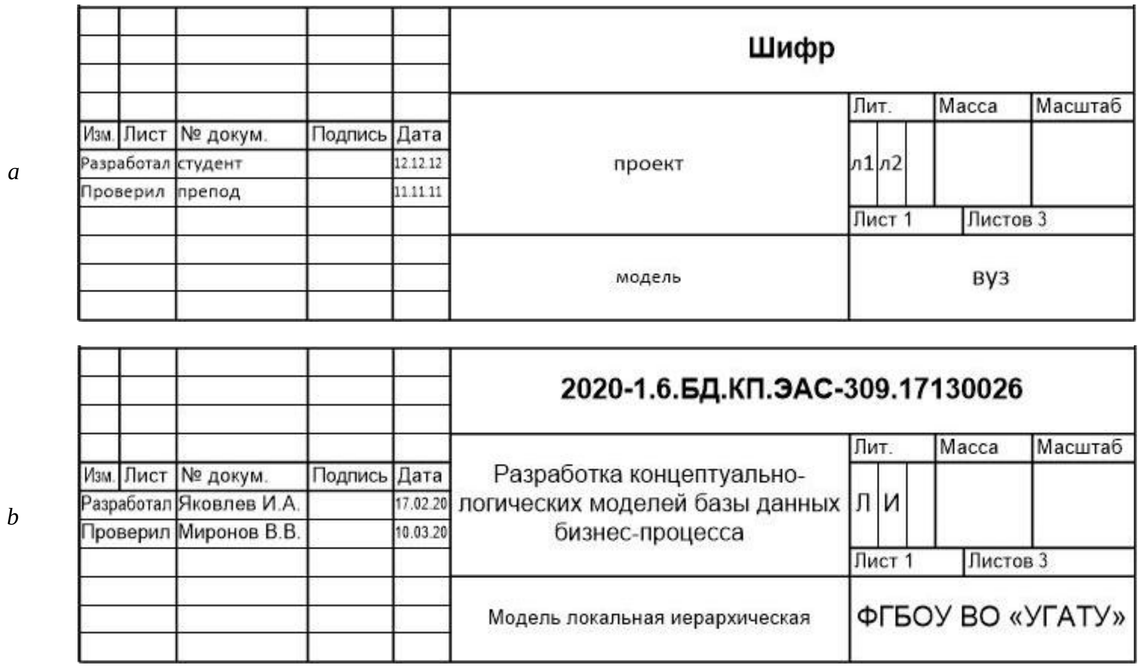
Personalized blanks are formed in two steps: 1) template development; 2) template customization. At the first stage, a blank template with preliminary marking of personalization points is manually developed in the environment of the graphic editor. At the second stage, software processing of the template is performed, in which personalization points are found in the template and personal data from the database is placed.

Personalization template. A template is a graphic document containing fragments of the resulting image. The image is built from logical elements – figures. The figures contained in the template may in one form or another be present in the resulting document. During the educational design, the student can adjust them in terms of size, location, connection, explanatory inscriptions, etc.

In Visio documents, figures can be present in two forms: in the form of the image itself, placed on the sheets of the document, and in the form of sample shapes (stencils - stencils), which are attached to the document and can be used to build the image itself. In OpenOffice/LibreOffice Draw documents, attached sample shapes are not provided, there are only image shapes.

A. Use of identification tags

With the parametric personalization discussed in this article, you must manually set the personalization points in the existing shapes of the template (in the image and / or in the attached set) in the form of identifying marks that would allow you to uniquely find the shape and adjust it accordingly. Each text can be associated with some text. The easiest way to mark an image is by manually placing unique text in it. Detection of this text during software processing will allow to identify the figure itself.



a

Шифр

Изм.	Лист	№ докум.	Подпись	Дата
Разработал		студент		12.12.12
Проверил		препод		11.11.11

проект

Лит. Масса Масштаб

л1 л2

Лист 1 Листов 3

модель

вуз

b

2020-1.6.БД.КП.ЭАС-309.17130026

Изм.	Лист	№ докум.	Подпись	Дата
Разработал		Яковлев И.А.		17.02.20
Проверил		Миронов В.В.		10.03.20

Разработка концептуально-логических моделей базы данных бизнес-процесса

Лит. Масса Масштаб

Л И

Лист 1 Листов 3

Модель локальная иерархическая

ФГБОУ ВО «УГАТУ»

