Evolvable Documents – an Initial Conceptualization

Marek Suchánek and Robert Pergl

Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
Email: marek.suchanek,robert.pergl@fit.cvut.cz

***Abstract*—We may say that documents are one of the cornerstones of our civilization. Information technologies enabled unparalleled flexibility and power for retrieving, storing, and sharing documents. However, in a daily documents-intensive job, one needs to deal with severe complications of documents evolvability and reusability of their parts. Maintaining consistency across several documents and their versions is typically a tedious and error-prone task. Similar evolvability challenges have been dealt with in software engineering and principles such as modularity, loose coupling, and separation of concerns have been studied and applied. There is a hypothesis that they may help in the domain of evolvable documents, as well. We perceive devising a conceptualization of documents as the first step in this endeavor. In this paper, we present a generic conceptualization leading to evolvable documents applicable in any documentation domain, and we propose next steps.**

Keywords–Electronic documents; Evolvability; Modularity; Conceptualization; Separation of Concerns.

I. Introduction

Documents are a vital carrier for storing and distributing knowledge - the precious result of various human activities. The amount of documents grows rapidly primarily thanks to their "cheapness" in the digital era. However, an interesting observation may be made: In spite of various means of storing, retrieving, and sharing documents in electronic forms, the foundations did not change, and the documents are the same hard-to-maintain and evolve structures as they always were. Imagine for example a document capturing regulations of a study program enrollment. Such a document is issued and maintained by the Dean of a faculty. However, it must be compliant with the university's regulations document, which in turn must be compliant with the regulations of the Ministry of Education. We have three levels of documents where the more specific ones contain parts of the more general ones, take them as-is or elaborate more specific versions, add further regulations, etc. Now, imagine that there is a change in the Ministry's regulations, which must be appropriately dealt with in the referring documents. This situation affects at least tens of Faculty's agendas which results in inefficiency, inconsistency, and other related problems.

The first observation is that documents are seen as monolithic wholes or wholes composed of highly coupled parts which cannot be separated or even reused. If we would be able to decouple parts of documents, make them loosely coupled just by higher concerns, and design them as reusable, it would significantly help in many domains, such as teaching materials, corporate documents, manuals, regulations, etc. The practice of software engineering suggests that if done properly, evolvability may be significantly improved, the efficiency of document management gained, and error rate decreased [1].

In Section II, we first briefly introduce a wide variety of related work affecting documents domain in terms of the modularity and evolvability. Section III is divided into three steps of our approach to creating a generic conceptualization, i.e., independent on a type of enterprise or domain involved. We apply concepts from theories used in computer science and software engineering verified by practice. Furthermore, we build our approach on the Normalized Systems (NS) theory [1], which is dealing with evolvability of information systems and it has been reported to be successfully applied in other domains than software development including documents [2]-[4]. In Subsection III-A, we split the domain into key parts and then, in Subsection III-B, we introduce conceptual models for them using the ontologically well-founded conceptual modeling language OntoUML [5]. After this exploratory and inductive part, Subsection III-C contains deduced possible and potentially suitable next steps and future work.

II. Related Work

Over the years of Information and Communication Technologies (ICT) field development, many solutions for working with documents and documentation emerged [6]. In this section, we briefly discuss some key areas and approaches related to electronic documents. This review of the current state-of-the-art provides a foundation for our conceptualization of the documents problem domain in general.

Nowadays, there are many different text processing tools, syntaxes and complex systems for dealing with documents within their whole life-cycle [6]. The goal of this part is not to describe particular existing solutions, but to briefly emphasize essential and interesting approaches or ideas that should be considered before developing new solutions. All of the mentioned approaches strife to make dealing with documents simpler and more effective.

A. Formats and Syntax

There is a plethora of markup languages and ways how to encode a document providing different advantages, some are focused to be easily readable in plain text, and others provide ways to encode complex document elements [6]. An interesting concept of versatility and evolvability is represented by the Pillar markup language for the Pharo environment [7]. It consists of a document model which is easily extensible by implementing new classes and visitors defining syntactic constructs meaning and handling. Furthermore, the provided tool allows export in many other formats and markups.

