Using Domain-Specific Modeling for Integration of Heterogeneous Business Activities

Verislav Djukić[1], Marko Bošković[2], Aleksandar Popović[3], Ivan Luković[4]

[1]Djukic – Software Solutions, Germany, info@djukic-soft.com
[2]Research Studios Austria, Austria, marko.boskovic@researchstudio.at
[3]Faculty of Science, University of Montenegro, aleksandarp@rc.pmf.ac.me
[4]Faculty of Technical Sciences, University of Novi Sad, Serbia, ivan@uns.ac.rs

Abstract. This paper presents an approach to integration of heterogeneous business activities based on software models and four-level architecture of Domain-Specific Modeling. Business activities are modeled using document templates which contain description of both static and dynamic characteristics of a system. Document templates are managed using software models in different Domain-Specific Graphical Languages. Document content increments and layout changes are related to progress in business activities. It is shown how to manage different models of document production at the highest possible level of abstraction. The majority of this paper is dedicated to the code generator, used for transformation of abstract layout and lifecycle models to different concrete specifications, such as XML, HTML, PDF, SVG and DVDocLang.

1 Introduction

Theory and practice of Model-Driven Software Development (MDSD) has recently meet many demanding requirements placed in front of Software Engineering. Generally, to apply MDSD techniques, there are two directions: one is the use of General-Purpose Modeling Languages such as UML, and the second is Domain-Specific Modeling (DSM) with Domain-Specific Languages (DSL) [14,16]. The notable increase in development productivity, the quality of code that is generated from abstract models, as well as flexibility of constructed systems in such way is a clear motivation for applying DSM methodology in wider and more complex domains. Current experiences in the deployment of MDD in document engineering justify an application of both DSM principles and DSL-s in specification and production of documents. Two currently popular modeling tools are MetaEdit+ Modeler [1,4] and Eclipse Modeling Framework (EMF) [22]. The first tool is commercial and already long time available at the market, while the second one is the result of the wider initiative to develop a free tool for the industry purpose. Apart from some advantages, current shortcomings of the EMF technology make it oft unattractive to software engineers in industry. The main practical reasons which significantly hinder the use of EMF are:

- A lack of repository for fast access to meta-data;

- Incompatibility of current with previous versions;
- Weak synchronization between created meta-models and models; and
- Weak support for refinement of DSL-s.

EMF allows creating extensions of the supported concepts. In this way, some companies have significantly customized and improved EMF and by this established their own internal standard for the application of EMF in their industry projects.

The main goal of this paper is to present an approach to integration of heterogeneous business activities based on DSM and DSL-a. We also present DVDocIDE, an integrated development environment that supports Model-Driven Document Engineering. Our approach deploys (i) principles of Model-Driven Development methodology described in [1,3,4], (ii) Document Engineering, described in [2,7,9,15,29], and (iii) the MetaEdit+ Modeler tool. For the problem illustration we use examples from the domain of Directory Publishing. To apply MDD in Document Engineering, till now we have developed the appropriate DSL-s, a Domain Framework, and PDF and HTML renderers, as described in [8,15,18]. In this paper we present:

- Management of document templates using domain-specific abstract models;
- Use of DSL diagrams for model-driven business applications;
- A language for transformation of abstract to concrete document templates; and
- A semantically based template editor, named DVDocEditor that supports our approach to the management of document templates.

The paper is structured as follows. In Section 2 we present a running example of document production model. A use of DSL-s for specification and management of the production model is given in Section 3. Section 4 outlines the use of DSL diagrams in business applications. The code generator and the DVDocRep Language for transformation of abstract to concrete document models are described in Section 5. In Section 6 we present a DSM approach based on the usage of an appropriate semantically based graphical editor, as an alternative to existing topologically oriented editors. In Section 7 is conclusion.

