A Document Model Management Framework based on Core Components

Michael Strommer, Christian Pichler
Research Studios Austria
Vienna, Austria
{firstname.lastname}@researchstudio.at

Philipp Liegl
Vienna University of Technology
Vienna, Austria
liegl@big.tuwien.ac.at

Abstract—The seamless and automatic exchange of electronic documents is one of the most important aims in today’s businesses. At the same time the need for a consistent framework for the management of electronic business documents, together with tool support rises. Tools, that foster the management of document models in a way to overcome interoperability issues, if such emerge. Because of various document formats interoperability is rarely achieved across the involved IT systems. Also, such MDA related tools are currently rare in the domain of business document engineering.

We therefore implemented a document model management framework, based on core components. This framework is built on an open source basis and may cope with interoperability problems providing transformation and schema generation components. The practical application of these features has been evaluated in a case study comprising two business document standards. The results of this case study underpin the relevance of the developed methods and show further directions for implementation.

I. INTRODUCTION

The automatic handling of inter-organizational or even cross-border business documents still remains a challenge in the domains of both, business-to-business (B2B) and administration-to-administration (A2A). Documents are often exchanged in the form of unstructured data as it is the case for example with PDF files, which hinders seamless integration of business documents and information systems. But even if structured data is exchanged, the variety of co-existing exchange formats, mostly based on XML Schema, may lead to interoperability failures. This is the case, when multiple document models are to be supported by the information systems involved. As a consequence Web Service interfaces must be adapted to cope with additional data formats and mappers must be implemented to support the different data representations.

One way to tackle these problems is to adopt the UN/CEFACT Core Components Technical Specification, which may be used to create context-specific document formats in a cost efficient and rapid way between the involved parties. Modeling takes place on a conceptual level only to abstract from implementation details and to foster collaborative modeling processes. Besides the CCTS conceptual modeling technique UN/CEFACT (United Nations Centre for Trade Facilitation and Electronic Business) also provides standardized, reusable document building blocks in a library called Core Components Library. So far, a consistent framework and tool support for modeling with core components is missing. In this paper we aim to close this gap and present a document model management framework based on CCTS and Enterprise Architect, called VIENNA Add-In. This framework copes with mapping and transformation problems of business document models in practice. The relevance of both, the framework and the VIENNA Add-In tool, is underpinned by a case study, involving the transformation of an e-Invoice of two rather complex business document standards. The contribution of this work is threefold:

1) We present a prototypical implementation of a document model management framework supporting various model-driven use cases (cf. Section V).
2) We provide empirical results of a complete mapping for two business document standards (cf. Section VI).
3) We show empirical results on the transformation of those two business document standards (cf. Section VI).

Before going into details we present a discussion of necessary concepts and our approach in Section II and III, respectively. We continue with related work in Section IV. At the end we conclude our work by means of a critical discussion and a future outlook, see Section VII.

II. CONCEPTUAL ENVIRONMENT

A. The Core Components Technical Specification: A Primer

The Core Components Technical Specification (CCTS) [1], maintained by UN/CEFACT, specifies a modeling approach for business document building blocks. Figure 1 gives an overview of the basic concepts of the Core Components Technical Specification. We distinguish between two major paradigms: Core Components (CC) and Business Information Entities (BIE). Core Components are context-independent building blocks for the creation of business documents. A core component does not have a specific business context, thus it may be used in any given scenario. In order to contextualize a core component to the specific needs of a given business domain, the concept of business information entities (BIE) is used.

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).
B. Core Components

The basic concept behind core components is the identification of objects and object properties. Thereby, object properties are divided into two sub-groups: simple object properties (e.g., date, name, age) and complex object properties (references to other objects). An object is represented using an aggregate core component (ACC). Aggregate core components represent an embracing container for simple object properties. In the center of Figure 1, the ACC `Tax` includes three simple object properties - `Code`, `Basis`, and `Rate`. Simple object properties are referred to as basic core components (BCC). Each simple object property has a data type known as core data type (CDT). The core data type of `Code` may be `Identifier`. A core data type defines the exact value domain of a given basic core component. In order to depict dependencies between different objects, the concept of association core components (ASCC) is used. In Figure 1 the ACC `Tax` is related to some other ACC using an association core component.