 ISBN: 978-1-61208-612-5

Converting between formats is also important to mention. A great example of a markup converter is Pandoc, which enables to convert from over 20 formats to more than 30 formats [8]. The lower number of input formats illustrates the fact that some of them are harder to process. At the same time, an output format of a document should be expressive and extensible. For example, LaTeX has mechanisms of custom packages and commands, environments, and macros. It is then a considerable challenge to convert it to another format lacking these extensions [9].

B. Templates and Styles

Separation of a graphical design and content is the first notion of separation of concerns in documents. A document, or any piece of data in general, can be rendered using an independent template associated with one or various styles. This approach can be seen in many documentation systems and languages, such as Extensible Markup Language (XML), HyperText Markup Language (HTML) and Cascading Style Sheets (CSS), LaTeX or even in various *What you see is what you get* (WYSIWYG) Office suites. This separation leads to good evolvability of document structure and style without touching the content itself. [6][9]

Using templates with styles to easily form and design complex structures is well observable in the field of web development. Many web frameworks are supplied with one of many template engines, namely, Twig, Jinja, JavaServer Pages, Mustache, or other. Template engine takes structured data and a template as input and then produces a rendered document, e.g., query result in the form of HTML document with table or JavaScript Object Notation (JSON) array based on the request. Moreover, it is usually possible to extend and compose templates together, and to create reusable components and macros. [10]

C. Sharing and Collaboration

Documents are often written by more than one person. Collaboration possibilities are related to the format used. If the document files are in plain text, then one of the solutions is to use Git or other version control system (VCS) [11]. There are also many cloud services allowing users to create and edit documents collaboratively, for instance, Google Documents, Overleaf, or Microsoft Office Online.

When mentioning Git and other VCSs, it is important to emphasize that it already provides a lot of functions that a powerful document system needs [11][12]. Such features are among others:

- tracking of history and comparing changes of version,
- tagging a specific version,
- signing and verifying changes,
- looking up who changed a particular line of text,
- working with multiple sources/targets and linking other projects submodules,
- logging and advanced textual or binary search within the changes,
- allowing changes in multiple branches,
- merging or combining changes.

Moreover, services like GitLab, GitHub, or BitBucket provide more collaborative tools for issues, change reviews, project management, other services integrations. One of important related services type is continuous integration (CI), which allows to build, check, and distribute results seamlessly. It can be used for example to compile the LaTeX document and send the Portable Document Format (PDF) to a file server or email address. [12][13]

D. Document Management Systems and Wikis

A document management system (DMS), as explained in [6] and [14], is an information system that is able to manage and store documents. Most of them are capable of keeping a record of the various versions created and modified by different users. The term has some overlap with the notion of content management systems. It is often viewed as a component of enterprise content management (ECM) systems and related to digital asset management, document imaging, workflow systems and records management systems.

One of the leading current DMS is an open source system Alfresco that provides functionality, such as storing, backing up, archiving, but also ISO standardization, workflows, advanced searching, signatures and many others [15]. From our perspective, the problem is that DMSs are mainly focused just on working with a document as a whole – documents stored as files with rich metadata, which doesn't contribute to an evolvability itself.

Knowledge can be gathered, formatted, and maintained in a wiki – a website allowing users collaboratively modify content and structure directly from the web browser [16]. Wikis are extensible and simple-to-use sets of pages that can be edited via a WYSIWYG field or manually with some simple or custom syntax, e.g., Markdown, reStructuredText, or DokuWiki. The system keeps track of changes within pages as well as the attachments, so it enables to compare differences and see who and when changed the document. Common extensions of wikis are tools for exporting to various formats or extending syntax and other user-friendly functionality [17]. There are many diverse commercial and open-source solutions with slightly different functionality. Commercial solutions are often called enterprise content management and consist of a wiki system and a DMS to manage documents in a better way than just with a plain DMS [14].

