Managing Variability and Evolution of Business Document Models[⋆]

Christian Pichler[1], Martina Seidl[2], and Christian Huemer[2]

[1] Inter-Organizational Systems
Research Studios Austria
Vienna, Austria
cpichler@researchstudio.at

[2] Institute of Software Technology and Interactive Systems
Vienna University of Technology
Vienna, Austria
{seidl, huemer}@big.tuwien.ac.at

Abstract. The United Nations Centre for Trade Facilitation and eBusiness (UN/CEFACT) standardizes business documents for electronic data interchange. Their approaches towards UN/EDIFACT and XML have later been followed by a conceptual modeling approach called Core Components (CC). Having used this approach for four years in practice, it became evident that the support for managing business document models is a prerequisite for successfully utilizing CC. This includes handling variants of business document models on the one hand, and managing the evolution of business document models on the other hand. In this paper we propose an approach to face these challenges by the means of Software Product Line Engineering (SPLE) in combination with dedicated model management operators. The contribution of the approach is twofold. First, SPLE is successfully applied in a new field enabling us to manage variants of business document models. Second, the model management operators support the evolution of business document model variants, whereas the operators defined, contribute to the evolution of product lines as well.

1 Introduction

Seamless information exchange between business partners is inevitable for successful collaboration in electronic commerce. For exchanging information electronically, standardized formats are required which are provided through Standard Developing Organizations (SDOs). These standards are typically created for a particular domain or industry. Business document standards may be distinguished into standards defined on the conceptual level and standards defined on

[⋆] The work of Research Studios Austria is funded by the Austrian Federal Ministry of Science and Research. Furthermore, this work has been carried out under the research grant Public Private Interoperability (No. 818639) of the Austrian Research Promotion Agency (FFG).

the transfer syntax level [1]. Defining a standard on the conceptual level means that a standard is defined using models such as UML class diagrams [2]. The conceptual representation is then used for generating the transfer syntax, such as an XML Schema schema [3], in the following denoted as XML schema.

Such a conceptual approach is envisioned by the Core Components technology [4] of the United Nations Centre for Trade Facilitation and eBusiness (UN/CEFACT), which originally became famous for maintaining the United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport (UN/EDIFACT) standards [5]. UN/CEFACT provides reusable Core Component assets which serve as a basis for creating business document models. One representative example of a business document model created based on Core Components is UN/CEFACT's Cross Industry Invoice (CII) [6] which has recently been mandated for electronic invoicing within the European Union [7]. Furthermore, UN/CEFACT accommodates generating XML schemas from conceptual models [8] through proper rules specified in the XML Naming and Design Rules (NDR) [9].

Having worked on tool support for the Core Components for four years, we recognized that successfully utilizing Core Components is inhibited due to several reasons. These reasons include deficiencies in managing variants as well as in coping with the evolution of business document models. Such types of problems are addressed in Software Product Line Engineering (SPLE) [10] as well as Software Configuration Management (SCM) [11]. In particular, SPLE deals with the strategic reuse of software systems sharing a common, managed set of features [12]. Furthermore, one aspect of SCM is managing different versions of software systems. Versions are differentiated into revisions as well as variants [13]. Revisions emerge along the time dimension and replace preceding revisions whereas variants intentionally coexist. Furthermore, concepts from SPLE promise benefitial advantages when being used for managing variants of software systems [14].

Nowadays, models are an important artifact in Software Engineering with the purpose of documenting software systems or for performing Model-Driven Engineering (MDE) [15]. The combination of MDE as well as SPLE seems promising resulting in the ability to leverage the benefitial advantages of both, MDE as well as SPLE [16]. Although there is currently much effort spent on model management [17] and the development of model versioning systems, there is, to the best of our knowledge, no approach which exactly fits our needs. The context of Core Components offers, compared to model versioning for conventional UML models, a restricted environment. The Core Components technology specifies the usage of Core Components and as a consequence this additional information may be used for managing variants as well as for coping with the evolution of business document models. In being confronted with a SCM problem, we propose to face these challenges by the means of SPLE as well as dedicated model management operators.