2 An example of document production model

For the purpose of illustration of DSM usage, we introduce here an example of a realistic document-centric system for production of business advertisements with its accompanying documents. In Fig. 1 it is presented the state-activity graph of the example. From all the requirements that depict a complex nature of such production system, we emphasize the following ones:

- Every change or progress in activity is documented with an appropriate PDF document. In other words, every state of a lifecycle is being documented;
- Production of logos can be manual or automatic. It is preferable to produce advertisements using as simple as possible Domain-Specific Language;

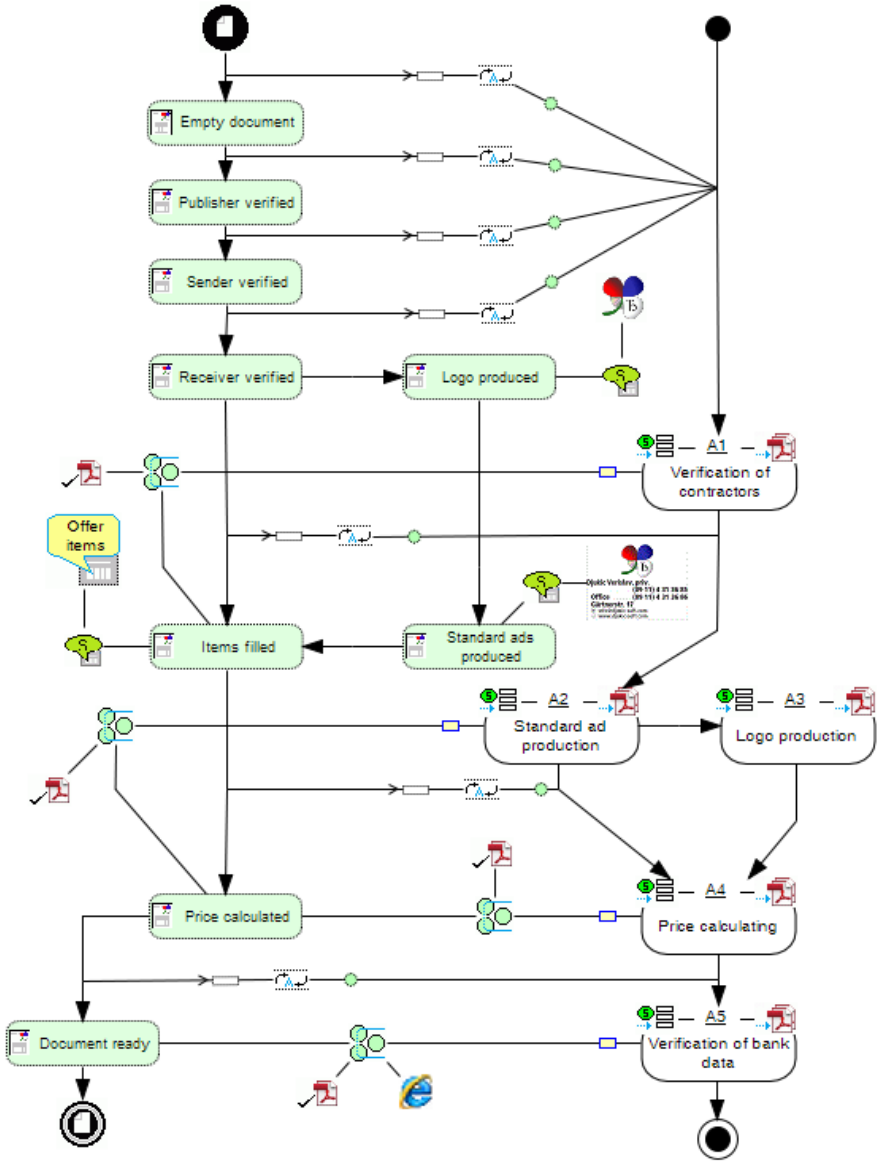

Fig. 1. A production model in Directory Publishing

- Different companies have different business rules and different legacy applications for automation of production;
- Documents must be in PDF format due to the required layout and print quality. For the last document states web page with the same content are generated;

- The number of document types and possible deviations from a base type is large and it is important to enable systematic refinement of their modeling language;
- Every activity can have its own DSL for specification of content units; and
- Synchronization of activities is carried out using messages or events whose parameters are PDF documents.