E. The Normalized Systems Theory

The Normalized Systems theory [1] deals with modularity and evolvability of systems and information systems specifically. It introduces four principles in order to identify and eliminate combinatorial effects (i.e., dependencies that are increasing with the system size): Separation of Concerns, Data Version Transparency, Action Version Transparency, and Separation of States. Applying the principles leads to evolvable systems composed of fine-grained and reusable modules. In the documents domain, only the first two principles are applicable [4], because actions and states are workflow-related.

The principles and concepts of the theory have been reported to be used in other domains, such as study programs [2] and documents [3]. In the paper [4], it is shown in a form of the prototype, how the theory can be used (especially the separation of concerns and creating modular structures) in the domain of documents for study programs. The prototype is able to combine selected fine-grained independent modules and to generate a resulting LaTeX document.

 ISBN: 978-1-61208-612-5

F. Source Code Documentation

Basically, for every widely-used programming language, there are one or more systems for building a documentation from annotations and comments that are placed directly in a source code. Such systems are, e.g., Javadoc for Java, Doxygen for C/C++, Sphinx for Python (see Listing 1), or Haddock for Haskell.

The fundamental idea is to place parts of documentation directly into a documented artifact (a variable, a function, a class, a module, a source file, etc.). The resulting documentation is as modular and evolvable as the writer creates it according to guidelines and with *Don't Repeat Yourself* (DRY) principle. It is then indeed easy to edit just a part of documentation related to one concern if the concern is separated in the source code. Another observation is that such documentation is composed of reusable parts. Linking the source file to different project results in its inclusion to a documentation of a different project, too. [18]

Listing 1. Documentation of Python source code

```
class Person:
    """This is simple example Person class

    You can create new person like this:

    .. code::

       bd = datetime.datetime(1902, 1, 1)
       p = Person("Peter Pan", bd)

    :ivar name: Full name of the person
    :vartype name: str
    :ivar birthdate: Birthdate of the person
    :vartype birthdate: datetime
    """

    #: Number of people instantiated
    people = 0
    ...

    @property
    def age(self):
        """Age of the person (birthdate-based)"""
        t = date.today()
        b = self.birthdate
        return self._age_diff(t, b)
```

III. Our Approach

Our approach to investigate and understand the problem domain of evolvable documents is to split it into *four separate key areas* and to build conceptual models of the domain in ontologically rich language OntoUML based on them. Next, we suggest possible solutions which can be based on them and could lead to improvement of documents evolvability.

A. Key Document Viewpoints

After the brief overview of current approaches in the ICT support for documents, this section introduces various key viewpoints that are not ICT-related but are typical for documents in any form. Each of them is shortly described, and a possible implication in the computer science domain follows. The viewpoints are defined with respect to the semiotic ladder that introduces several steps from social world to physical world: pragmatics, semantics, syntactics, and empirics [19].

Pragmatics and semantics, that are related to the meaning and intentions, are covered within first three subsections. Syntactics is related to the last subsection called Structure. Encoding the document in the physical world, as other parts of empirics, is out of the this work's scope.

1) Meaning: Apparently, the meaning is the key part of a document, as the purpose of the document is to store and carry a piece of information that can be retrieved in the future [20]. As the triangle of reference says, the meaning is encoded in symbols of some language via concepts [21]. The common problem is that in case of documents, the language is a natural language. Because of that, documents are hard to be understood by computers effectively in the sense of their true meaning. Advanced methods in data mining and text processing disciplines are trying to address this [22]; however sometimes the meaning is hard to be decoded even by human beings themselves...