This paper is structured as follows. In Section 2 we provide the necessary background on UN/CEFACT's Core Components serving as a basis for business

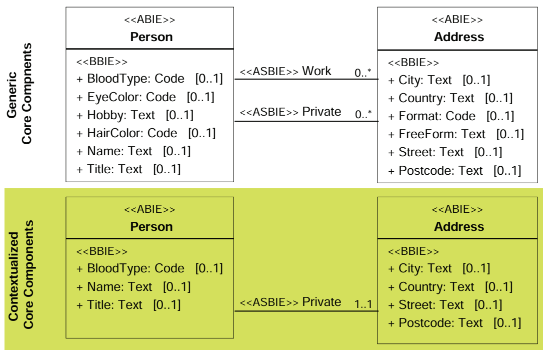


Fig. 1. Simplified Core Component Concept.

document modeling. Section 3 describes an example business document model and elaborates on the challenges encountered in business document modeling. In Section 4 we first apply PLE in the field of business document modeling, and second, introduce model management operators for dealing with the evolution of business document models. Section 5 describes related work. In Section 6, we conclude the paper with a critical discussion and an outlook on future work.

2 Background: UN/CEFACT's Core Components

UN/CEFACT's Core Components represent domain independent, transfer syntax independent, reusable building blocks for assembling business documents. The Core Component concepts as well as the underlying metamodel are defined in the Core Components Technical Specification (CCTS) [4]. For utilizing the Core Components in state-of-the-art modeling environments it would be sufficient to represent core components using class diagrams of the Unified Modeling Language (UML). However, since the Core Component concept "derivation by restriction", discussed in the following, is currently not feasible in UML, we introduced the UML Profile for Core Components (UPCC) [18, 19] which we submitted to UN/CEFACT for standardization.

For reasons of simplicity, the concepts are presented in an abridged manner—the full specification of the concepts may be found in [4]. Basically, we distinguish between the *Basic Business Information Entities* (BBIE), *Aggregated Business Information Entities* (ABIE), and *Association Business Information Entities* (ASBIE). An ABIE contains one or more BBIEs and an ASBIE creates relationships between ABIEs. An example illustrating a simple Core Components model is shown in the *Generic Core Components* layer of Figure 1. In particular, Figure 1 shows the ABIE `Person` with the BBIEs `Name`, `Title`, etc. Furthermore, `Person` is associated with the ABIE `Address` twice, namely through the ASBIE `Private` and the ASBIE `Work`.

For the application in concrete business scenarios, the Core Components must be customized to the context of the particular domains and the requirements of the involved organizations in order to obtain concrete business documents. For instance, SDOs of different business branches may use the *Generic Core Components* model as a basis for creating their variants fulfilling their domain-specific requirements. The Core Component concept specifies that creating contextualized business document models must follow a "derivation by restriction" mechanism. This means that a contextualized business document model is created through removing elements from a particular Core Component. For example, in Figure 1, the layer *Customized Core Components*, illustrates the customization of the *Generic Core Components* layer for fitting particular domain requirements. For example, the ABIE `Person` does not contain the BBIE `HairColor`, or the ASBIE `Working` is omitted.

Furthermore, UN/CEFACT provides the publicly available *Core Component Library* (CCL) [20] containing predefined Core Components. The *Generic Core Components* layer in Figure 1 illustrates an excerpt from the CCL. Those predefined Core Components may be reused for defining a multitude of business document models in various business contexts. Nevertheless, the development of concrete business documents is a highly dynamic process where mulitple participating organizations are involved. The next section presents a typical scenario which illustrates problems arising in such a dynamic development environment.

3 Motivation and Challenges

In the Core Component Technical Specification (CCTS) [4], UN/CEFACT provides the foundation for creating Core Component (CC) models through the underlying metamodel as well as additional rules. In the following we describe challenges encountered in dealing with CC models. We elaborate on managing business document model variants as well as business document model evolution.