By this, we identified and presented here a list of requirements which is in some extent simplified and generalized. This generalization allows their recognition in various business domains. In the following text, we discuss specific concepts that are included in the activity graph from Fig. 1.

3 Construction and use of DSL for specification and management of production models

Using MetaEdit+ Modeler [4] a DVDocFlow DSL is constructed. DVDocFlow is aimed at expressing reporting rules within the business activities. The example of a usage of its main concepts is illustrated by Fig. 1. DVDocFlow language concepts are similar to the UML state/activity concepts. It includes concepts of state and activity, but also some specific concepts, such as: content units, layout elements, as well as appropriate relations, roles and ports. The main reason why we developed our own DSL instead of deploying UML for modeling specific business activities is in the following: Our goal was not to support software modeling and code generation only, but also document production management by means of software models. Besides, UML is a family of general purpose languages and a lot of meta-modeling customizations are needed to address all the problem domain requirements.

On the diagram from Fig. 1, states, activities and documents are put in relations. For the reasons of simplifying, only one content unit “Offer items” has been shown. It is a symbol left in the middle of the Fig. 1. Layout concepts are not shown. However, they are discussed in Sections 4 and 5. In the production model for “Offers” the following activities have been identified:

- A1- Verification of pages in the contract – A composite activity consisting of the verification by the ordering, executive and publishing entities;
- A2 – Advertisement production – It may be based on: (i) general purpose tools not using meta-data or (ii) simple DSL-s;
- A3 – Logo production – It is mainly based on general purpose tools;
- A4 – Calculation of production and publishing price. It is preferable to carry out the activity automatically, by applying already specified business rules; and
- A5 – Verification of payment data.

Apart from activities, relevant states have been identified in the document lifecycle. The states are denoted in Fig. 1 with dotted rounded rectangles. The first state in Fig. 1 is “Empty document”, while the last one is “Document ready”. Edges decorated with the icons of three small circles in Fig. 1 specify reporting activities – in which state and how it is being reported. Those are the n-ary relations. When a PDF icon is

associated to the relation, the state that is in relation with an activity is being documented by rendering a PDF document. The analogous holds for the web browser icon associated to the relation, when an HTML page is generated. Activities A1, A4 and A5 are similar in their nature, while A2 (small ad production) and A3 (logo production) are specific. A2 is the activity that is supposed to be fully automated by using DSL-s. A more detailed description of a language for automation of A2 has been given in [6,8,18]. The requirements to tools for construction of the modeling language are to: (i) provide a flexible way of modeling production specifics in different organizations; (ii) reduce a number of document templates; and (iii) provide a production management by means of a graphical interface.

The first requirement is satisfied by supporting various complex production models. In one case, as an example, A2 and A3 do not exist as separated, but rather as one activity. The second case is that the activity A2 is very complex, when there are different advertisement production models for different types of advertisements. The third possible case is that the production offer is one completely automated activity comprising the whole document formal description and generation. In many real cases, the verification of data about ordering, publishing and production entities are carried out within one activity. The problem becomes complex due to a demand to use the same document layout templates for different production models. Additionally, there are organizations whose activities are subjects to rigorous rules of quality assessment and work scheduling, where a document becomes the only mean of proof. Our point of view is that the content, structure, layout, behaviour and meta-data should be viewed as logically related document dimensions, in order to provide tools for their specification and rendering. There are some recent initiatives, ideas and small implementations in [10,21,22,23,24] trying to put in relationship layout and content definitions, but not all dimensions.

4 The use of diagrams in business applications

Here we demonstrate the benefits of using DSM tools for business process management. DVDocFlow language, modeling tool and code generation can be used:

- As graphical interface for client applications;
- For integration of individual DSL for some particular purpose;
- For implementation of a workflow engine; and
- For implementation of a document management system (DMS).