Meaning, purpose, concern and other content information may be provided as metadata of the document. Considering such metadata, there should be a simple, single and flexible model for such description of documents for an easy automated processing. [23][24]

If a meaning of a text is captured in a machine-readable way, then it is possible to extract desired information, compare the meaning of different documents, find logical dependencies, and many others with an automated processing. The most basic form of captured meaning are *triplets* that consist of subject, predicate, and object [24]. Such an assertion is very simple but powerful. For specific languages, it is possible to derive them more easily than from the others (e.g., English with its stable sentence structure vs. Slavic languages); text mining may also be used for derivation [22]. The assertions can naturally have relations between themselves and form a swarm of assertions, which is helpful for comparing different sources of information. The information storing based on triplets is typical especially for bioinformatics.

2) Concerns: Writing a document happens with concern in mind, and typically there are multiple concerns across a document. We can understand a concern in a document as a principle that binds sentences in a paragraph, paragraphs in a section, and sections in a document together. The whole document then speaks about the highest-level concern that is then split into parts recursively, until we reach some atomic level such as paragraphs containing a set of statements. Lower-level concerns can act as a separator of document modules, and higher level concerns are then composed by multiple submodules. It indicates that splitting the concerns further is not intended by the author.

For example, considering manual for a product, the top-level concern is about the product in general with sub-concerns installation, usage, license, and warranty. The usage can be then again split into concerns related to usage of specific parts of the product. On the other hand, the warranty might not have any further sub-concerns.

3) Variants: Apart from the primary concerns in a document, there are also cross-cutting concerns that are not related to meaning and information inside a document, but rather to its usage. Such cross-cutting concerns are an intended audience, a language, a form of document (slides, handout, book, etc.), and so on. Those represent variants of a single document. They are a source of possible combinatorial effects.

 ISBN: 978-1-61208-612-5

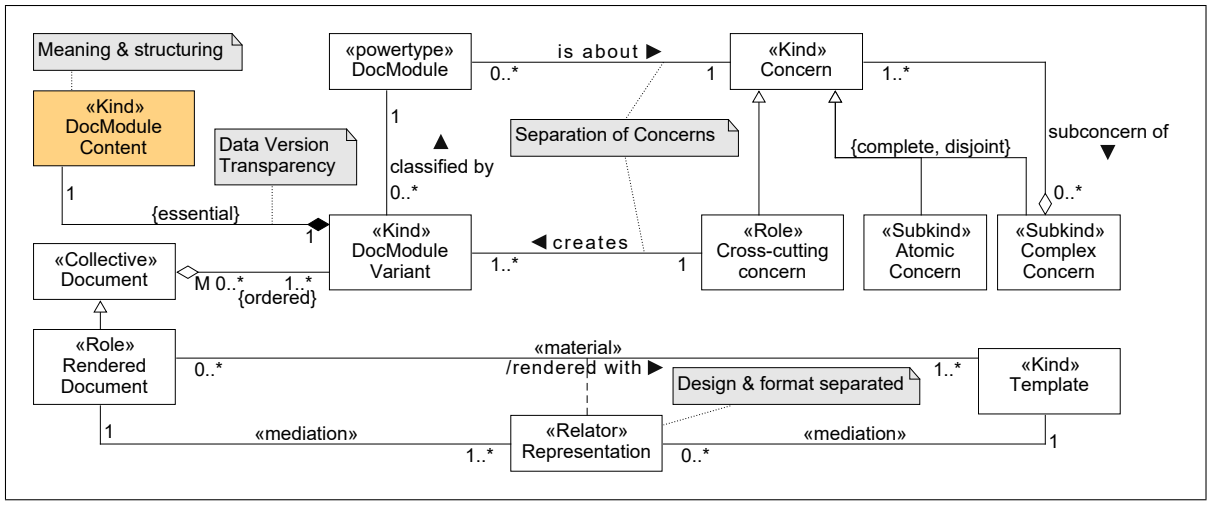

Figure 1. Conceptualization of concern-based document modularization in the OntoUML ontologically well-founded language.