Fig. 5. An example of placing identification marks in the title block of a document (*a*) and the result of personalization (*b*) (in Russian)

In Fig. 5 *a* shows an example of identification marks in the title block of a design document template. Russian words "Шифр" (cipher), "проект" (project), "модель" (model), "вуз" (university), "студент" (student), "преподаватель" (teacher), "12.12.12", "11.11.11", etc. are identification marks entered by the developer in the unchanged image of the template. At the stage of personalization, these labels will be found in the template replaced with personal data: document code, project name, university name, surnames and initials of the student and teacher-consultant, development and verification dates, etc. (Fig. 5, *b*). Working in the environment of a graphical editor, a developer can simply select the places for making identification marks and filling out their personal data. By setting the visual properties of labels (color, size, font, etc.), the developer thereby sets the display properties of the corresponding personal data.

Software processing of templates for parametric personalization requires consideration of the internal organization of graphic documents. To find the identification tag in the template and substitute a personal value instead, you need to know the internal structure of the document. The study of the available information about the internal structure of the VDX / VSDX and ODG / FODG formats, as well as the direct study of the XML markup of graphic documents using XML text editors, made it possible to identify information about the features of various graphic formats that are significant in terms of the problem being solved.

B. *Documents as a "flat" XML file*

In Fig. 6 shows models of the structure of graphic documents in VDX and FODG formats. The presented XML markup tree contains a hierarchy of only those XML elements and their attributes that are relevant to the text associated with the image shapes, which is essential for the considered parametric personalization task. In order not to clutter up the picture, the markup elements and attributes are omitted that specify the properties of the figures that are insignificant for the problem being solved, such as position on the sheet, sizes, colors, etc. In the applied graphical notation, the names of XML elements are specified with an underscore, XML attributes without an underscore, the text content of an XML element is indicated by an equal symbol. The nesting of XML elements is shown using the "down-right" connectors, the light triangle indicates the possibility of zero, one or more instances, and the light circle indicates the possibility of zero or one instance of a nested (child) element.

VDX template. A Visio VDX document (Figure 6, *a*) is an XML document with a `VisioDocument` XML root element. Two of its child XML elements are important for this task: `Masters` and `Pages`. The `Masters` element defines a set of sample shapes (master figures, stencil shapes), and the `Pages` element defines the sheets of a Visio document that contain the actual image. Sample shapes can be used to build images on sheets of a document. When changing the properties of the sample shapes, the corresponding default properties of the image shapes on the sheets based on these samples are changed by default.

The `Masters` XML element contains `Master` children, each of which defines a separate master shape. XML attribute `ID` contains the identifier of the sample, attribute `NameU` - its name (other attributes are omitted). `Shapes` child XML element defines many sample shapes. Each of these shapes is defined by its `Shape` XML element nested in the `Shapes` element. The XML `ID` attribute specifies the shape identifier, the `NameU` attribute specifies the shape name. The remaining attributes that specify the numerous properties of the figure are not shown in the figure. Numerous XML child elements are also not shown, excluding of the `Text` element, which defines the text associated with the shape.

Note the recursive nature of the structure of the figures: a complex figure may include several nested figures, which, in turn, may contain other nested figures. This is reflected in the model using a connector that attaches the parent `Shapes` as an (optional) child to the child `Shape`.

The Pages XML element contains child `Page` elements, each of which defines a separate sheet of the document. XML attribute `ID` contains the identifier of the sheet, attribute `NameU` is its name (other attributes are omitted). The `Shapes` child XML element defines the many shapes that are placed on the sheet. The `Shapes` element has the same internal structure as in the `Masters` element.

Note that these `Shape` XML elements may have an optional `Master` reference attribute. It contains the value of the identifier `ID` pointing to the `Master` element, if this figure on the sheet was obtained from the corresponding sample figure. Keep in mind that in this case the `Shape` element sets only those properties of the figure on the sheet that differ from the properties of the sample figure.

The `Page` element also contains a nested `Connections` element, which defines many connections of image shapes on a sheet. Each connection is described by its `Connect` element.

Thus, the identification tag is the text content of some XML `Text` element nested in some XML `Shape` element, which, in turn, is nested in the parent `Shapes` element. This element is a descendant of the `Master` element if the label is placed in the sample shape, or a descendant of the `Page` element if the label is placed in the figure on the sheet.