3.1 Variant Management

Following the Core Component approach, as elaborated on above, Standard Developing Organizations of different business branches may utilize the Generic Domain Model as basis for creating variants—fulfilling their domain-specific requirements. Note that the Generic Domain Model illustrated in Figure 2 represents an excerpt from the Core Component Library (CCL). For example, in the healthcare domain it is necessary to represent information on the blood type of a particular person whereas the same information is irrelevant in the commerce domain. As a result, two variants of the Generic Domain Model are created, the Healthcare Domain Model and the Commerce Domain Model. Both domain models represent copies of the Generic Domain Model with domain specific adaptions, as illustrated in Figure 2, Mark A and Mark B.

Similar to a business document model based on the CCL, these specific domain models may again serve as a basis for different organizations within the domain to define their own variants of business document models, as illustrated in Figure 2, Mark C. For example, healthcare providers may further restrict the Healthcare Domain Model for representing patient information. Consequently, a wealth of different variants of business document models have to be managed.

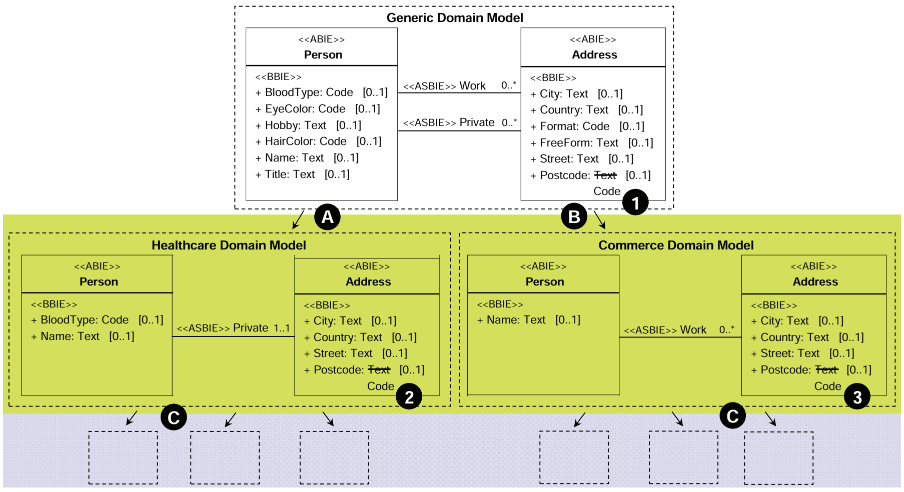


Fig. 2. Variants of a Business Document Model.

3.2 Model Evolution

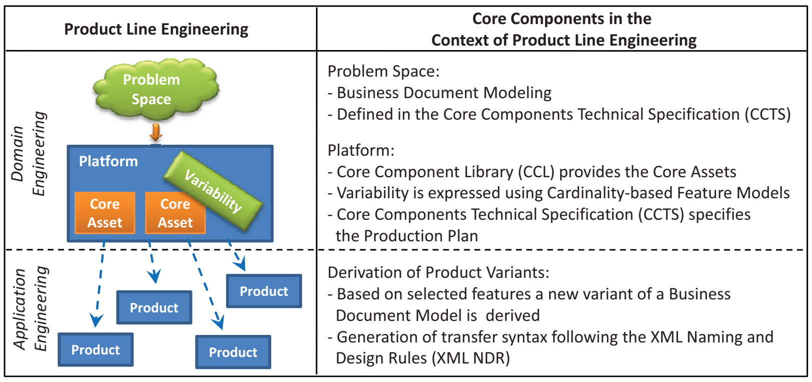
Business document models as well as their variants evolve over time due to different reasons, such as extensions, refactorings, or bug fixes. Assume that an inappropriate data type `Text` has been used in the CCL. Therefore, a new version of the Core Component Library is released, where the type of the BBIE `City` is changed from `Text` to `Code`, as illustrated in Figure 2, Mark 1.

As a result of the changes applied to the Generic Domain Model, it is necessary to propagate these changes to all variants which are based on the Generic Domain Model. Applied to the example, it is necessary to adopt the Healthcare Domain Model was well as the Commerce Domain Model, as indicated in Mark 2 and Mark 3, of Figure 2.