C. Business Information Entities

If core components are used in a certain business context, they become so called business information entities. Business information entities are always derived from an underlying core component by restriction. This implies that a business information entity must not contain any attributes or associations which have not been defined in the underlying core component. In Figure 1, a based on dependency between the business information entity `My_Tax` and the core component `Tax` signifies this fact.

Similar to core components, business information entities distinguish between three different elementary types. An aggregate business information entity (ABIE) is used to aggregate simple object properties. Simple object properties are denoted using the concept of basic business information entities (BBIE). On the right hand side of Figure 1 the ABIE `My_Tax` aggregates a single BBIE `Rate`. Note that the business information entity `My_Tax` does not contain all attributes of the underlying core component `Tax`. Thus, the ABIE restricts the underlying aggregate core component (ACC). The value domain for a given basic business information entity is defined using the concept of a business data type (BDT). The BBIE `Rate` contained in the ABIE `My_Tax` may be of type `Rate`. Dependencies between different business information entities are defined using the concept of association business information entities (ASBIE). Note that a business information entity must contain the name of the underlying core component. However, so called qualifiers may be used to facilitate the distinction between core components and business information entities. For example, the qualifier `My_` is used in Figure 1 for the definition of the ABIE `My_Tax`. Eventually, different ABIEs are aggregated to a final business document using a message assembly (MA). As shown on the right hand side of Figure 1, the message assembly tax document aggregates different ABIEs using the concept of association message assemblies (ASMA). For a detailed discussion on core components and business information entities see [2].

A similar relationship as established between core components and business information entities is also established between core data types and business data types. Essentially, a business data type must always be based on a core data type and may only use attributes which have already been defined in the underlying core data type.

D. Core Component Library

Since it is imperative that all business information entities are based on core components, a consolidated library of predefined core components must be provided, as shown on the left hand side of Figure 1. A global library of core components is currently developed and maintained by UN/CEFACT and has become known as Core Component Library (CCL) [3]. Business document modelers may search and retrieve core components from the core component library. Furthermore, anybody interested in the definition of core components may submit new core component definitions to UN/CEFACT, where the harmonization and standardization of core components is organized. Updated versions of the core component library are released twice a year, containing new core component definitions and updates of existing core components. Business information entities are themselves organized in libraries to which we refer to as business information entity libraries. Since business information entities are a domain-specific concept, standardized business information entities are provided per different domain rather than on a global level.

E. A UML Profile for Core Components

Core components in the core component library are defined using spreadsheets and thus integration into modeling tools is difficult. In order to apply the core component concepts as a means of modeling business documents we initiated the development of a UML profile that allows using core components with standard UML modeling tools. Today this standard has become known as the UML Profile for Core Components (UPCC) [4]. The UPCC fully represents all necessary concepts from the Core Component Technical Specification including business information entities and data types.

III. RESEARCH APPROACH

A. Matching vs. Mapping

Mapping is the task of defining semantic relationships between elements of different schemas or models manually,
i.e., $map_{AB} = M_A \leftrightarrow M_B$. Mapping has been subject to several research papers some of which are [5], [6], [7].

Contrary to this manual procedure exists the automatic method of matching which generates a best effort mapping, i.e., $\sim map_{AB} = match_{AB}$. There exist a multitude of matching approaches trying to overcome different problems occurring with structure, names, or cardinalities of correspondences. For a detailed discussion and categorization of matching strategies see for example [8].