Pictures and meta-data used on a particular instance of graph can be exchanged with client applications using properties and generators. A collection of objects is traversed and an XML structure is created, that contains identifications and state names, as well as positions and symbol dimensions of objects. If there are composite states, XML contains also an identification of the appropriate sub-graph. It is useful to integrate all meta-models of individual DSL-s to one meta-model. Over the XML structure, whose schema is implicitly defined using graph and this generator, the following generic operations are implemented:

- Select all documents for a given state and type: SelectWhere(docType,stateID);
- Select all documents for given type, neglecting their state: SelectAll(docType);
- Go to the next document state: NextState(docID,stateID);
- Assign a figure to the state: SetInstancePct(docID,pctID);
- Display a composite state: ExpandState(docID,steteID);
- Generate an application for a transition: CreateApp(docID,nextState);
- Find the activity that leads to particular state : ActivityForState(docID,stateID);
- Render PDF or HTML: CreatePDF(docID), CreateHTML(docID);
- Display an instance in a particular state: ShowInState(docID, stateID); and
- Display all instance states: ShowAllStates(docID).

Using these generic operations, a part of functionality for the workflow engine and document management system is implemented. End-user document management is based on client application (Fig. 2) and meta-data placed in each document instance.

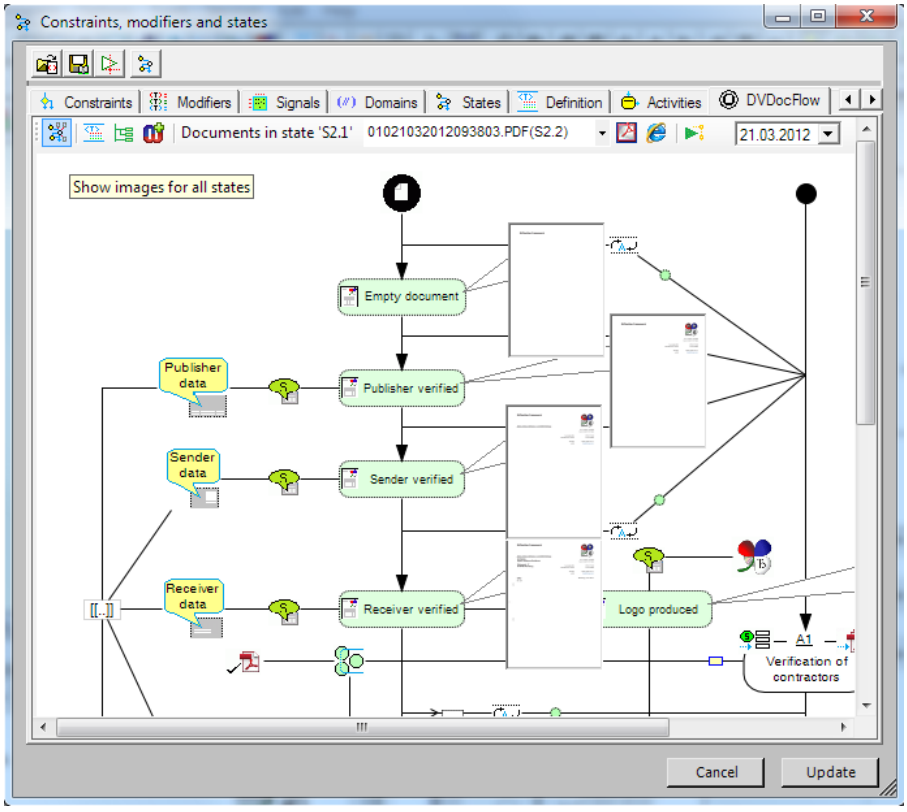

Fig. 2. A control that uses formally described production models in DSL

The set of document instances can be used as a database [19], and in that case, some operations of browsing are traversing meta-data from PDF or HTML files. More detailed description of all meta-data is given in [6].