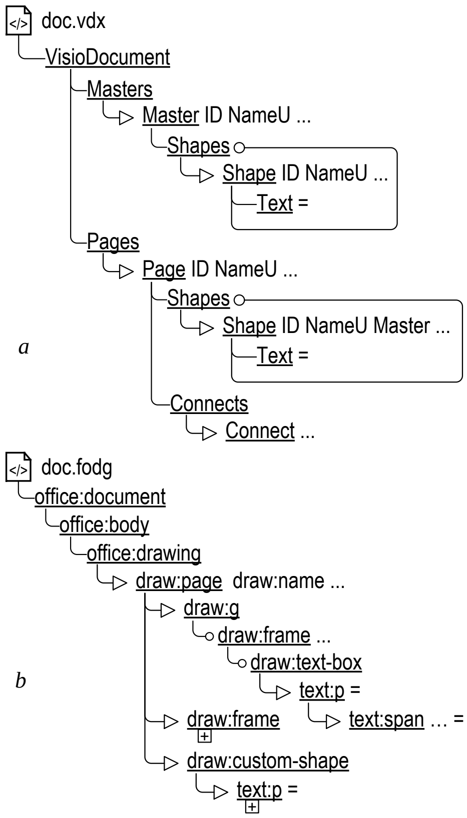


Fig. 6. Model XML-markup of documents VDX (*a*) and FODG (*b*)

FODG Template. A document in the FODG format (LibreOffice Draw et al., Fig. 6, *b*) is an XML document with an XML root element `office:document`. For the task at hand, its offspring element `office:drawing` is important with nested `draw:page` elements corresponding to individual sheets of the document (the sheet name is specified by the `draw:name` attribute). The text fragments present on the sheet are represented in the XML markup as the text content of the element from the namespace with the prefix "`text`": the paragraph element `text:p` or the spacing element `text:span` nested in the paragraph element. A paragraph element, in turn, can be nested in elements such as `draw:text-box`, `draw:frame`, `draw:custom-shape`.

Thus, the identification tag in the case of the FODG format, as in the case of the VDX format, is the text content of some XML element. Note that in the FODG format there is no concept of sample shapes attached to the document. Figure performing this role, it is necessary to place in the blanks on the paper sheets themselves.

C. *Documents as a ZIP archive*

The distributed structure of graphic documents determines a more complex procedure for their personalization. In Fig. 7 shows the structure of the main components of a ZIP archive of documents in VSDX and ODG formats.

Features of the template in VSDX format. In Fig. 7, *a* show the structure of the main components of the ZIP archive of the document in VSDX format. Content information about the image of the document is collected in the `visio` folder, where in the subfolder pages there is information about the image on the sheets, and in the `masters` folder about sample shapes attached to the document. Each sheet of the document has its own XML file: `page1.xml`, `page2.xml`, ..., in addition,

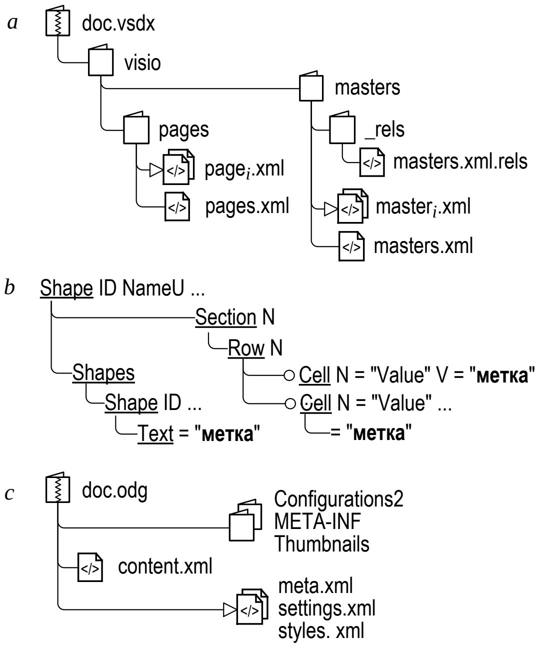


Fig. 7. The main components structure of the ZIP-archive document formats VSDX (*a*, *b*) and ODG (*c*)