1) Coma++: In our studies and experiments we used Coma++ [9] as a matching tool for schemas because it is very well documented and provides fair usability. Coma++ consists of three central components to support the matching process. The first component is called Schema Pool which organizes all the schemas being matched and maintains a graph-based, simplified representation of those schemas. The Execution Engine takes the schemas as input and tries to match similar concepts based on various matchers and user defined properties. The output is then stored in the third component called Mapping Pool which keeps the produced mappings and allows for further manual manipulation in order to refine the mappings. All this functionality can be accessed via a GUI in a very convenient and easy to use way as shown in Figure 2.

2) A Small Case Study: The automatic generation of mappings for the transformation of business document standards saves cost and time. Therefore, we evaluated the Coma++ tool by matching a CCL Address and a UBL Address. XML Schema imports and includes have been reduced to concentrate on the fragment of an Address.

Table I gives an overview of the matching results. Evidently, the matching results in this concrete example are rather good.

By eliminating additional concepts and refraining from structural issues we solely measured the name matching capabilities of the string matchers implemented within Coma++, i.e., Affix, Ngram, SoundEx, and EditDistance [10]. As shown in Figure 2 names in UBL and CCL are very tightly related and there are only marginal differences. The mapping from MarkCare to CareOf has been correctly identified whereas the correspondence PostalZone to PostcodeCode could not be established.

### Pros.
- Coma++ produces results for an initial mapping.
- Several name matchers and re-use oriented matchers based on dictionaries are included.
- Provides an easy to use graphical tool.
- Supports distribution of schemas, i.e., handles imports and includes.
- Support of fragment matching and pivot schemas.
- ComaAPI for integration into Java programs.

### Cons.
- Names must not be too different.
- The graph representation may not be computed for large schemas, which is the case for UBL and CCL. Neither of those could be loaded entirely into the workspace.
- Manually or automatically extracting fragments from large schemas is very time consuming.
- Matchings have to be validated by the user, who needs to have a broad knowledge of the underlying business domain.

Besides some rather technical than conceptual flaws of Coma++ our small evaluation illustrates the potential, name matchers may have on the domain of business document mapping.

### B. Pivot Model

Matching may leverage the amount of time needed to establish a mapping between two business document models. As soon as multiple models have to be integrated, the cost for mappers skyrockets. To avoid this, the general approach is to introduce a pivot model $M_P$ such as an ontology, which comprises all the concepts used in a certain domain. We already introduced the basic concepts for such an ontology based on Web Ontology Language (OWL) and core components in [11]. Then, a mapping is established from the models being

<table>
<thead>
<tr>
<th>Concept</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements UBL</td>
<td>24</td>
</tr>
<tr>
<td>Elements CCL</td>
<td>38</td>
</tr>
<tr>
<td>Real correspondences</td>
<td>24</td>
</tr>
<tr>
<td>True positives</td>
<td>17</td>
</tr>
<tr>
<td>False positives</td>
<td>3</td>
</tr>
<tr>
<td>False negatives</td>
<td>4</td>
</tr>
<tr>
<td>Precision</td>
<td>0.85</td>
</tr>
<tr>
<td>Recall</td>
<td>0.81</td>
</tr>
<tr>
<td>Fmeasure</td>
<td>0.83</td>
</tr>
</tbody>
</table>

**TABLE I**

MATCH RESULTS COMA++ FOR CCL AND UBL ADDRESS.
we do not deal with instances and their transformations right and or expressions on how to split or merge are neglected as output are one or more BCCs in the pivot model. In the pivot model. On the model layer we establish model mappings MM concepts to concepts of CCTS, i.e., map Schema based pivot model, i.e., map mapping language for the transformation of XML Schema-based instances called Malan is presented in [15]. Malan covers both, a specification of correspondences between elements and the generation of transformation programs based on these correspondences. The approach presented in [15] closely relates to model transformation technologies.

The manual work accompanying the task of schema mapping shall be automated by matching tools such as Coma++ [9], which we evaluated and compared to our approach earlier.

**IV. RELATED WORK**

Considering our model management framework in general and certain framework components in particular, we may identify several fields of related work.