Fig. 2. shows a user control in which are implemented operations for browsing and incremental specification and rendering of documents. For the purpose of DVDocIDE synchronization with MetaEdit+ Modeler, a plug-in has been made, whose source code and videos that demonstrate control can be found in [28]. User control in Fig. 2. uses graph and metadata from the MetaEdit+ Modeler. Selection of all documents for a given state is called with double click of mouse on the target state. By means of the upper-left corner toolbar buttons we provide: (i) the overview of all instance states; (ii) the overview of the DVDocFlow textual representation in DVDocLang format; (iii) the insertion of metadata from the archive; and (iv) the call of MetaEdit+ Modeler to edit the production model. Drop down list contains all documents that are selected according to the given criteria. Using the PDF button and the browser symbol, a PDF or a HTML document is shown, respectively. Applications are generated incrementally, on a state change request. The arrow is a transition invocation in some of the next possible states. When scrolling over a particular state image, an overview of the document in that state is given. Depending on the number of documents and their production kind, some of the operations are implemented using SQL queries over database. These queries are generated using MERL code generator and there is no need for writing them separately for each production model.

5 Code Generation and transformation of abstract to concrete models using DVDocRepLang

This Section describes DVDocRepLang, a language for transformation of platform independent (PIM) to platform specific document models (PSM). From the perspective of DSM architecture, this is language for specification of code generators. Comparing to the EMF and MetaEdit+ Modeler, this language offers large possibilities for transformations of layout attributes. It can be used for any graphical editor that is implemented in .Net. Beside explicitly given properties, it is also possible to get/set system properties of .Net UserCtrl-a, as well as, properties of other controls for visual presentation of content (text editors, tables, figures, etc.). A particular attention is paid to simplify transformations of topological and semantic relationships between layout elements. Detailed description of DVDocRepLang with his syntax and examples is given in [20]. Code generator in Example 5.1 illustrates structural patterns and is related to the Fig. 3.

Input in the transformation is at least one diagram and at least one report generator. Inputs can be also files, console input data, parameters and results of some other generators. The output of the transformation is texts and figures, organized in textual streams and files. Parts of the content can repeat to several outputs.

Diagram elements are referenced using the following commands: for object **.[name;],** for relations **>[name;],** for roles **#[name;],** for ports **~[name;]** for properties **:[name;].** Collections and loops are being declared using the **FOREACH {setDef}** command, where **setDef**, is a constant or a set of diagram elements of the same type. They can be filtered using the **WHERE 'pattern*'** command and sorted using **'ORDERBY' orderCriterion {',' orderCriterion**} command. Branching is

specified using **IF (cond) THEN (block) ELSE (block)**. Loop and branching commands can be nested. The generator interpretation starts from one or a collection of objects, relations or roles and traverses the diagram. Internally, the current element and current output is saved on the stack.