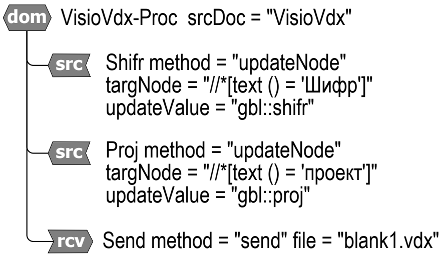

dom VisioVdx-Proc srcDoc = "VisioVdx"
src Shifr method = "updateNode"
targNode = "//*[text () = 'Шифр']"
updateValue = "gbl::shifr"
src Proj method = "updateNode"
targNode = "//*[text () = 'проект']"
updateValue = "gbl::proj"
rcv Send method = "send" file = "blank1.vdx"

Fig. 8. A fragment of the HSM model for personalizing a document in the format VDX

the `pages.xml` file contains information that applies to all sheets. Similarly, each sample document has its own XML file: `master1.xml, master2.xml, ..., in addition`, the `masters.xml` file contains information related to the entire set of samples.

Note that if the `masters.xml` file contains general information for each sample, then the `master1.xml`, `master2.xml`, ... files contain detailed information on the sample shapes. So, it is in these files that the identification marks of the sample figures will be placed. The relationship between the general and detailed information of the sample figures is reflected in the file `masters.xml.rels`, which lies in the `visio/masters/_rels` folder.

Although there are differences in the XML markup of the VSDX and VDX formats, the internal XML markup of the `master1.xml`, `master2.xml`, ... files is generally similar to the XML markup of the `master` element, and the `page1.xml`, `page2.xml`, ... files are XML markup of page element in VDX format. An important feature of VSDX markup that you need to keep in mind is that the text of the identification mark can be represented in the XML markup of the document both in the form of the text content of the XML element and in the form of the value of the XML attribute. In Fig. 7, *b* shows how in the `Shape` element the text “`label`” can be set as the value of the `V` attribute of the `Cell` element, as the text content of this element, as well as the text content of the `Text` element.

Features of the template in ODG format. In Fig. 7, *c* shows the structure of the main components of the ZIP archive of the document in ODG format. The information that interests us that sets the image is contained in the `content.xml` file located in the archive at the top level. The root XML element in this file is the `office:document-content` element, which contains the nested `office:body` element (see Fig. 6, *b*), which is identical to the FODG element of the same name. Thus, in this case, as in the case of the FODG format, the identification tag corresponds to the text content of the corresponding XML element.

D. *Addressing labels using XPath-expressions*

In the process of personalization, you need to find an XML element (or XML attribute in the case of the VSDX format) in the document, the value of which coincides with the value of the identification label, and replace this value with a new one. The first part of this task - the search task - can be solved using XPath [26]. XPath lets you specify expressions that address specific nodes in the tree of an XML document.

So, an XPath expression

```
//* [text () = 'value']
```

addresses all XML elements with a given "`value`" of text content. For unambiguous identification, it is necessary that the value of the identification label is unique throughout the document. The expression

```
//Master//Text [text () = 'value']
```

addresses all XML `Text` elements with specified text content that are descendants of `Master` elements. That is, this expression searches among the sample Visio document shapes. For unambiguous identification, it is necessary that the value of the identification label is unique within all the sample figures attached to the document. The expression

```
//Page [NameU = 'имя']//* [text () = 'value']
```

addresses all XML elements with a given "`value`" of text content that are a descendant of a `Page` element with a given "`name`". That is, this expression searches among the shapes on a given sheet of the document. For unambiguous identification, it is necessary that the value of the identification label is unique within this sheet. Finally, the expression

```
//* [text () = 'value'] | //@* [. = 'value']
```

addresses all XML elements and all XML attributes with a given `value`. This option is convenient for the case when it is not known in advance how the label in the document is presented - as an element or as an attribute.

Requirements for identification marks. When defining identification tags in a graphical editor environment, it is necessary to ensure their uniqueness in a certain range, as well as atomicity, i.e. uniformity of the label text format. The last requirement is that the label in the XML tree of the document be represented by a single node, accessible for addressing with a simple XPath expression, and not be split into several nodes with different formatting. Therefore, it is advisable to label short words without spaces or control characters.

E. *Personalization identification tags*

Thus, the search for identification labels can be set with the help of XPath-expressions. The next task is to replace the found tags with the corresponding personal data. To implement this functionality in HSM, `src` sources `updateNode` were provided as part of DOM processing elements oriented to work with DOM objects, which allow searching the nodes of a given value in the XML document tree and setting a new value.

