

## Domain-Specific Modeling in Document Engineering

Verislav Djukić  
Djukić – Software Solutions  
Nürnberg, Germany  
+49 (0)911 4313-686  
info@dvdocgen.com

Ivan Luković  
University of Novi Sad  
Faculty of Technical Sciences  
Novi Sad, Serbia  
+381 (0)21 4852-445  
ivan@uns.ac.rs

Aleksandar Popović  
University of Montenegro  
Faculty of Sciences,  
Podgorica, Montenegro  
aleksandarp@rc.pmf.ac.me

□ **Abstract**— Specification languages play a central role in supporting document engineering. We describe in this paper how domain-specific languages, along with domain-specific frameworks and generators, can support formal specification and document rendering in directory publishing. With flexible metamodel-based tools we have developed four languages for the modeling of: (i) small advertisements, (ii) appropriate documents, (iii) workflow control and (iv) layout patterns. The paper provides a more detailed description of the first and the third language, including a brief account of the language interpreter, as well as code, document and application generators. The presented approach enables, in a typical document-centric system, specification of both static and dynamic characteristics of the system on a high abstraction level with domain-specific concepts. The concepts of incremental document specification and incremental document rendering have been introduced, in order to address the problem of very frequent specification(s) refinements. The expression power of the created languages is demonstrated with a representative examples of document engineering covering document content specification, workflow control and application generation. All of the aforementioned languages are integrated into a single meta-model, under the name of DVDocLang which is, due to its simplicity, highly applicable for user-driven conceptual modeling.

### I. INTRODUCTION

Document engineering (DocEng) represents a scientific discipline which attempts to unify different types of analyses and modeling perspectives, in order to aid various specification, design and document implementation activities and all the processes which, both, create and consume them [2]. Formal document specification and rendering, as a part of document engineering, comprise a research area in which recent years two distinct directions have become prominent: (i) the first, mostly presented in academic work adhere to general approaches and solutions based on XML languages and (ii) the second, emerging due to the need for the production of large amounts of valid documents, is characterized by adherence to the complex layout document

rules in specific business domain. These different directions aim to solve document engineering work in two ways. One side attempts to describe domain-specific problems by the using General Purpose Languages (GPL) which has a rather negative direct consequence creating the need for the development of a multitude of applications, concentrated on either one single problem or on a class of similar problems. Others attempt to develop custom and complete frameworks which would, to the greatest extent, simplify and automate the document production process as well as significantly improve their overall layout quality. This dichotomy, existing between the approaches based on GPLs and those based on domain-specific ones, is not only present in the area of document engineering, but in software engineering in general ([14],[15]). This topic is receiving more and more attention in the academic community, with the methodology tools necessary for solving the problematic aspects being intensively (constantly) developed. In this paper, the authors present their substantial long-term experience in the area of Domain-Specific Modeling (DSM) [1], with the emphasis on the development of the domain-specific framework. The examples chosen for the illustration of DSM in document engineering [2], have been acquired from the directory publishing, and applied to the formal specification and visualization of the documents. The problems being solved are essentially diverse in nature, and are in direct relation to a number of software engineering domains such as: construction of formal languages, Domain-Specific Languages (DSL), conceptual modeling, form-based analysis (FBA) [3], user-driven modeling (UDM), model-driven architecture (MDA) [16], service-oriented architecture (SOA), rendering of the PDF and HTML documents and generation of web applications. The first part of the paper describes the domain and the domain-specific languages (DSL), together with the analysis of the practical benefits of their usage. The second part of the paper provides a brief theoretical overview on incremental specification, and rendering of documents and applications. Accordingly, the paper is divided into seven sections: next Section introduces the domain and provides a substantial example of a small advertisements modeling language DVDocAd. This language is constructed for the purpose of expressing topological and semantic relations between the content units (CU) of small advertisements [9]. Section three describes the workflow control language DVDocFlow as well as the underlying principle for modeling the business activities. Section four

□ A part of the research presented in this paper was supported by Ministry of Education and Science of Republic of Serbia, Grant III-44010, Title: Intelligent Systems for Software Product Development and Business Support based on Models.

describes the notion of incremental specification, which can be regarded as the greatest contribution to DSM in document engineering. Section five describes the usage of the meta-models and the repository to generate applications. Section six provides an explanation of the domain-oriented libraries. We conclude by describing our experiences of the practical value of the presented approach as well as the plans for a further course of action.