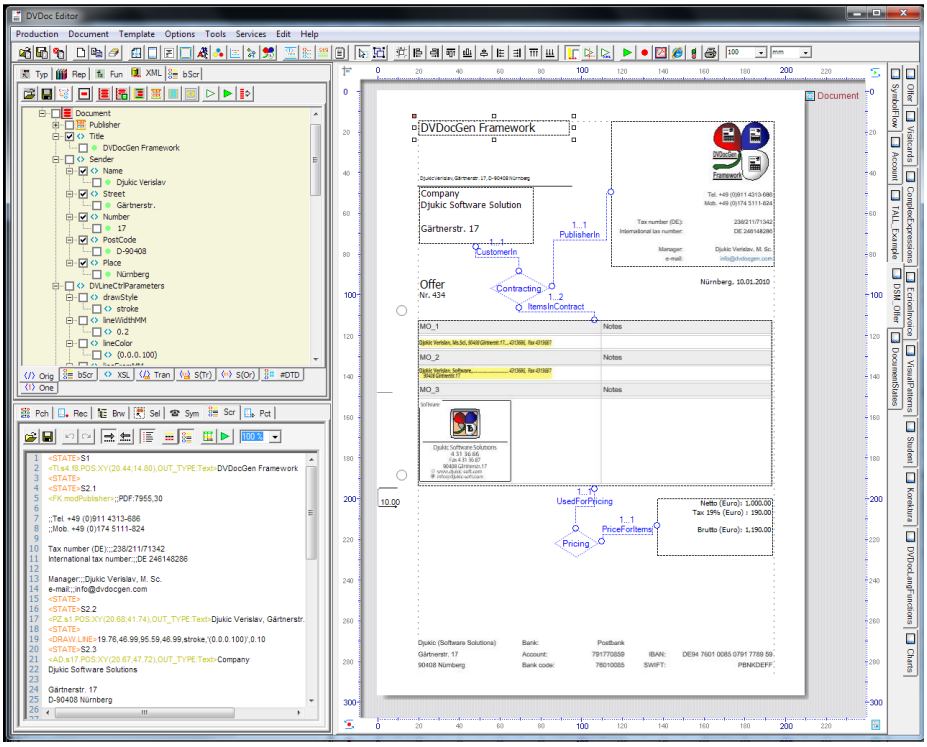

Fig. 3. DVDocIDE for modeling and meta-modeling of document templates

Properties and outputs can belong to different semantic domains. Built-in functions are also supported; mostly those supported by .Net over String, Math, Picture, Date etc. classes. Parts of the output stream can be kept in variables, then manipulated by some functions and then returned to the current stream. Sub-generators can be invoked with parameters. The text editor is equipped with finisher and text highlighter. The main advantage of our language and the tool is that they can, at the same time, to:

- Simply transform some parts of a graph to different target frameworks;
- Transform to different Domain-Specific Languages;
- Transform semantic to topological relations;
- Reference on properties defined in DSL and system properties; and
- Validate templates and transformation by the means of incremental document rendering.

To the best of the knowledge of authors, there are no other tools providing all these functions in a unified manner.

Example 5.1: The 'Structural Patterns' generator is intended for the generation of patterns which are primarily used to describe overall document structure. Pattern contains specification of rules necessary for validation of input data during the document production process. Pattern may be used to create initial document. Recently, several works has been published which use different language grammars for the definition of structural patterns [5,7,9]. We have decided to define a specific language that is simple and expressive enough. The main concepts of the language for the pattern definition are elements, ordering, cardinality and relations. The list of elements can be either ordered or unordered. Cardinalities are denoted as follows:

- * - for allowance of arbitrary number of appearances;
- + - for one or more appearance;
- ? – for mostly one appearance; and
- x…y – for an interval of appearance, from… to.

Set of posible topological relations is fixed, and comprises relation such as: IsLeft, IsBellow, etc. Structural patterns are defined over elements, without considering their layout attributes. The example shows the generator that generates such patterns and then gives an overview of some typical structural patterns.

```
REPORT StructuralPatterns
  foreach {>}
  { 'PATTERN  PAT_' :dsmName; ' OL('
    foreach {~}
    {
      :dsmName; ','
    }   ') '
    :relPos; ' ('
    foreach {~}
    {
      :dsmName; ','
    }   ') END' nl
}
ENDREPORT
```

The first loop traverses all relations and for each of them creates a new pattern. The second and the third go through all roles that are used for connecting objects to relations (e.g. Tables and texts in Fig. 3) and take the names of the roles. Property ':dsmName;' denotes object, relation or role name. The output looks like:

```
PATTERN
 PAT_Contracting
 OL(Customer[1..1],Publisher[1..1],OfferItems[1..2])
 IsLeft(Customer,Publisher)
```

```
END
PATTERN
 PAT_Pricing OL(OfferItems[1..1],Price[1..1])
 IsBellow (OfferItems,Price)
END
```

The name of the first pattern is PAT_Contracting, and it contains an ordered list of data about the customer, publisher and the offer items. All of them are mandatory and for one customer two tables with items are allowed. All elements are in the Rel_Contracting relation. Topological relations are expressed by: IsLeft, IsBellow,...