3.3 Supporting Variant Management and Model Evolution

For managing variants of business document models we propose to utilize concepts from Software Product Line Engineering (SPLE) [10]. Applying these concepts to business document modeling enables us to efficiently manage variants of business document models. Furthermore, as illustrated in the example above, business document models are also subject to evolution. In terms of SPLE this means that the core assets of the platform change. For dealing with evolution we need dedicated model management operators. Having such operators at hand allows to understand and manage the evolution of business document models within a product line. In particular, the operators support the evolution of the product line's core assets as well as the corresponding evolution of the product line's products. Hence, these operators also contribute to today's challenges of evolution in Model-Driven Product Line Engineering (MDPLE).

Exploring the evolution of product lines within the context of business document modeling offers a unique environment having benefitial advantages. Cre-



Core Components in the Context of Product Line Engineering

Problem Space:
- Business Document Modeling
- Defined in the Core Components Technical Specification (CCTS)

Platform:
- Core Component Library (CCL) provides the Core Assets
- Variability is expressed using Cardinality-based Feature Models
- Core Components Technical Specification (CCTS) specifies the Production Plan

Derivation of Product Variants:
- Based on selected features a new variant of a Business Document Model is derived
- Generation of transfer syntax following the XML Naming and Design Rules (XML NDR)

Fig. 3. Business Document Modeling in the Context of Product Line Engineering.

ating variants of business document models follow the "derivation by restriction" mechanism, hence the variability among variants as well as the evolution of business document models is constrained. Therefore, the business document modeling environment offers a constrained environment for exploring first steps towards dealing with the evolution of Model-Driven Product Lines.

4 Model Management

In this section, we propose concepts for addressing the challenges encountered in business document modeling. These include, the application of SPLE concepts to manage variants of business document models, as well as the definition of model management operators for supporting the evolution of business document models. The meta models and models, representing the core assets of the product line's platform in business document modeling, are defined using the Eclipse Modeling Framework (EMF), in particular Ecore [21]. The metamodel implemented represents an Ecore-based equivalent of the metamodel specified in the Core Components Technical Specification (CCTS) [4]. Furthermore, the model management operators defined, address the evolution of business document models. The evolution of meta models and feature models will be discussed in Section 6, future work. Another important aspect is the granularity of evolution. In the context of business document models we perform model evolution on the level of classes, attributes, as well as associations.

4.1 Variant Management

In the following, we illustrate the application of Software Product Line Engineering (SPLE) concepts to business document modeling, as illustrated in Figure 3. Generally speaking, Product Line Engineering (PLE) consists of Domain Engineering as well as Application Engineering.

Domain Engineering is comprised of defining the problem space, creating a platform containing core assets and variability points, as well as defining a production plan [10]. Applied to the context of creating business document models based on Core Components we observe the following. Clearly, the problem space addressed in the context of Core Components is the area of business document modeling. The platform of the product line is implicitly defined through the already existing Core Component Library which contains all core assets which may be used in product variants, i.e. business document models. As a next step, it is necessary to specify the production plan which describes how product variants are derived from the platform. However, as illustrated earlier, the CCTS defines that business document models, based on predefined Core Components, must only be created utilizing the "derivation by restriction" mechanism. Therefore, this mechanism represents the production plan for creating product variants, representing variants of business document models. Currently, for deriving business document model variants, we provide tool support through the VIENNA Add-In [22].

Application Engineering addresses the process of deriving product variants from the artifacts defined in process of Domain Engineering. Applied to the context of Core Components, variants of business document models, based on the Core Component Library, are derived. Furthermore, the variants of business document models are transformed into XML schemas according to the XML Naming and Design Rules (NDR) [9].

4.2 Model Evolution

When managing the variants of business document models, we are confronted with two different challenges: (1) we intend to keep the hierarchy of business document models as redundancy free as possible, i.e., we aim at the detection of business document models which are introduced twice in order to keep the overall structure as simple as possible and (2) if one business document model is modified than all derived child documents have to be updated accordingly, i.e., we are confronted with the evolution of the business document models. For the definition of the necessary model management operators, we use the following simplified view on a business document model based on the concept of Core Components.