**Schema Mapping and Matching.** Schema mapping has been evaluated by Legler and Naumann in [5]. They present a comprehensive classification of mapping approaches, based on the kind of correspondences supported, e.g., single or multiple correspondences. Furthermore, they evaluated state-of-the-art tools and pointed out the problems occurring when it comes to actual schema transformation. A theoretical foundation for model management and model mapping is presented by Bernstein in [14]. We base our definitions of model mappings and model operators like map, Match, or Diff upon these foundations. In our framework we use the Altova MapForce mapping model for defining and storing semantic correspondences upon which transformations are being performed. A mapping language for the transformation of XML Schema-based models is given by Legler and Naumann in [5].

**Document Management.** Document management tools that allow to model and integrate document formats are rare, especially for the UN/CEFACT Core Components approach. The tools mostly related to our framework are the iSURF eDoCreator [17] and the GEFEG.FX tool [18]. The eDoCreator may be described as an online repository allowing on-the-fly manipulation of UBL-based business documents. The GEFEG.FX tool supports data modeling and data integration besides several other features. Another important feature of GEFEG.FX is the support for standards, e.g., various EDI formats and CCTS. Also Microsoft BizTalk Server, another data and business process integration tool, is supported. A more generic toolset in the domain of XML Schema and XML data management is provided by Altova’s XMLSpy and MapForce. In combination these tools provide the capability to design data formats and generate XSLT transformations from mappings for document instances. A visual editor is provided, which provides user friendly design functionality. Though, no abstract conceptual model is provided for the definition of business document models.

### Table II

**Explicit Mapping Operators Supported.**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Simple Element</th>
<th>Complex Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUP</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>BCC</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ASC</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

### Table III

**Implicit Mapping Operators Supported.**

<table>
<thead>
<tr>
<th>Complex Element</th>
<th>Simple Type</th>
<th>Complex Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDT</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ACC</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ASMA</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MA</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**C. Model Management Operators**

Our general mapping procedure for document models is described in detail in [12], [13]. This mapping procedure comprises mapping heuristics and guidelines as well as mapping operators for both, the model and the metamodel layer. On the metamodel layer we establish mappings from XML Schema concepts to concepts of CCTS, i.e., \( \text{map}_{\text{XMLSchema} \rightarrow \text{CCTS}} = M_{\text{XMLSchema} \rightarrow \text{CCTS}} \). On the model layer we establish model mappings between any concrete XML Schema-based document and our CCTS based pivot model, i.e., \( \text{map}_{\text{XMLSchema} \rightarrow \text{CCTS}} = M_{\text{XMLSchema} \rightarrow \text{CCTS}} \) with \( M_{\text{CCTS}} = M_p \).

**Model mappings.** Table II summarizes the model mappings allowed during the mapping procedure by the user. On the contrary further implicit mappings are computed automatically as shown in Table III.

**Mapping cardinalities.** In general model mappings support a cardinality of \( 1 : 1 \), only. In fact this is sufficient in most cases. Deviations from this rule exist in terms of mapping functions and whenever a BCC in the CCL is of a pure generic nature. In the latter case explicit mapping functions and whenever a BCC in the CCL is allowed during the mapping procedure by the user. On the contrary further implicit mappings are computed automatically as shown in Table III.

**Mapping functions.** For simple elements we specify two commonly used mapping functions, which are \( \text{split}(\text{input1, output1, output2, ...}) \) and \( \text{merge}(\text{input1, input2, ...}, \text{output}) \). In both cases the output are one or more BCCs in the pivot model \( M_p \). Tokens and or expressions on how to split or merge are neglected as we do not deal with instances and their transformations right now. At the moment we only transform schemas.
V. IMPLEMENTATION

A. Framework

We developed a framework for document management, covering conceptual aspects as well as implementation details. A first impression on this framework was already envisioned in [19]. An overview is depicted in Figure 3. This framework has been entitled VIENNA Add-In and may be downloaded from [20]. The VIENNA Add-In is an open source plug-in for the UML modeling tool Enterprise Architect that supports all steps from designing to implementing a document model. To guide the user through this process several utilities and wizards are available. The implemented components rely on various standards originating from UN/CEFACT such as the CCTS. In order to support reverse engineering and transformation of business document models we currently depend on mapping models defined in MapForce. The supported modeling and integration scenarios are described in more detail in subsequent sections.