## II. MODELING OF SMALL ADVERTISEMENTS

By using meta-concepts, as GOPRR (Graph, Object, Property, Role and Relationship) employed in MetaEdit+ language workbench [4], any domain-specific language can be constructed. We describe here one particular language that is constructed towards the most important characteristics of small advertisements (“small ads” for short). They include the following:

- Small ad represents a collection of semantically based content units (CU), predominantly textual;
- Optionally, small ad contains a picture, i.e. a logo;
- Small ad always contains at least one single name, either the name of the company or the name of the person;
- Elementary-type content units are mostly limited to a name, address, telephone number, E-mail, web address and a logo;
- When displaying content units emphasis is placed on topological relations, primarily on their sequence;
- Telephone number is always placed in the top right corner, in continuation of a name or an address;
- In the course of element formatting, splitting of the text for specific content types is not allowed;
- Alignment can be left, right, centered or combined (applicable to each content unit separately);
- Specific content type can be assigned the leading role in a logical unit, i.e. implicitly defined complex CU;
- Small ad can be comprised of a number of logical units.

The aforementioned characteristics can be described by different languages. In this case, 2D graphics and the following meta-concepts presented in Fig. 1 are employed:

- Object of the type “picture” (logo);
- Object of the type “textual contents” including subtypes: “name”, “location”, “telephone number”, “street address and number”, “E-mail” and “web address” (represented by rectangles containing text within);
- Relations: “content line” (represented by an ellipsis with three dots), “telephone connection” (represented by a telephone symbol) and “content unit” (filled in circle);
- Roles: “is a part of content unit”, “leading in line”, “successor in line”, “telephones in” and “tel. rings in”.
- Properties: ad height and width, logotype height, font and text colour, alignment and leading symbol.

By employing MetaEdit+, within just one hour for middle experience user, a new domain language has been

constructed and an example, presented in the Fig. 1, is modeled. The tool provides graphical editors for the creation of ad instances and checking mechanisms to validation the specifications through given set of language rules. Such an approach, to ad modeling, is inherently different from the process of drawing in general-usage graphic tools. Hence, what is achieved this way is that each advertisement becomes a “pentaformat” document [5]. It means that the document is viewed as a 5D entity: content, structure, layout, meta-data and behavior.

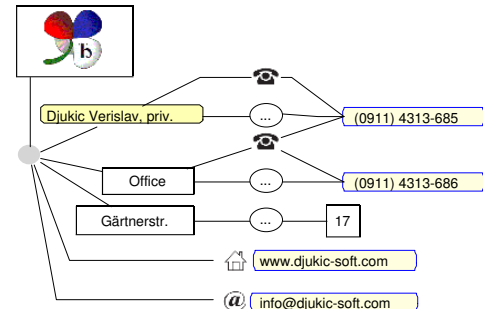


Fig. 1 Model of simple advertisement

The ad from Fig. 1 does not represent the final image a user requires. It is not usual for relations and roles of ad elements to be displayed explicitly. Instead, in the course of rendering, the layout rules are interpreted in the way in which the elements will unambiguously point to the type of content and relation by their position, font, colour and alignment. MetaEdit+ as a modeling tool was not primarily employed here for the purpose of drawing small ads, but to specify domain knowledge in a form of a DSL so as to provide a validation of the created specifications and a generation of special-purpose applications. Ad-production interface can then completely rely on the rules of DVDocAd language. Due to practical reasons, and for the purpose of accelerating the modeling process, a textual language equivalent - DVDocLang [6], has been created. Moreover, the language in question can be integrated into arbitrary framework much easier, and is quite suitable as a basic interface for the end user. Transformation to and from the graphic language is possible. We denote a statement created by DVDocLang syntax, specifying the content, structure, layout and behavior of the document as “logical script”.

**Example 1.** A logical script that specifies content of the advertisement from Fig. 1, is specified as follows:

```
<LOGO>7937,20
<NA>Djukic Verislav, priv.<PH>(0911)4313685
<NA>Office<PH>(0911)4313686
<ST>Gärtnerstr.<HN>17
<EM>info@djukic-soft.com
<IN>www.djukic-soft.com □
```

Knowledge about objects, relations, roles and layout is, due to practical reasons, singled out and regarded as a part of the document type definition. It is comprised of global attributes (elements, logo\_enabled, logical\_fonts, width, height, logo\_params) and content unit attributes (POS,

GROUP, SYMBOL, FORMAT, FONT, LOG\_FONT, ID and ROLE) [6], as demonstrated in the following example.