For example, teaching a course requires a textbook and lecture slides which are, of course, very closely related. When you do some update in the textbook, you need to update the affected slides. Now, imagine teaching the course in two languages and some classes for seniors and some for juniors. So, you have 8 different variants and adding one more language would lead to another 4. Apparently, it is becoming hard to manage these separate documents correctly. This is the core challenge, where combinatorial effect-free documents should help.

4) Structure: A structure of a document is essentially a hierarchy of the document composition: chapters, sections, subsections in various levels, paragraphs, and parts of paragraphs. Then there are also other *block elements*, such as lists, tables, figures, code examples, equations and similar. Next, we distinguish so-called *inline elements*, which are parts of text inside a block to capture the different meaning of words (e.g., a link, important, math, a quote, a superscript, etc.) or to provide additional information, for example, a reference or a footnote. Notice that we don't state anything about the style here.

The flexibility of a document structure is an enabler of evolvability. Aligned with the notion of modules in programming, every modular unit should be loosely coupled with remaining parts and allow to be moved to a different place even in a different document. A heading level represents a typical problem: there is a level of the unit involved, and it gets more complicated with cross-references. It goes even deeper when we consider that its position in a document may form a list of prerequisites that the reader should know beforehand.

Finally, we would expect a possibility to define a new custom element, based on those already specified in the structure, to increase usability and flexibility. That indicates the need for multilevel modeling in the document structure. For example, a table with predefined rows and columns can be used for invoices, or a special type of paragraph can indicate the higher importance of content for readers.

B. Conceptualization of Documents

Based on the previous considerations, we can now assemble the conceptual models. We use already mentioned language OntoUML which uses high-level and well-defined terms from the Unified Foundational Ontology (UFO) as stereotypes and significantly enhances semantics and expressiveness of basic Unified Modeling Language (UML). Details about the language and the ontology are fully explained in [5]. The connector of all the introduced models is the *document content*, the carrier of information. All models are connected, compatible, and describe different viewpoints introduced in the previous section. Moreover, NS patterns and modularization are well observable in the following models.

1) Concern-based Document Modularization: Figure 1 shows the diagram of the conceptual model with the separation of concerns pattern for documents. A document is a modular structure composed of module variants. Concerns as the drivers of modularization are naturally binding elements of documents to groups. Cross-cutting concerns are then the special case of general concerns in case they produce variants of document modules. Documents can be rendered using many templates, while the content is still the same. That separates a used style and typography from the actual content.

For example, in a manual for a software product, there are the following concerns: installation, usage, warranties, etc. Some of those have sub-concerns, which creates submodules, e.g., installation for various platforms. A cross-cutting concern, in this case, can be the language. Variants of "installation" are formed by using different natural languages. The manual is an ordered collection of various variants. Thus it is possible to have a multi-language manual, but also language specific manual, or just installation manual in English and then reuse these module variants easily. Finally, the manual can be then rendered with a template for printing, website, annotated XML, eBook, and so on.

Language is a typical cross-cutting concern in documents, but it can also be a case of general concern for creating modules. Consider a document about some ancient language. Probably some top-level module will be about the language concern with sub-concerns related to different parts of the language. Such document can be published in many languages as a cross-cutting concern as well.