B. Modeling

Through so-called MDG technologies supported by the Enterprise Architect we implemented the UPCC specification from UN/CEFACT. This profile specification allows the user to model CCTS conforming models within the Enterprise Architect modeling environment. Thereby, this tool allows to model business documents in three different ways:

- Model from scratch and define your own document building blocks, see Figure 4, Mark 1. Business information entities are then derived by derivation by restriction. The modeled core components are put into a library for later reuse and sharing.
- Model by means of using predefined and standardized core components from the CCL and apply context to these building blocks, shown in Figure 4, Mark 2. The reuse of these standardized core components ensures interoperability because of a common semantic basis.
- Take an existing model and reuse those building blocks by means of reverse engineering, see Figure 4, Mark 3. Reverse engineering leads to transforming logical artifacts into conceptual models. This transformation is described in Subsection V-D. From this conceptual model business information entities may be automatically generated to reflect the logical business document model specified by the underlying XML Schema.

Each of these modeling scenarios involves the creation of business information entities, which must be based on existing core components. The manual creation of these artifacts, including dependencies, is a time consuming task. Numerous wizards for creating contextualized business information entities in our add-in take care of derivation by restriction and dependencies.

C. Subsetting

There are at least two ways of doing subsetting and transformation on the metamodel layer when using on the one hand a conceptual pivot model \( M_P \equiv M_{CCL} \) and on the other hand a logical model \( M_A \), based on XML Schema. In the following operations we treat the terms schema and model equally, i.e., CCL in our case. Note that the model \( M_{CCL(A)} \),
is a subset $M_{P'}$ of the pivot model $M_P$. After modeling we obtain $M_{CCL(A')}$ as a subset of $M_{CCL(A)}$ and may compute the difference

$$\text{Diff}(M_{CCL(A)}, \text{Match}(M_{CCL(A)}, M_{CCL(A')}))$$  \hspace{1cm} (1)$$

between them. The complement is then omitted in the generated XML Schema file.

**NDR based.** Figure 5 illustrates the first approach in which Naming and Design Rules (NDR) of a specific document standard are the central artifact in the schema generation process. A conceptual model $M_{CCL(A')}$, which is a subset of the original model $M_A$, is read by a generator and serialized into an XML Schema using for example the NDR of CCTS. Our VIENNA Add-In provides such a schema generator implementing the NDRs of UN/CEFACT. The cost of implementing specific NDRs is considerable, though.

**XML Schema file based.** In contrast to the NDR-based schema generation approach Figure 6 shows an alternate method, that uses the original schema $A$ as basis for the generation of schema $A'$. Steps 1 and 2 are the same as before. However, in Step 3 a generator component loads schema $A$ as a copy into memory and computes the union of schema $A$ and model $M_{CCL(A')}$, i.e.,

$$\text{Merge}(M_A, M_{CCL(A')}, \text{map}_{AP})$$  \hspace{1cm} (2)$$

The union remains unchanged in the schema, but everything else is deleted or modified and saved as schema $A'$. This approach is of generic nature and may deal with any XML schema that comes across. We implemented this schema based generation method as part of the transformer within the VIENNA Add-In, too.

**D. Transforming**

To transform two different types of schema-based documents models with the VIENNA Add-In a 2-Step process has to be carried out. The components needed for a full transformation from models $M_A$ to $M_B$ are the *Importer* and the *Transformer*.