Definition 1 (Business Document Model). *The business document model $\mathcal{M}$ over a set of properties P is a tuple of the form*

$$\langle \mathsf{BBIE}, \mathsf{ABIE}, \mathsf{ASBIE}, agg, props \rangle$$

where BBIE denotes the set of Basic Business Information Entities, ABIE denotes the set of Aggregated Business Information Entities, and ASBIE denotes the set of Association Business Information Entities. For each $a \in \mathsf{ASBIE}$ it holds that $a = (id, a_1, a_2)$ with $a_1, a_2 \in \mathsf{ABIE}$ and where id is an identifying label. The composition of BBIEs in order to form ABIEs is expressed by the function $agg : \mathsf{ABIE} \rightarrow \mathsf{BBIE}^{\mathsf{BBIE}}$. In order to embrace the different types of entities we introduce the set $\mathcal{E}$ with $\mathcal{E} = \mathsf{BBIE} \cup \mathsf{ABIE} \cup \mathsf{ASBIE}$. Finally, the function $prop : \mathcal{E} \rightarrow \mathsf{P}^{\mathsf{P}}$ assigns each entity a set of properties.

The set P contains various types of properties like types and visibilities which are necessary to describe the different kinds of entities in more detail. In the following, this vague definition is sufficient as we are not concerned with the semantics of the properties.

Redundancy Elimination in the Hierarchy of Business Documents. If many variants are derived according to the needs of different organizations or companies, the dependency tree rapidly grows. As a consequence, a user new to business document modeling based on Core Components is not easily able to identify which business document model fits his/her specific needs. Then the user would be either discouraged to use the Core Components or the user would start to put his/her adaptations at a very high level of the business document models hierarchy although a suitable business document model is already at hand. Hence, the hierarchy expands into the breath. We propose the implementation of an operator which is able to identify equivalent business document models or even two business document models $\mathcal{M}_1$ and $\mathcal{M}_2$ where $\mathcal{M}_1$ is the subset of $\mathcal{M}_2$ even if they are located in different branches. Furthermore, such an operator may be used to check whether the hierarchy of business document models is valid. The subset operator is defined as follows.

Definition 2 (Subset Operator). *A business document model $\mathcal{M}_1$ given by $\langle \mathsf{BBIE}_1, \mathsf{ABIE}_1, \mathsf{ASBIE}_1, agg_1, props_1\rangle$ is a subset of a business document model $\mathcal{M}_2 = \langle \mathsf{BBIE}_2, \mathsf{ABIE}_2, \mathsf{ASBIE}_2, agg_2, props_2\rangle$ denoted by $\mathcal{M}_1 \subseteq \mathcal{M}_2$ iff it holds that $\mathsf{BBIE}_1 \subseteq \mathsf{BBIE}_2$, $\mathsf{ABIE}_1 \subseteq \mathsf{ABIE}_2$, and $\mathsf{ASIE}_1 \subseteq \mathsf{ASIE}_2$. Furthermore, for each $a \in \mathsf{ABIE}_1$ it holds that $agg_1(a) \subseteq agg_2(a)$ and for each $e \in \mathsf{BBIE}_1 \cup \mathsf{ABIE}_1 \cup \mathsf{ASIE}_1$ it holds that $prop_1(e) \subseteq prop_2(e)$.*

With this subset operator we are able to identify business document models which are wrongly positioned in the hierarchy of business document models and in consequence we are able to optimize the overall structure of the model hierarchy. The subset operator is not only necessary for organizing the repository, but also in the context of evolution as we see in the following.