 ISBN: 978-1-61208-612-5

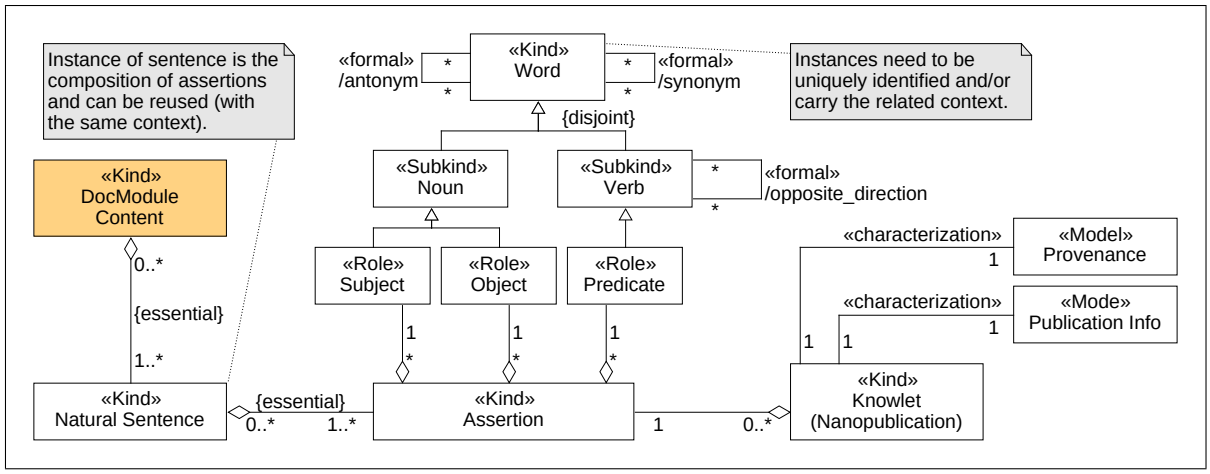

Figure 2. Conceptualization of meaning encoded in nanopublications.

2) Meaning in Nanopublications: The way a meaning is encoded within a document module content is shown in Figure 2. A content is formed by natural sentences, which are essential for the content as a whole. In a sentence, there can be one or more assertions, which are triplets in a simplified view: subject, predicate, and object. It is possible to form multiple assertions with the same meaning by using synonyms, and by switching subject with objects while using predicate for opposite direction.

Knowlet, or so-called nanopublication, is such an assertion with additional information and provenance as characterizations. Nanopublications are widely used within semantic webs and Resource Description Framework (RDF) in general, as described in [22] and [25]. Each instance of a word should be uniquely identifiable, in semantic web this problem is solved by the use of Uniform Resource Identifier (URI). For example, even with a simple assertion like *cat is white*, we need to know which cat the assertion is about, or if it is about all cats. The context is crucial for assertions, but it is hard to be adequately captured [22].

This expression of meaning could allow machines to read and understand the content in a more efficient way than is possible with text mining. Moreover, a semantic search, comparison, or reasoning can be built in a more straightforward way. It could lead to easier work with the documents, their parts and changes, and significant resource savings.

3) Document Content Structuring: The task of document content structuring has been addressed many times through syntax for composing documents and systems like the already-mentioned Pillar. For our purpose, the conceptualization is designed on a higher abstraction level, as shown in Figure 3. A content of the module is composed of block elements that contain text and inline elements that decorate part of text within a block element. Those types of elements are essentially powertypes [26] in the conceptualization, and their instances are particular usages of them. For example, the most common instance of a block element is a paragraph, and an instance of a paragraph is a particular paragraph containing a particular text, which is in our model covered by the atomic content kind not further subdivided. It works similarly for figures, pieces of data, file imports, and so on.