**Example 2:** A logical script that specifies layout and type definition of the advertisement from Fig. 1:

```
<TEMPLATE>=elements:(NA;Name),(ST;Street),
(HN;HouseNr),(PH;Phone),(IN;Inet),(EM;eMail)
<TEMPLATE>=logical_fonts:(s1;Tahoma,Bold;2;97;2),
(s2;Tahoma;2;97;2)
<TEMPLATE>=logo_enabled:true
<TEMPLATE>=width:40
<TEMPLATE>=height:50
<TEMPLATE>=logo_params:7937,center,20,false
<NA>=POS:new_line,GROUP:true,LOG_FONT:s1
<PH>=POS:con,GROUP:true,FORMAT:'2n|3n',...
<ST>=POS:new_line,GROUP:true
<HN>=POS:con,GROUP:true,LOG_FONT:s1
<IN>=SYMBOL:(SymbolFont;&H29),POS:new_line,
GROUP:false,LOG_FONT:s2
<EM>=SYMBOL:(SymbolFont;&H40),POS:new_line,
GROUP:false,LOG_FONT:s2
```

For the two, grouped (GROUP:true), content-unit types differing in position (POS:new\_line and POS:con) on a level of appropriate instances, establishment of a relation “content line” with the following roles – “line leader” and “adherent in line” is permitted.

This way, the modeling of small ads is reduced to a description of the structure and the content, based on a fairly simple syntax. It is intuitively acceptable, and in agreement with the expectations of the producer of small ads<sup>1</sup>. Fig. 2 shows previously specified ad, generated through DVDocRender. This is a domain-specific document renderer that interprets DVDocLang specifications and produces PDF or HTML specifications, as well as images.



Fig. 2 Small ad generated by DVDocRender

In agreement with the DSM approach, the first step was the construction of small ad modeling language. Subsequent steps encompass the process of making or integration of domain-specific libraries, used for interpreting and rendering of advertisement documents. Grammar rules and templates are generated with the assistance of MERL [4], the reporting language of MetaEdit+. This report serves as an entry point for automatic parser generation. Such instance and type specifications of small ads are sufficient for generating other specification varieties, e.g. in SGML, XSL-FO, HTML, DVDocLang and other formats. Furthermore, such a description of relations between objects is sufficient for

pattern construction which is employed for advertisement structure validation, i.e. validation of documents [7]. Capabilities of the document generator were discussed in detail in the comparative analysis of the concepts and characteristics of XSL-FO and DVDocLang provided in [8]. In the case of the ad from Fig. 2, our document generator is even more than forty times faster [10] than the FOP renderer [13] accepting XSL-FO specifications as the input.

### III. WORKFLOW CONTROL LANGUAGE






The fact is that a number of renowned workflow control systems, such as Bonita, JWT, Alfresco et.al. exist. Still, they do not provide a solution to the problem of parallel refinement of the document layout and business process models. Incremental specification by means of a simple DSL unifying both activities, together with the incremental rendering, is seen as a prerequisite for the building of the software system in which the progress in each of the activities is documented by a valid document instance. This is the exact reason why we have constructed a new language, by employing MetaEdit+. We describe in this section DVDocFlow – a specific language used for workflow control as well as for synchronization of parallel activities. It is employed for the description of the dynamic characteristics of the system, i.e. document behavior. This represents one of the dimensions of the previously mentioned “pentaformat”. Typically the production of small ads is not insular, but is a part of more complex activities such as sending of offers, error correction in ads and conclusion of the advertising contracts. The analysis of the documents, in which small ads represent certain content units, pointed out to two important facts: (i) the state of an ad affects the state of the document it is a part of and (ii) collection of potential states of the documents depends on the overall state of all content units, and transitional rules. This collection of states and transitions, which we refer to as the document life cycle, varies depending on the concrete production model. In addition, it is indirectly determined by the degree of automation of the production process. Accordingly, our main goals in the course of constructing a new language is that it should provide a solid foundation for the description of different production models as well as that the documents which did not yet reach their end state can “respond” to the change of a production model. It is not necessary for a concrete system to possess a workflow engine (WFE). Knowledge of the life cycle, built in each document instance, can completely replace the WFE. Incremental specification and rendering are viewed as a domain-specific solution, used for realization of the specification and tracing of the document behavior. It is presented in Section 5 in more detail.

In Fig. 3 we present an example of an offer-production model, specified in DVDocFlow. In the course of language creation process for DVDocFlow the same meta-concepts






<sup>1</sup> Successfully applied in production of ads in ten countries.