In Fig. 8 shows the HSM-model of the DOM-object, providing personalization of identification marks in the document in the form of a "flat" XML-file within the framework of the general scheme shown in Fig. 4, *a*. In the DOM object `dom:VisioVdx-Proc`, a Visio document in VDX format is loaded in accordance with Fig. 4, *a*. Using the nested `src:Shifr` and `src:Proj` source elements, the user searches the document for identification tags and replaces them with personal data. Then, using the `rcv:Send` receiver element, a personalized document is sent to the user.

The figure shows two source elements that personalize two identification tags. The `src:Shifr` source sets the search and personalization of the “Cipher” label (see Fig. 5). The `method = "updateNode"` attribute instructs the node to be updated in the tree of the loaded XML document. Атрибут `targNode` содержит XPath expression addressing the updated node. The expression `"//* [text () = 'Шифр']"` instructs to find an XML element whose text content is equal to the value `'Шифр'`. The attribute `updateValue = "glb::shifr"` sets the new value of the updated node as the value of the global variable `shifr`. The `src:Proj` source, which sets the value of the “`project`” label, and other sources not shown in the figure are constructed in the same way.

The document is processed in a similar way in the form of a ZIP archive within the framework of the general scheme shown in Fig. 4, *b*.

Thus, the provided possibility of using XPath expressions for addressing XML nodes and the ability to replace text values of nodes makes it quite simple to specify parametric personalization based on identification marks in the HSM model, uniformly for different formats of graphic documents based on XML.

IV. APPLICATION IN THE EDUCATIONAL PROCESS

The proposed approach was implemented and tested on the research prototype of the HSM model interpreter, executed on the PHP platform and hosted on the web server of the Ufa State Aviation Technical University (USATU). Practical application is implemented for information support of the design process on the site of the course design in the discipline of "database" (http://hsm.ugatu.su/artem/dbproj/).

The serviced course project provides for the development of conceptual and logical database models for information support of the business process in accordance with individual tasks. Design includes 12 successive stages, 8 of which include the development of graphic models of different levels of detail: from local hierarchical to global relational. For each stage of modeling, templates were developed, labeled in accordance with the above approach. At the beginning of the next stage of designing, the student is given the opportunity to download workpieces in the VDX format from the web server, personalized on the fly, taking into account the executor, the task option and the stage to be performed. Personalization includes, firstly, filling out the main inscription of design documents with personal data in accordance with the ESKD standard (see Fig. 5), and secondly, posting information from previous stages that may be useful at this stage, such as a set of figures- samples needed to build a model. Having received a personalized blank, the student, using a graphical editor, builds the required model on its basis and uploads it to the server. There, the document passes the correctness check programmatically and becomes available for verification by the teacher-consultant.

The practical application of the system over several academic semesters has shown the efficiency and effectiveness of the proposed approach. Due to the personalization of blanks of graphic documents, students have reduced the time spent on the technical side of the preparation of project documentation, and, accordingly, increased the time for the actual design.

Further development of the approach. In the current implementation of the system, processing is provided only for the VDX format, which dictates the use of the Visio 2010 editor. It is planned to provide the possibility of simultaneous operation of the system with various graphic formats, which will make it possible for students to personally select the applied graphic editors.

In the future, it is proposed to extend the proposed approach to structural personalization. When developing database models, the next model is based on the previous ones, which requires copying previously constructed fragments and figures. It is planned to work out the issues of isolating from graphic documents the informative information necessary for the next stages of design, and the formation of personalized blanks with the placement of the necessary image elements.

V. CONCLUSIONS

Thus, the article considers the solution to the problem of personalizing graphic documents based on situationally oriented databases, which has the following features:

- graphic documents are presented in XML-oriented formats, both in the form of a “flat” XML file (VDX, FODG), and as a ZIP archive of XML files (VSDX, ODG);
- preliminary preparation of the graphic template is carried out manually in the environment of the graphic editor using text identification tags;
- software processing of the template is performed in DOM objects using virtual documents displayed depending on the format on the entire graphic document or on its part in the ZIP archive;
- addressing of identification marks in the template is set by XPath-expressions that take into account the peculiarities of XML-marking of graphic documents in various formats;
- the efficiency and effectiveness of the proposed approach is confirmed by practical use in the information support of educational design of conceptual-logical database models.