Some typical patterns are:

PATTERN A OL(B,C,D) END

Interpretation: Element A consists of three elements that can be in the given order.

PATTERN A UL(B,C,D) isLeftOf(C,D) isBelow(D,B) END

Interpretation: Element A consists of three unordered elements, but C has to be left from D and D on the top of B.

PATTERN A UL(B,C[3..5],D) END

Interpretation: B appears only once, C from 3 to 5 times and D exactly one. Elements can be in any order.

PATTERN A OL(B*,OL(C,D)) END

Interpretation: First appears B arbitrary number of times, then C and D.

6 Semantic based graphical template editor

Template editor is a part of the DVDocIDE and it is intended for specification of document layouts and their dynamic characteristics, i.e., dynamic characteristics of business processes that are modeled using such documents. Semantics of the relationships between content units is specified using PIM-s. Every content unit, regarding its complexity, can be associated with an activity that creates such a content unit. Using this relationships of content units and activities, and using the specification of lifecycle that is described in Fig. 1, the DVDocIDE generates transformation formulas that enable incremental specification and incremental rendering of documents [6,15,18]. The logical model of document templates transformed into concrete syntax is called a logical script. A simplified version of such a logical script looks like:

```
<CU.layout_1_ID>Content 1
<STATE>state_1_ID
<CU.layout_i_ID>Content i
<STATE>state_i_ID
<CU.layout_n_ID>Content n
<STATE>state_n_ID
```

State_i_ID is an identifier of the i-th state, which a document reaches if the i-th content and layout are defined. Thanks to such a way of binding elements of layout and content, with dynamic characteristics, the same template can be used for different

production models. Particularly important is the fact that the document is rendered up to the arbitrary i-th state, and that in the non-terminal state, documents can change their layout as well as content and behaviour. If we return to the example in the Fig. 1 this means that during the production we manage diagrams. Videos that demonstrate incremental specification and document rendering are given in [25,26,27,28].

7 Conclusions

Comparing to the tools of the same purpose, one of the main advantages of our tool for specification of templates is that it provides a clear differentiation of meta-modeling and modeling. Meta-modeling is used for defining objects of presentations and their styles, relations, roles and ports. Modeling is used for constructing both document instances and their types. Since DVDocEditor provides modeling objects, relations and semantic domain specifications, it may be also used for logical data modeling. Furthermore, it provides a synchronization with MetaEdit+ Modeler and a software development tool IIS*Case [29]. One of the main benefits of our approach is that we have provided a language, code generator and tool for transforming abstract to concrete document templates. Its practical applicability has been proven in some commercial projects.

8 Related works

Olivier Beaudoux and Arnaud Blouin presented in [23] a framework for graphical components based one MDE technology. Code generating or transformation of abstract to concrete models is limited to stereotyped concepts. Benjamin Klatt in [24] introduced Xpand language for transformation graphics to so called out-domain languages. They discussed their template languages in details. However, they did not present a method of handling document layout, structure and dynamic properties.

9 References

1. Steven Kelly, Juha-Pekka Tolvanen, "Domain-Specific Modeling: Enabling Full Code Generation", ISBN: 978-0-470-03666-2, March 2008, Wiley-IEEE Computer Society Press.
2. Robert J. Glushko, Tim Mc Grath, "Document Engineering", MIT Press 2008.
3. Anneke Klippe, Software Language Engineering: Creating Domain-Specific Languages Using Metamodels, Addison-Weslay 2008, ISBN: 0-321-55345-4
4. MetaEdit+ Modeler, MetaCase, www.metacase.com
5. Di Iorio, A. Pattern-based Segmentation of Digital Documents: Model and Implementation, Ph.D. Thesis, UBLCS-2007-05, Department of Computer Science, University of Bologna, 2007.
6. Verislav Djukic, "DVDocLang Language Reference", Accessed: March, 2012 www.dvdocgen.com/Framework/DVDocLang.pdf