(GOPRR) have been used, as in the case of DVDocAd. The DVDocFlow meta-concepts and symbols are:

Objects:

- Document states: 
- Activities: 
- Content units (Complex, Logo): 
- Template: 
- Document generator: 

Relations and included roles:

- Increment of a state:  ("Sets into a specific state", "Comes with a content unit");
- Layout definition:  ("Preferred for document", "Document contains content unit")
- Pattern based:  ("Template uses a pattern", "Pattern for document templates");
- Activity states:  ("State from an activity", "Sends to state", "Visual repres. of progress in PDF format"); and
- Synchronization action:  ("Waits an activity to

end", "Request accepted").

The overall number of concepts and properties available in DVDocFlow is much greater, but we explained here just those ones necessary to introduce our production model in brief. Aside from the explicitly described concepts, for modeling and rendering documents the following is also important:

- Content units, their layout included, are linked to a specific role in a real system;
- In each state, except for the final, a specification increment can be joined to a document instance. It can alter the life cycle independently from the previously defined one, for the type instance is a part of;
- All the changes, in a real system, are documented by concrete document instances in PDF, that can be generated for any state of activity;
- From the specification of dynamic characteristics, acquired from modeling tools, a code, which manages document transitions, is generated;
- Relying on modeling infrastructure (repository, editors, code generation language and API) application prototypes are generated;