ACKNOWLEDGMENT

Russian Foundation for Basic Research grant No 19-07-00682.

REFERENCES

[1] Baswara S. Y., Widhiastuti R., Dewi L. C. Learning Model Based on Information Technology in an Accounting Education Courses Based on Technology at Faculty of Economics in Universitas Negeri Semarang // KnE Social Sciences. 2020. 4 (6). С. 1280–1285. https://doi.org/10.18502/kss.v4i6.6678.

[2] Yu Z., Xiong Z. Comparative analyses for the performance of Rational Rose and Visio in software engineering teaching // Journal of Physics: Conference Series. IOP Publishing, 2018. Т. 1087, № 6. С. 062041. doi:10.1088/1742-6596/1087/6/062041

[3] Medoh C., Telukdarie A. Business process modelling tool selection: A review // 2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). IEEE, 2017. С. 524–528.

[4] Sarvepalli A., Godin J. Business Process Management in the classroom // Journal of Cases on Information Technology (JCIT). 2017. Т. 19, № 2. С. 17–28.

[5] He L., Lian J. Instructional Design of Practice Course of Logistics System Planning and Design Based on Visio // 2018 9th International Conference on Information Technology in Medicine and Education (ITME). IEEE, 2018. C. 526–530.

[6] David J Parker. Mastering Data Visualization with Microsoft Visio Professional 2016. Packt Publishing Ltd, 2016.

[7] Chernykh K.V., Kozhukhova A.V., Tolmacheva L.V. Use of the OpenOffice.org software package in the professional activity of an engineer // Applied information systems in land transport technologies (mechanical engineering). 2019.S. 5–8. (In Russian).

[8] Xu, Y., Zhang, M., Gao, Z., The Construction of Distance Education Personalized Learning Platform Based on Educational Data Mining, Advances in Intelligent Systems and Computing, 1017, pp. 1076-1085, 2020.

[9] Liu, D. Y.-T., Atif, A., Froissard, J.-C., Richards, D., An enhanced learning analytics plugin for Moodle: Student engagement and personalised intervention, ASCILITE 2015 — Australasian Society for Computers in Learning and Tertiary Education, Conference Proceedings, pp. 180-189, 2019.

[10] Lee, D.H., Cho, S.H., Kim, Y., A design and development of the learning contents management based on the personalized online learning, International Journal on Advanced Science, Engineering and Information Technology, 8(4), pp. 1321-1326, 2018.

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

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

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

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

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

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

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

[18] V. V. Mironov, A. S. Gusarenko, and N. I. Yusupova, 'Situation-oriented databases: polyglot persistence based on REST microservices', Applied Informatics, vol. 14, no. 5(83), pp. 87–97, 2019, doi: 10.24411/1993-8314-2019-10038. (In Russian).

[19] V. V. Mironov, A. S. Gusarenko, and N. I. Yusupova, 'Application of Web Services Based on Situation-Oriented Database for Monitoring the Viewing of the Educational Video-Content', Modeling, Optimization and Information Technology, vol. 7, no. 4, Oct. 2019, doi: 10.26102/2310-6018/2019.26.3.031. (In Russian).

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

[21] Valeriy Mironov, Artem Gusarenko, and Nafisa Yusupova, 'Stream Documents Processing Invariance in Situation-Oriented Databases', in 7th Scientific Conference on Information Technologies for Intelligent Decision Making Support (ITIDS'2019), 2019, doi: 10.2991/itids-19.2019.55.

[22] 'DatadiagramML Schema'. http://m8y.org/Microsoft_Office_2003_XML_Reference_Schemas/Help/html/vixmlconSchema.htm (accessed May 17, 2020).

[23] 'Standard ECMA-376'. http://www.ecma-international.org/publications/standards/Ecma-376.htm (accessed May 17, 2020).

[24] 'Publicly Available Standards'. https://standards.iso.org/ittf/PubliclyAvailableStandards/c066363_ISO_IEC_26300-1_2015.zip (accessed May 17, 2020).

[25] 'Document Object Model (DOM) Level 1 Specification'. https://www.w3.org/TR/REC-DOM-Level-1/ (accessed May 17, 2020).

[26] 'XML Path Language (XPath)'. https://www.w3.org/TR/1999/REC-xpath-19991116/ (accessed May 17, 2020).