Evolution of Business Document Models. As any model, the specifications of business documents may evolve over time. This evolution comprises the extension due to more precisely stated requirements as well as updates like refactorings or bugfixes of either the generic domain model or of the dedicated domain models. It is even imaginable that formerly defined model elements are replaced or even removed if it turns out from experience that these elements are often used in an incorrect manner or that they are not used at all. Consequently, the changes have to be propagated to all business documents models which are created based on the modified business document model in order to ensure compliance of the different business documents. The relationships between business document models form a tree. Hence, changes are propagated along the affected branch. Evolution is an issue in the following situation. Given the business document models $\mathcal{M}_1$ and $\mathcal{M}_2$ where $\mathcal{M}_1$ is derived from $\mathcal{M}_2$, i.e. it holds that

$\mathcal{M}_1 \subset \mathcal{M}_2$, and $\mathcal{M}_2$ evolves to $\mathcal{M}'_2$, the relationship $\mathcal{M}_1 \subset \mathcal{M}'_2$ does not hold any more. In order to identify the differences between two business documents models we define the difference operator as follows.

Definition 3 (Difference Operator). *The difference between a business document model* $\mathcal{M}_1 = \langle \mathsf{BBIE}_1, \mathsf{ABIE}_1, \mathsf{ASBIE}_1, agg_1, props_1 \rangle$ *and a business document model* $\mathcal{M}_2 = \langle \mathsf{BBIE}_2, \mathsf{ABIE}_2, \mathsf{ASBIE}_2, agg_2, props_2 \rangle$ *(denoted by* $\mathcal{M}_1 \Delta \mathcal{M}_2$*) is defined by the tupel* $\langle added, removed, updated \rangle$ *where* $added = \mathcal{E}_2 \backslash \mathcal{E}_1$*,* $removed = \mathcal{E}_1 \backslash \mathcal{E}_2$*, and* $updated = \{e \in \mathcal{E}_1 \cap \mathcal{E}_2 | props_1(e) \neq props_2(e)\} \cup \{e \in \mathsf{ABIE}_1 \cap \mathsf{ABIE}_2 | agg_1(e) \neq agg_2(e)\}$ *where* $\mathcal{E}_i = \mathsf{BBIE}_i \cup \mathsf{ABIE}_i \cup \mathsf{ASBIE}_i$ *with* $i \in \{1, 2\}$*.*

In this context, evolution is only possible in one direction: changes are propagated from the more general business document models to the its more specific offsprings. This is due to the "derivation by restriction" mechanism specified in the Core Components Technical Specification [4].

4.3 Model Management by Example

In the following, we illustrate the application of the model management operators to the business document models variants, illustrated in Figure 2.

From the definition of a business document model, it follows that in the Generic Domain Model shown in Figure 2 $\mathsf{ABIE} = \{\mathsf{Person}, \mathsf{Address}\}$, $\mathsf{ASBIE} = \{(private, \mathsf{Person}, \mathsf{Address}), (work, \mathsf{Person}, \mathsf{Address})\}$, and BBIE contains Name, Title, City, Country, etc. The function *agg* relates elements of BBIE with elements of ABIE, i.e., Name and Title are features of Person whereas City and Country are features of Address. Finally, the *prop* function assigns additional information like types, visibility, multiplicities etc. to the entities.

In Figure 2 the Healthcare Domain Model and the Commerce Domain Model are subsets of the Generic Domain Model, but the Healthcare Domain Model and the Commerce Domain Model are not related via the subset operator due to the different ASBIEs. Applying the difference operator to the updated Generic Domain Model, illustrated in Figure 2, shows that the only change is given by the update of the type property of the BBIE Postcode. The changes detected through difference operator need to be propagated to the different business document model variants based on the Generic Domain Model. Therefore, the Healthcare Domain Model and the Commerce Domain Model need to be updated accordingly.

5 Related Work

Combining Model-Driven Engineering (MDE) and Product Line Engineering (PLE) provides substantial benefits method for creating similar products and systems [16]. Several examples may be found where model-driven product line engineering proved to be successfull, including [23]. We study concepts from PLE which have an impact on business document model variants.

For expressing the variability among variants of business document models, the concept of cardinality-based feature modeling, as described by Czarnecki et al. [24], is highly relevant. For mapping feature models to core assets of a model-driven product line, different approaches exist. For instance, Czarnecki et al. [25] propose a template-based approach for mapping feature models to data models or behavioral models.