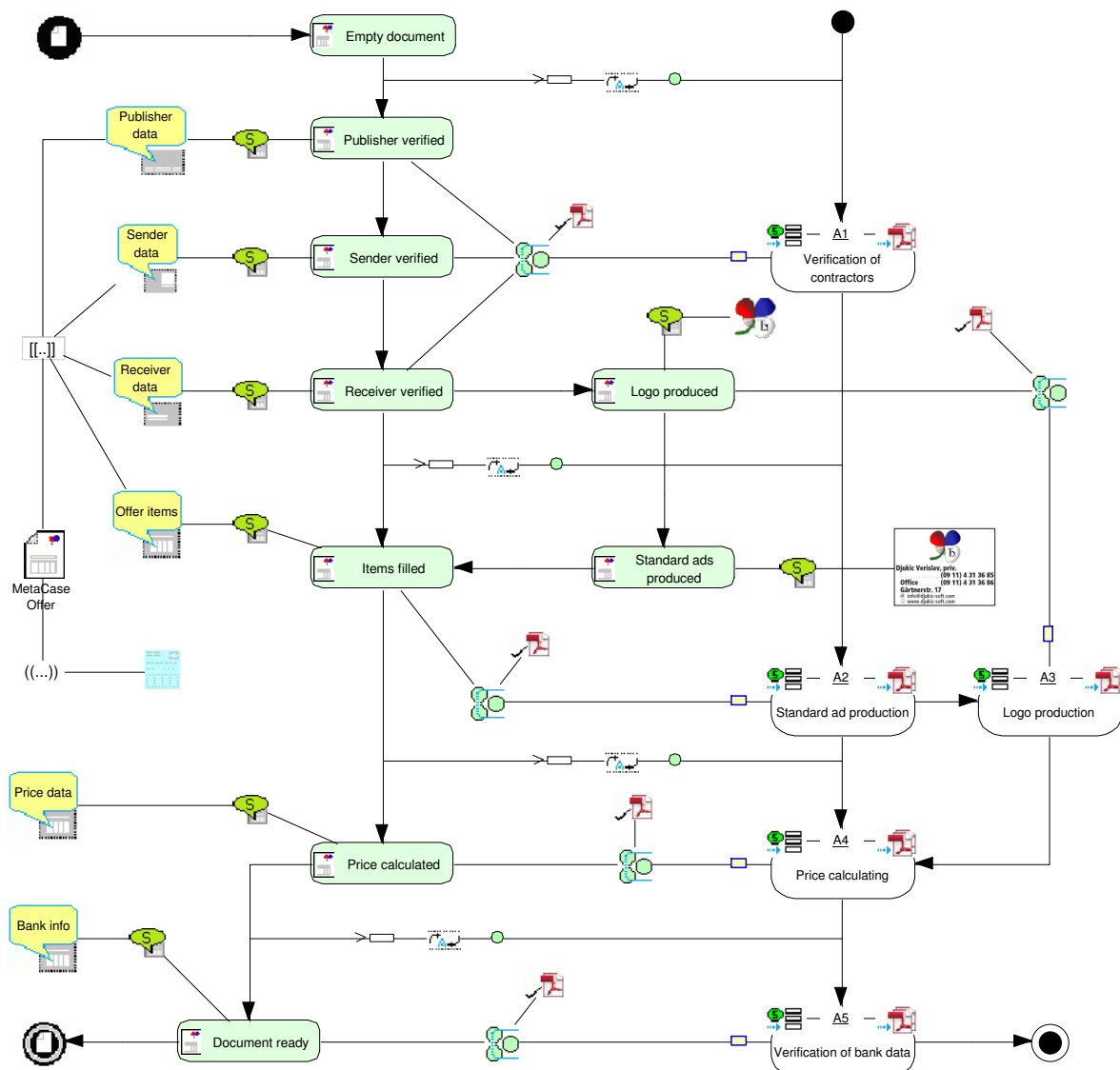


Fig. 3 Production model for "Offer"

- There exists a domain-specific editor that provides specification of document types by drawing typical instances and using available patterns for different types of content units;
- UML activity and state diagrams are suitable for specification of the document activities and states. Thus, in DVDocFlow language, similar symbols are used for the activities, states and transitions; and
- The main purpose of DVDocFlow language is to provide modeling relations between the states of the documents, content units, layout forms and activities in which the appropriate content units are created.

Formal specification of a production model in DVDocLang, expressed by the concepts which integrate activities, states, layouts and behavior of the documents, is an extension of incrementally oriented logical script, as shown in Examples 1 and 2.

**Example 3:** A basic form of logical script for expressing production model of document type “Offer”:

```

<STATE>Empty document
<CU>Script from validation of sender
<STATE>Sender verified
<CU>Script from validation of receiver
<STATE>Receiver verified
<CU>Script from validation of publisher
<STATE>Publisher verified
<CU>Script from logo production
<STATE>Logo produced
<CU>Script for simple ads,i.e. (1) and (2)
<STATE>Standard ads produced
<STATE>Items filled
<CU>Script from price calculating
<STATE>Price calculated
<CU>Script from verification of bank data
<STATE>Document ready □
    
```

Each element of a logical script, marked with tag <CU>, is either a simple value or a complex content unit. In the case of a complex content unit, it is related to a number of different document states. Recipient, publisher and sender data, displayed in Fig. 3, can be represented by one composite content unit. Commands, in the form of <STATE>Name, are a part of a logical script, related content units to the states of a document.

Further characteristics of such conceptualized workflow control language include the following:

- General approach in the specification of a document layout represents one simple case in which all of the <CU> are known prior to the beginning of rendering;
- The content of the expression with <CU> is instance-related. The definition of content unit type, which includes the layout properties, can precede an instance and has the form <CU>=*Definition of CU\_type*. This implies that the layout can be redefined while it is in a non-terminal state, depending on a certain activity;
- Expressions - in the form of <CU>=*Definition of CU\_type*, also define a collection of new and potential states

since they alter the layout, regardless of the fact that the content remains unchanged;

- Definition of the content unit type is rich enough to enable PDF and HTML generation, as well as that of a web application;
- Work progress is documented by a valid, legible content unit or document instance;
- In case meta-data is placed in the form of annotations in a PDF or meta-properties in HTML, then each instance also describes completely the type it belongs to, and can, thus, be cloned and inherited. Viewed from the standpoint of optimal refinement of layout-definition, cloning and inheriting are highly important; and
- Separation of the content from a layout definition enables for a specific component to acquire layout definition instances, for the purpose of their merging in the course of rendering process.

Fig. 4 shows a layout of a document's instance, in four different states. From the standpoint of the user who produces/creates documents, this specific manner of business process progress reporting is, by far, most acceptable. State represented by (1) refers to the confirmed contractor data, (2) readymade small ads and filled in list, (3) calculated cost and (4) verified banking data. Presented example refers to a rather simplified case – when the layout of content units displayed, remains unchanged regardless of the states.