**First Step.** Carry out the following mapping operations using the MapForce tool:

$$\text{map}_{AP} = M_A \leftrightarrow M_{CCL}$$  \hspace{1cm} (3)$$

These mappings enable to use the generic XML Schema Importer component of the VIENNA Add-In. The Importer performs the transformation operations

$$M_{CCL(A)} = t(M_A, M_{CCL}, \text{map}_{AP})$$  \hspace{1cm} (5)$$

$$M_{CCL(B)} = t(M_B, M_{CCL}, \text{map}_{BP})$$  \hspace{1cm} (6)$$

to obtain CCTS/CCL conforming representations of models $M_A$ and $M_B$, respectively.

**Second Step.** Both models are now ready to compare and we may compute the union of models $M_{CCL(A)}$ and $M_{CCL(B)}$:

$$\text{Merge}(M_{CCL(A)}, M_{CCL(B)}, \text{map}_{CCL(A)CCL(B)})$$  \hspace{1cm} (7)$$

Note, that $\text{map}_{CCL(A)CCL(B)}$ is retrieved by simple name matching and does not require any manual tasks. The result of the merge operation may be seen as a subset $M_{CCL(A')}$ or $M_{CCL(B')}$ depending on the direction of transformation. Having computed the merge of the conceptual models we finish this transformation step with a subsetting step described in the *XML Schema file based* paragraph, which yields the final XML schema. The transformation operation

$$M_B/M_A = t(M_A, M_B, \text{map}_{AB})$$  \hspace{1cm} (8)$$

is then complete. We assume that the transformation yields only a subset of the original model, thereby referring to $M_B/M_A$. In case no loss of information occurs, $M_B/M_A$ equals $M_B/M_A$. The general picture of this transformation process is depicted in Figure 7.

**VI. Case Study**

In previous sections of this paper we introduced a framework for mapping standards, based on XML Schema, such as the Universal Business Language (UBL), to Core Components (CC) contained in the Core Components Library (CCL). We evaluated our framework using two different standards based on XML Schema. In particular, we evaluated the mapping of ebInterface, a local Austrian standard for electronic invoicing [21], as well as the UBL Invoice to Core Components. Successfully mapping both standards to CC enables us to perform
a transformation from one business document standard to another, which is also part of our evaluation. We investigated the transformation of ebInterface to UBL. An overview of our evaluation is provided in Figure 7. The results of both, mapping as well as transformation, are presented in Table IV and Table V, respectively. In the following, the results are discussed in more detail.

A. Assumptions

Before presenting the results it is necessary to discuss assumptions made during the actual case study. Earlier in this paper we presented that a mapping is created through a user as part of a manual mapping process using Altova MapForce. Based on these manual (explicit) mappings, the importer derives implicit mappings. However, for evaluating the success of a mapping process only the explicit mappings are examined. The reason for doing so is because in the mapping process a user only maps elements of a business document standard, which are designed to contain the actual data, referred to as data elements or elements. Structural information maintained through complex types, e.g., a Person has an Address, is implicitly mapped and therefore derived automatically by the tool. Simply speaking, we only count the leaf nodes of the XML Schema. The way we count mappings also considers the schema design pattern applied. Both, ebInterface and UBL are built using Garden of Eden. Counting leave nodes then is a matter of counting referenced simple type elements within global complex types.

B. Mapping ebInterface to CCL

One of the standards chosen to be mapped to the CCL, is ebInterface. The process of the performed mapping is illustrated in Figure 7, Mark A. Overall, the results of mapping ebInterface to CCL are presented in the first row of Table IV. The table shows that ebInterface contains 130 data elements. Out of these 130 elements, 76 have been explicitly mapped to the CCL, which represents the number of mappings created through the user as part of the manual mapping process. However, 54 data elements could not be mapped to the CCL due to different reasons, such as the extensive use of xsi:type, which is currently not supported in our mapping framework. Having these numbers at hand allows us to estimate a ratio expressing the success of mapping ebInterface to the CCL, which is 58.46% in the current case. Therefore, in the case of ebInterface, the mapping process illustrated in Figure 7, Mark 1, results in a loss of information of 41.54%. As mentioned above, the reason for loss of information is due to unsupported concepts in our mapping framework. Although the number of unsupported concepts remains low, the application re-use of concepts within the ebInterface XML schema makes this number appear rather high.