Several approaches and tools are available for supporting the process of Model-Driven Product Line Engineering (MDPLE). Antkiewicz et al. [26] introduce the FeaturePlugin which supports creating Feature Models. Ecore.fmp, a successor of the FeaturePlugin, is introduced by Stephan et al. [27], which allows instantiating class models as feature models. Furthermore, Ecore.fmp allows viewing Ecore models as Feature Models, as well as the configuration of Ecore models which may be interpreted as instantiating product variants. Heidenreich et al. implement a tool named FeatureMapper [28], which enables creating mappings between Feature Models and Ecore models. Furthermore, FeatureMapper allows deriving product variants based on a specified feature configuration. Beuche [29] describes pure::variants, which allows realizing product lines in combination with Ecore models as well.

Groher and Voelter [30] present an approach for Aspect-Oriented Model-Driven Product Line Engineering (AO-MD-PLE). In their approach, they also illustrate negative variability in structural models where an overall model is connected to feature models. Specific feature configuration then serve as a basis for instantiating model variants. For implementing negative variability, a tool named XVar is presented.

Though a number of approaches exist for creating model-based product lines, the support for the evolution of a product line's assets is limited. The necessity for addressing the evolution in product lines has also been identified in literature, such as [31, 32]. For example, Dhungana et al. [33] present their work on supporting product line evolution by organizing variability models as a set of interrelated model fragments. In our work, we address the evolution of business document models, representing the core assets of the product line. We define dedicated model management operators for enabling model evolution in our product lines.

6 Conclusion and Future Work

In this paper we identified the need for managing variants of business document models as well as the need for supporting the evolution of business document models, which are both necessary for successfully utilizing UN/CEFACT's Core Components concept.

The contribution of this paper is two-fold. First, we applied well-known concepts from PLE in a new field, namely business document modeling. Doing so enables us to effectively manage variants of business document models. Furthermore, we investigated existing tool support for creating model-based product lines, as well as deriving product variants, i.e. business document model variants. Second, we defined model management operators as a first step towards supporting evolution in model-based product lines. Though defined in the context of business document models, the operators defined are applicable to software models, e.g. UML class diagrams in software product lines, as well.

Future work, based on the findings presented in this paper, includes the following. First, the implementation of the model management operators is continued. Since we actively participate in UN/CEFACT, we have access to a pool of models which allows us to evaluate the concepts proposed. The evaluation allows us to assess the completeness and correctness of the model management operators proposed. Furthermore, the evaluation helps us gain experience in the evolution of business document models which may lead us to propose a classification of possible evolution scenarios. In a consecutive step it is also necessary to address the evolution of other artifacts present in model-driven product lines whereas existing approaches, such as presented in [34], are subject to evaluation. This includes the evolution of metamodels, feature models, feature configurations, as well as co-evolution of business document model instances. It is also necessary to address the matter of co-evolution. This means that changes applied to business document models also affect actual instances of the business document models, which needs to be handled. In the long-run, it is planned to provide tool support for managing the evolution of business document models.

References

1. Liegl, P., Zapletal, M., Pichler, C., Strommer, M.: State-of-the-art in business document standards. In: Proc. of the 8th Int. Conf. on Industrial Informatics, to appear, IEEE (2010)
2. OMG: Unified Modeling Language (UML) http://www.uml.org/.
3. W3C: XML Schema 1.1 http://www.w3.org/XML/Schema.
4. UN/CEFACT: Core Components Tech. Spec. (CCTS) 3.0 http://www.untmg.org/ccts/spec/3_0.
5. UN/CEFACT: United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport (UN/EDIFACT) http://www.unece.org/trade/untdid.
6. UN/CEFACT: Requirements Spec. Mapping for Cross Industry Invoice v.2.0
7. Europen Commission Export Group: Final Report on e-Invoicing
8. Huemer, C., Liegl, P.: A UML Profile for Core Components and their Transformation to XSD. In: Proc. of IEEE 23rd Int. Conf. on Data Engineering Workshop. (2007) 298–306
9. UN/CEFACT: XML Naming and Design Rules 3.0 http://www.unece.org/cefact/xml/xml_index.htm.
10. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering - Foundations, Principles, and Techniques. Springer (2005)
11. Tichy, W.F.: Tools for Software Configuration Management. In: Proc. of the Int. Workshop on Software Version and Configuration Control. (1988) 1–20
12. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2007)
13. Conradi, R., Westfechtel, B.: Version Models for Software Configuration Management. ACM Computing Surveys **30**(2) (1998) 232–282
14. Dalagarno, M., Beuche, D.: Variant Management. In: 3rd British Computer Scociety Configuration Management Specialist Group Conference, BCS MSG (2007)