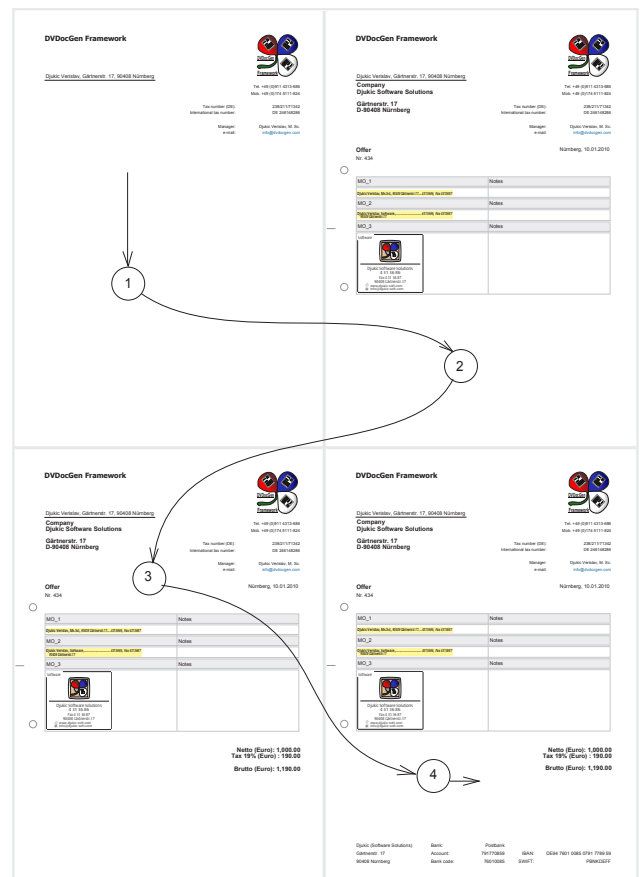


Fig. 4 Document rendering for each of the different states

#### IV. INCREMENTAL SPECIFICATION AND RENDERING

The emphasis in the workflow control language was placed on the procedure that enables the synchronization of activities in a real system as well as on reporting work-progress. In document engineering we also need to specify document production models, rendering processes as well as their complete implementation based on an incremental approach. In the broadest sense, incremental specification and rendering are regarded as document engineering approaches, allowing content, structure, layout and behavior related changes (or additions) of a document being in an arbitrary state. Aside from substantial improvements regarding the rendering speed, our approach significantly simplifies the entire document production process and application generation necessary for the automation of production. By means of an increment specification, whose primary source is the activity producing the content unit, the approach makes the controlled and automated document-knowledge refinement possible. Such a refinement simplifies the production of untypical document instances. Those are the instances somehow differing from the previously defined type or that cannot be created by the existing applications. The notion of a "modifier" denotes a knowledge increment. It can be named, unnamed or unresolvable. A named modifier consists of a group of attributes recognized in advance as a possible type variation. It is unnamed if it does not belong to any previously defined variation. It is unresolvable if there is no any language concept suitable for a full specification of the document instance.

The core elements of the proposed incremental specification, illustrated in Fig. 5, are the following:

- Each content unit or each combination of content units matches at least one specific document state. Fig. 5 displays the potential document states (Initial, S1, S2, Final\_1, Final\_2) for a document „D”. Marked with D1-D5 are visual representations of the documents, in specific states. In an initial state a document is empty.
- At least one layout specification is associated to each content unit, as well as a set of known layout variations (a visual pattern with its variations). The visual pattern is a pattern necessary for the representation of a particular type of the content unit. It is referred from the logical script and easily customized (contextualized) by referring to its variations. The list of content units is placed separately, in the upper right corner of Fig. 5. The content units are marked with CU1-CU4.
- The increment named as SpcIncr in Fig. 5, as well as a complete document, is specified by the domain-specific language DVDocLang. It is possible to define an increment in advance, so the document would ‘carry’ it from the previous state, while attempting to change to the next state. The role of any activity would then be just to accept and return the increment, but not to create it.

- An empty document corresponds to the initial state. End-state candidates are all those in which a document contains a single or a group of content units which are a result of a specific, self-contained real system activity.
- It is possible to define a type and document instance constraints, referring to a structure, contents and layout in each of the states, as well as interpretation specific constraints for any output format (PDF, HTML).
- When, at the time of generation, a document requires change of a state, the generator informs the environment and waits for the continuation signal (WaitForCU).
- Each subsequent state can be defined by an increment specification with additions to the content, structure and layout. The increment is presented in Fig. 5 as a parameter „spcIncr”, in a function call for document rendering: ContDoc(docID,lastState, spcIncr).