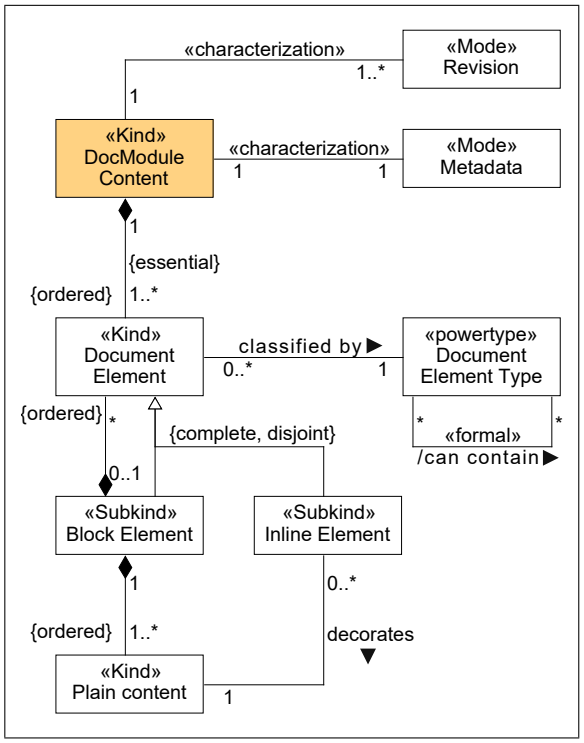

Figure 3. Conceptualization of structuring document module content.

Element type instances can be the well-known unordered or ordered lists, tables, definition lists, links, forms, cross-references, figures, quotes, external references, etc. On top of that, using powertypes allows defining new structural elements with different semantics, e.g., an important paragraph, specific table combined with a form, or external file. Metadata for each module content and document element can be provided. Content may be maintained as revisions that allow keeping track of changes.

C. Next Steps Towards Evolvable Documents

The last part of our approach is about the next steps which are suggested to be done in the near future as a consequence

based on the introduced conceptualization. Of course, the domain of documents is changing rapidly as well as the computer science which affects it significantly. Therefore is not just single possible way how to achieve evolvable documents and many options need to be explored, evaluated and then found out if there is some more suitable way. Mentioned steps seem to us very promising based on our own experience.

1) A Prototype of Evolvable Documents System: One of the possible next steps is to design and develop a simple prototype which further elaborates ideas from this paper and implements them. The result should be easy to use in any domain. The prototype would serve to find proof(s) of concept and to uncover new challenges.

The process of prototype development would be simplified by using the provided conceptualization and could explore missing, incorrect or unnecessary concepts by induction. Very important is to develop a system which is evolvable itself regarding Normalized Systems theory and it is not discouraging users with a complicated user interface.

2) A Methodology for Evolvable Writing: The prototype itself is not something that can be used directly and widely. However, during the process of further elaboration, some form of generic guidelines for creating evolvable documents can emerge. The possibility to write evolvable documents is highly affected by selected tools and formats.

Instead of developing a new solution, there is an alternative to build a more generic methodology and try to implement it as an integration of solutions currently available. Many of such current tools have been already mentioned: Git, Pandoc, LaTeX, Markdown, XML, GitHub, Pillar, etc.

IV. Conclusion

In this paper, we presented our initial approach to evolvable documents based on the principles of Normalized Systems theory but, compared to the related work, applicable in generic. The presented conceptualization is the basis of this genericity. By incorporating modularization based on the semiotic ladder and the NS concepts together with the ontology-driven conceptual modeling language OntoUML, we uncovered different aspects and challenges in the documents domain. The presented tightly-related conceptual models demonstrate the power of modularization and they can become a foundation for further discussion and building of a methodology or a system prototype using the model-driven development (MDD) methods. During the future research, it is likely that the models will need to be extended both in scope and detail.

Acknowledgments

This research was supported by the CTU grant No. SGS17/211/OHK3/3T/18, and was partially done as part of Summer Camp '17 organized by the FIT CTU in Prague, and also of Normalized Systems Summer School '17 organized by the University of Antwerp. This work also contributes to the CTU's ELIXIR CZ Service provision plan.

References

[1] H. Mannaert, J. Verelst, and P. De Bruyn, Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design. Kermt (Belgium): Koppa, 2016.

[2] G. Oorts, H. Mannaert, P. De Bruyn, and I. Franquet, "On the evolvable and traceable design of (under) graduate education programs," in Enterprise Engineering Working Conference. Springer, 2016, pp. 86–100.

[3] G. Oorts, H. Mannaert, and P. De Bruyn, "Exploring design aspects of modular and evolvable document management," in Enterprise Engineering Working Conference. Springer, 2017, pp. 126–140.