7. Antonina Dattolo, Angelo Di Iorio, Silvia Duca, Antonio A. Feliziani, Fabio Vitali, "Structural patterns for descriptive documents", Proceedings of the 7th international conference on Web engineering, Italy, Lecture Notes In Computer Science, 2007
8. Ivan Lukovic, Verislav Djukic, DVDocLang vs. XSL-FO, www.dvdocgen.com/Framework/DVDocLang_XSL-FO.pdf
9. Angelo Di Iorio, Luca Furini, Fabio Vitali, "Higher-level Layout through Topological Abstraction", ACM DocEng 2008
10. Apache Software Foundation: "FOP", http://xmlgraphics.apache.org/fop/0.95/index.html
11. Microsoft Extensible Application Markup Language (XAML), http://xml.coverpages.org/ms-xaml.html
12. User Interface Markup Language (UIML), http://www.uiml.org/
13. Verislav Djukic, "DVDoc Renderer Benchmak", Accessed: March, 2012 http://www.dvdocgen.com/Framework/DVDocRenderBench.pdf
14. Kosar T., Oliveira N., Mernik M., Pereira M. J. V., Črepinšek M., Cruz D., Henriques P. R., "Comparing General-Purpose and Domain-Specific Languages: An Empirical Study", Computer Science and Information Systems (ComSIS), ISSN: 1820-0214, Vol. 7, No. 2, May 2010, pp 247-264.
15. Verislav Djukic, DVDocGen Framework, application interface, http://www.dvdocgen.com/Framework/DVDocFramework.pdf, Accessed: March, 2012
16. OMG Model Driven Architecture, http://www.omg.org/mda/
17. Exstensible Stylesheet Language, Formatting Objects (XSL-FO), Reference Manual, http://www.w3.org/TR/xsl/.
18. Verislav Djukić, Ivan Luković, Aleksandar Popović, "Domain-Specific Modeling in Document Engineering", Proceedings of the Federated Conference on Computer Science and Information Systems, Poland, 2011
19. Ivan Lukovic, Verislav Djukic, "DVQL Language Specification", www.dvdocgen.com/Framework/DVQL.pdf, Accessed: March, 2012
20. Verislav Djukić, Aleksandar Popović, "DVDocRepLang grammar specification", www.dvdocgen.com/Framework/ DVDocRepLang.pdf, Accessed: March, 2012
21. Colin Atkinson, Thomas Kühne, "The Essence of Multilevel Metamodeling", Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, Springer-Verlag London
22. Eclipse Modeling Framework Project (EMF) , http://www.eclipse.org/modeling/emf/
23. Olivier Beaudoux, Arnaud Blouin, "Using Model Driven Engineering technologies for building authoring applications", Proceedings of ACM Symposium on Document Engineering, 2010
24. Benjamin Klatt, "A Closer Look at the model2text Transformation Language", http://wiki.eclipse.org/Model2Text_using_Xpand_and_QVT_for_Query
25. Verislav Djukić, DVDocRepLang demo, video, Accessed: March, 2012 http://www.dvdocgen.com/Framework/ModelTransformation.wmv
26. Verislav Djukić, DVDocFlowLang demo , video, Accessed: March, 2012 http://www.dvdocgen.com/Framework/DVDocFlow.wmv
27. Verislav Djukić, Using DVDocIDE , video, Accessed: March, 2012 http://www.dvdocgen.com/Framework/UsingDVDocIDE.wmv
28. Verislav Djukić, Using MetaEdit+ from DVDocIDE , video, Accessed: March, 2012 http://www.dvdocgen.com/Framework/DVDocIDEMetaEditCtrl.wmv
29. Ivan Lukovic, Pavle Mogin, Jelena Pavicevic, Sonja Ristic, "An Approach to Developing Complex Database Schemas Using Form Types", Software: Practice and Experience, ISSN: 0038-0644, Vol. 37, No. 15, 2007, pp. 1621-1656.