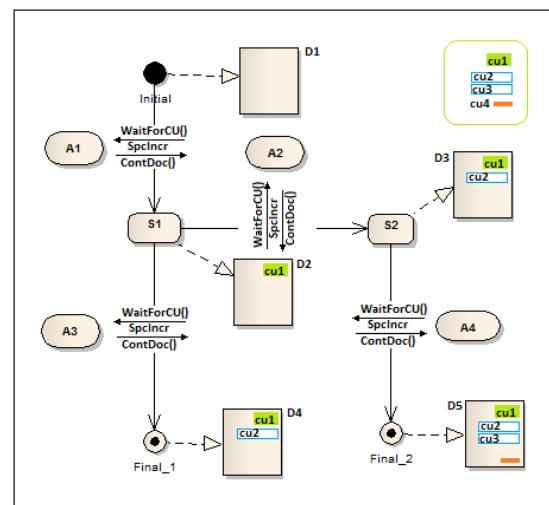


Fig. 5 Incremental specification and rendering

The core elements of the proposed incremental document rendering are the following:

- It is possible to generate a document in PDF, HTML, PS or other specific format in each state. Additionally, it is also possible to continue the process of generation by starting from a previous state. An entry point to continue the generation process is previously generated PDF or HTML document, containing the meta-data (specification of a document class it is a part of, content and the last-state identifier) and optionally, the increment of the logical script.
- Base specifications of content units are stored on a template server, a component which is a part of the document generator infrastructure.
- In each output format, a document contains meta-data, while the infrastructure services can be called by an interchange of XML packages over the known scheme, or by the interchange of documents in PDF or HTML format.

- In a simplified case, when document states are not of any importance, rendering is reduced to a change from the initial into an end state, without the synchronizations of real system activities.

## V. GENERATION OF APPLICATIONS

In the incremental approach, a document is observed as a collection of elementary units – grouped together according to particular rules. DVDocLang is a language for linear textual representation of such a collection. The rules, discussed previously, are, to the greatest extent, dependent on concrete topological relations of CU types valid for a specific document, as well as on an semantic domain of attribute values. If semantic domains and the rules for composing the structure are set, it is sufficient to connect particular content types to adequate visual patterns (Fig. 3). The MetaEdit+ tool, storing the life cycle of an “Offer”, allows us to automatically generate and test an application employed for the purpose of “Offer” production. Instead of the logical script from the Example (3),  $\langle CU \rangle$ Script from... a script, in the form of  $\langle CU.edit \rangle$ Script from..., is generated. The difference is in a modifier “edit”, which is interpreted as an application generation command aimed at editing the contents of the current CU. Formal DSL specification and MERL reports allow us to create a representative pattern collection, as well as grammatical rules necessary for their validation. The patterns in question simplify the creation of new documents and enable validation and altering of the existing document structure. For the generation of applications the following criteria have been met:

- For applications, to be driven by a business-process model, described basically at the moment of specification of the production model and document layout;
- For a HTML and PDF document’s visual interpretations, to be as close as possible to one another, or to, even, be completely indistinguishable;
- For a simple-syntax structured text (especially in a section used to describe the structure and the content of a document), to be acceptable for a user on an intuitive level;
- For the properties describing the content-unit layout of a document, to be automatically translated into the properties of adequate screen forms (i.e. controls);
- To exist a common algorithm for transformations of an application from a particular state into an appropriate document, and vice versa;
- For each document instance, in each state, to provide a fast generation of an application used to move a document into the next state.

Once a document, in a particular state of the life cycle or in the course of rendering, needs to go into the next state, the

document generator sends to the environment the action-related signal. This way the activity manager present in a real system is informed that a particular action should be performed, i.e. that a new content unit ought to be created. Fig. 6 shows one of the possible signal processing case scenarios which requests the change of a document from the S1 state into the S2 state. In case an application for the A2 activity already exists, its signal, announcing continuation of the rendering process, is waited for, with the content increment being assumed. However, if the application does not exist, it can be generated on the basis of the script increment given in the course of document type specification, to which an instance belongs to.

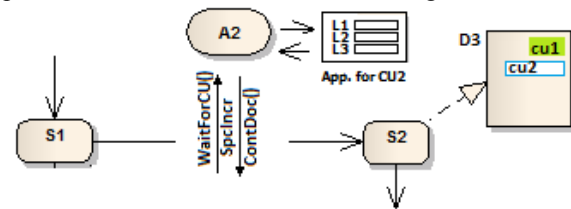


Fig. 6 Generating of application on demand

The most complex case is the one of generating production applications for such CU whose content, structure and layout were not known in advance. Owing to the DVDocLang, which is semantically rich enough, generating such web and .Net applications is possible. Web applications are implemented by way of a collection of HTML documents, which in meta-data contains: (i) a logical script, (ii) optional template definition and (iii) Java Script function which translate the current state of the interface into a logical script, and return it upon the web-service call. On a server side, combining of the initial form script with the current interface state takes place producing the current instance script.