C. Mapping UBL to CCL

The second standard that we evaluated the mapping to the CCL with, is the Invoice document of the Universal Business Language (UBL), which corresponds to Mark B, in Figure 7. In particular, we focused on mapping the UBL Invoice to Core Components. Overall, UBL contains 952 different data elements. Out of these concepts, 348 elements were explicitly mapped to Core Components. On the contrary, 604 data elements remained unmapped. However, it is important to mention that the UBL Invoice only utilizes parts of the overall UBL standard. Hence, the success ratio of 35.55% seems low, while the Invoice is almost fully mapped to Core Components. Accordingly, the loss of information through the mapping process, as illustrated in Figure 7, Mark 1, is 64.45%.

D. Transforming ebInterface to UBL

Mapping both business document standards enables us to transform one business document standard to another. In this evaluation we transformed ebInterface to the Universal Business Language (UBL). Transforming UBL to ebInterface doesn’t make sense and is therefore not evaluated. The reason it doesn’t make sense is because the concepts and semantics expressed in UBL, are by far not covered in ebInterface. However, on the contrary almost all concepts and semantics of ebInterface may be expressed in UBL. In the following, the results of transforming ebInterface to UBL, summarized in Table V, are presented. The input to the transformation, as illustrated in Figure 7, Mark C, are the Core Component models resulting from the import process described above. Based on the two models, as well as the mapping of both standards to the Core Component Library, a transformation is possible. The outcome of the transformation is an XML schema, conforming to UBL, but expressing the semantics...
of ebInterface. Accordingly, we are interested in the data elements maintained after the import process. In particular, 91 data elements have been preserved in the import process. After transforming ebInterface to UBL, 57 data elements have been preserved. Overall, this results in a loss of information of 43.85% through the transformation process, which is indicated in Figure 7, Mark 2.

VII. CRITICAL DISCUSSION AND FUTURE WORK

Overall, the evaluation shows that we are able to achieve reasonably good results in mapping business document standards to Core Components. It should also be highlighted, that the manual mapping effort for the user is greatly supported through the automatically derived mappings. However, different reasons such as the unavailability of particular concepts in the Core Component Library and insufficient mapping operators inhibit complete mappings of business document standards to Core Components. These deficiencies result in a loss of information, indicated in Figure 7, Mark 1. In addition, one may observe that along with the transformation, another kind of loss of information occurs, indicated in Figure 7, Mark 2. Therefore, the success of a mapping process and a subsequent transformation is influenced through the loss of information occurring in the mapping as well as transformation. Applied to ebInterface this means that, after mapping the standard to Core Components and transforming the resulting model to UBL, 43.85% of the original structure and semantics of ebInterface are maintained. Currently, our tool does support various modeling and transformation tasks as is necessary for model management. However, the registration and retrieval of modeling artifacts from a repository is missing. Also, we do not support model versioning, which plays an important role in the evolution and management of business document models. Future work will therefore be directed towards these missing components for our document model management framework. Other directions will consider the improvement of the existing components. As already mentioned above, there is considerable amount of information that is lost during transformation. In practice such low rates of success in transformation are not acceptable. One missing property in a business document may cause business service interfaces to fail. While we cannot cope with missing concepts in either of the involved schemas, we may improve our approach and methods. We believe that additional and improved tool support can prevent the information loss, considerably. Consequently, we will revamp our mapping model and Importer component to support more complex mapping scenarios. To minimize the amount of work in the mapping process we will further investigate automatic matching to create an initial set of mappings.

REFERENCES