15. Bézivin, J.: On the Unification Power of Models. Software and System Modeling **4**(2) (2005) 171–188
16. Czarnecki, K., Antkiewicz, M., Kim, C.H.P., Lau, S., Pietroszek, K.: Model-driven Software Product Lines. In: Comp. to the 20th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications, ACM (2005) 126–127
17. Bernstein, P.A., Melnik, S.: Model Management 2.0: Manipulating Richer Mappings. In: Proc. of the ACM SIGMOD Int. Conf. on Mgmt. of Data, ACM (2007)
18. Liegl, P.: Conceptual Business Document Modeling using UN/CEFACT's Core Components. In: Proc. of the 6th Asia-Pacific Conf. on Conceptual Modeling, Australian Computer Society (2009)
19. UN/CEFACT: UML Profile for Core Components (2009)
20. UN/CEFACT: UN/CEFACT's Core Component Library (UN/CCL) http://www.unece.org/cefact/codesfortrade/codes_index.htm.
21. Eclipse Foundation: Eclipse Modeling Framework (EMF) http://www.eclipse.org/modeling/emf/?project=emf.
22. VIENNA Add-In development team: Visualizing Inter-ENterprise Network Architectures http://vienna-add-in.googlecode.com/.
23. Wende, C., Heidenreich, F.: A Model-based Product-Line for Scalable Ontologies. In: Proc. of the 1st Int. Workshop on Model-Driven Product Line Engineering. (2009) 51–58
24. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Formalizing Cardinality-based Feature Models and their Specialization. Software Process: Improvement and Practice **10**(1) (2005) 7–29
25. Czarnecki, K., Antkiewicz, M.: Mapping Features to Models: A Template Approach Based on Superimposed Variants. In: Proc. of the 4th Int. Conf. on Generative Programming and Component Engineering. Volume 3676 of Lecture Notes in Computer Science., Springer (2005) 422–437
26. Antkiewicz, M., Czarnecki, K.: FeaturePlugin: Feature Modeling Plug-In for Eclipse. In: Proc. of the 2004 OOPSLA Workshop on Eclipse Technology eXchange (ETX), ACM (2004) 67–72
27. Stephan, M., Antkiewicz, M.: Ecore.fmp. A tool for editing and instantiating class models as feature models. Technical report, University of Waterloo (2008)
28. Heidenreich, F., Kopcsek, J., Wende, C.: FeatureMapper: Mapping Features to Models. In: Proc. of the 30th Int. Conf. on Software Engineering, ACM (2008)
29. Beuche, D.: Modeling and Building Software Product Lines with pure::variants. In: Proc. of the 12th Int. Software Product Line Conference, IEEE (2008) 358
30. Groher, I., Voelter, M.: Expressing Feature-Based Variability in Structural Models. In: Proc. of the Workshop Managing Variability for Software Product Lines. (2007)
31. McGregor, J.: The Evolution of Product Line Assets. Technical report, Carnegie Mellon Software Engineering Insitute (2003)
32. Svahnberg, M., Bosch, J.: Evolution in software product lines: Two cases. Journal of Software Maintenance **11**(6) (1999) 391–422
33. Dhungana, D., Neumayer, T., Grünbacher, P., Rabiser, R.: Supporting the Evolution of Product Line Architectures with Variability Model Fragments. In: Seventh Working IEEE / IFIP Conference on Software Architecture, IEEE (2008) 327–330
34. Rose, L., Paige, R., Kolovos, D., Polack, F.: An Analysis of Approaches to Model Migration. In: Proc. of the 1st Int. Workshop on Model Co-Evolution and Consistency Management. (2009) 6–15