Except for web applications, specifications of production model and layout are sufficient for incremental generation of valid application prototypes in different programming languages, such as Java and C#. Particularly important for document engineering in practice are XAML [11] and UIML [12] languages as they describe in a platform-independent way the layout and functionality of screen forms. As far as the .Net platform is concerned, priority has been given to the development of the collection of user controls, the properties of which are set dynamically – using MetaEdit+ repository and API. The next section describes such a component as well as its ‘behavior’ in a particular case.

## VI. DOMAIN ORIENTED LIBRARIES

The collection of domain-specific languages is primarily constructed with the intention to enable the mapping of domain-specific problems onto specific language concepts aspiring on a higher abstraction. Governed by the principle “what you see is what you get” (WYSIWYG), two libraries of controls have been created. The first is intended for applications in which rapid document modeling by means of

structured text is required, driven by formal DSL specification. The second is designated for template designing by means of drawing typical examples. In some cases, three types of representations are combined for the same language concept: structured text, screen forms and 2D graphic. Fig. 7 displays such combination of representations. Based on the defined data types, location, relations and anticipated size of the content unit, the space is divided into rectangles (areas). To each of the rectangles an editor is assigned, with the user being directed to the area where the data should be entered. For the 'Offer' illustrated in Fig. 4, a default layout has been displayed in Fig. 7, left. Located right is the editor, consisting of forms, i.e. of a collection of ordered triplets (CU type, modifier, value).

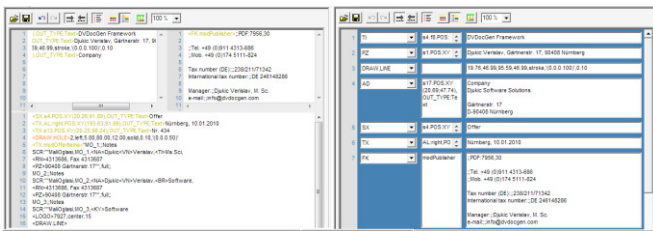


Fig. 7 Editing by structured text and screen forms

The generation of applications for document instance production is reduced to the use of the MetaEdit+ repository, for the purpose of assuming the appropriate control properties. Property values are described as MERL [4] reports, i.e. as queries on a repository. Property examples include: the list of allowed content types, modifiers allowed for a specific content type, collection of special symbols etc.

## VII. CONCLUSION

DSM can be applied to support document engineering. We have presented in more detail two domain-specific languages: DVDocAd that is employed for the modeling of small ads and DVDocFlow – that is employed for the modeling of business activities in relation to the documents and their content units. Both of the languages have been defined and implemented by means of MetaEdit+ tool. As a result, formal document specification in the domain of directory publishing is simplified with DVDocLang as it uses higher-level modeling concepts that are semantically close with the document formats. It also facilitates effective user-driven modeling. The issue of 'atypical' instances has been solved by applying the incremental specification as well as by document and application generation. With the developed approach we have got up to forty times faster rendering in the case of small ads [13] in comparison to a FOP generator. In the case of composite documents, including the concepts which almost entirely account for XSL-FO [17], the speed is ten times greater. One framework is created, which makes workflow reporting possible at any given time. By directly relating content units to activities that form them we are able to control the document related workflow. A document is

treated as a 5D entity (pentaformat). If the document is reduced to three dimensions (content, structure and layout) automatic application generation, inheritance and cloning become impossible.

We have presented in this paper an overview of the developed languages and related tools. They enable automation of document engineering in directory publishing. We believe that various areas of document engineering may also be supported by our approach. In particular two most significant areas that we are concentrating on at the moment are semantically based 2D graphic editors for template drawing (DVDoc Editor) and query language for document browsing (DVQL).

## VIII. ACKNOWLEDGEMENT

The authors would like to kindly thank Juha-Pekka Tolvanen from the University of Jyväskylä for his valuable support and proof reading.

## 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] Dirk Draheim, Gerald Weber, „Form-Oriented Analysis“, Springer-Verlag 2005, ISBN 3-540-20593-4.
- [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", 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 Benchmark", <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] Mernik M., Heering J., Sloane M. A., When and How to Develop Domain-Specific Languages, ACM Computing Surveys (CSUR), Association for Computing Machinery, USA, Vol. 37, No. 4, 316-344. 2005
- [16] OMG Model Driven Architecture, <http://www.omg.org/mda/>
- [17] Extensible Stylesheet Language, Formatting Objects (XSL-FO), Reference Manual, <http://www.w3.org/TR/xsl/>.