[4] G. Oorts, H. Mannaert, and I. Franquet, "Toward evolvable document management for study programs based on modular aggregation patterns," in PATTERNS 2017: the Ninth International Conferences on Pervasive Patterns and Applications, February 19-23, 2017, Athens, Greece/Mannaert, Herwig [edit.]; et al., 2017, pp. 34–39.

[5] G. Guizzardi, Ontological foundations for structural conceptual models. CTIT, Centre for Telematics and Information Technology, 2005.

[6] B. Duyshart, The Digital Document. Taylor & Francis, 2013.

[7] T. Arloing, Y. Dubois, S. Ducasse, and D. Cassou, "Pillar: A versatile and extensible lightweight markup language," in Proceedings of the 11th edition of the International Workshop on Smalltalk Technologies. ACM, 2016, p. 25.

[8] M. Dominici, "An overview of pandoc," TUGboat, vol. 35, no. 1, 2014, pp. 44–50.

[9] S. Kottwitz, LaTeX Cookbook. Packt Publishing, 2015.

[10] Wikipedia, Template Engines: JavaServer Pages, WebMacro, ASP. NET, Template Engine, Web Template System, Web Template Hook Styles, Haml, Template Processor. General Books, 2011.

[11] K. Ram, "Git can facilitate greater reproducibility and increased transparency in science," Source code for biology and medicine, vol. 8, no. 1, 2013, p. 7.

[12] S. Chacon and B. Straub, Pro Git, ser. The expert's voice. Apress, 2014.

[13] E. Westby, Git for Teams: A User-Centered Approach to Creating Efficient Workflows in Git. O'Reilly Media, 2015.

[14] K. Roebuck, Document Management System (DMS): High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors. Lightning Source, 2011.

[15] V. Pal, Alfresco for Administrators. Packt Publishing Ltd, 2016.

[16] B. Leuf and W. Cunningham, The Wiki Way: Quick Collaboration on the Web. Addison-Wesley, 2001.

[17] A. Porter, WIKI: Grow Your Own for Fun and Profit. XML Press, 2013.

[18] C. Bunch, Automated Generation of Documentation from Source Code. University of Leeds, School of Computer Studies, 2003.

[19] R. K. Stamper, "Applied semiotics," in Proceedings of the Joint ICL/University of Newcastle Seminar on the Teaching of Computer Science, Part IX: Information, B. Randell, Ed., 9 1993, pp. 37–56.

[20] B. Frohmann, "Revisiting "what is a document?"," Journal of Documentation, vol. 65, no. 2, 2009, pp. 291–303.

[21] C. K. Ogden and I. A. Richards, "The meaning of meaning: A study of the influence of thought and of the science of symbolism," 1923.

[22] B. Mons, H. van Haagen, C. Chichester, J. T. den Dunnen, G. van Ommen, R. Hooft et al., "The value of data," Nature genetics, vol. 43, no. 4, 2011, pp. 281–283.

[23] E. Duval, W. Hodgins, S. Sutton, and S. L. Weibel, "Metadata principles and practicalities," D-lib Magazine, vol. 8, no. 4, 2002. [Online]. Available: http://www.dlib.org/dlib/april02/weibel/04weibel.html

[24] R. Cyganiak, D. Wood, and M. Lanthaler, "RDF 1.1 concepts and abstract syntax. W3C Recommendation," 2014. [Online]. Available: https://www.w3.org/TR/rdf11-concepts/

[25] T. Kuhn, P. E. Barbano, M. L. Nagy, and M. Krauthammer, "Broadening the scope of nanopublications," in Extended Semantic Web Conference. Springer, 2013, pp. 487–501.

[26] G. Guizzardi, J. P. A. Almeida, N. Guarino, and V. A. de Carvalho, "Towards an ontological analysis of powertypes," in JOWO@IJCAI, 2015. [Online]. Available: http://ceur-ws.org/Vol-1517/JOWO-15_FOfAI_paper_7.pdf

 ISBN: 978-1-61208-612-5