An Evaluation of Mapping Strategies for Core Components

Christian Eis, Philipp Liegl, Christian Pichler and Michael Strommer
Research Studios Austria, Vienna, Austria
{*firstname.lastname*}*@researchstudio.at*

***Abstract*—Electronic data interchange (EDI) is inevitable in enabling successful collaborations between different business partners. Exchanging information electronically requires standardized formats for information exchange. There exist a variety of standards including bottom-up standards as well as top-down standards. However, business partners may utilize different standards resulting in a loss of interoperability. To cope with the variety of standardized formats we propose the approach of mapping these formats to the common concept of core components introduced by the United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT). We evaluate the applicability of different strategies for mapping an arbitrary XML Schema based standard to core components by the example of ebInterface, an Austrian invoicing standard. The evaluation provides evidence that mapping of arbitrary standards to core components indeed provides substantial benefit in leveraging the interoperability between different business document standards.**

I. Motivation and Introduction

One of the major obstacles towards a seamless integration of different processes across company boundaries is the heterogeneity of data. Although technologies such as Simple Object Access Protocol (SOAP) and paradigms such as Service Oriented Architectures (SOA) provide technological solutions for interoperability, data heterogeneity still remains an unsolved issue.

In recent years a significant trend from delimiter based EDIFACT standards [1] towards XML based solutions [2] was observed. Although XML brought a more convenient and contemporary representation format, interoperability issues stayed the same. First, several different standardization organizations define their own, proprietary XML standard definitions. The different initiatives are not aligned with each other and are thus mostly incompatible. Second, there are two major business document standard families: top-down standards and bottom-up standards. In a top-down approach many different requirements of the involved stakeholders are included in the final standard. The resulting standard definition is a superset of all different requirements. Such approaches tend to be complex and difficult to implement, in particular for small and medium-sized enterprises (SME). In contrast, bottom-up standard definitions try to find a subset of the most important requirements of all involved stakeholders. If business partners require additional elements or attributes, which are not reflected in the core standard definition, well defined extension points in the standard are used. Using the extension points, domain specific amendments may be made to the core standard definition, without altering the overall compatibility to the core standard definition. Thus, we define mismatching and incompatible business document standards as the major obstacle towards a seamless information integration across company boundaries. A solution for one global standardization initiative is provided by the core components initiative, founded by the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT). In a nutshell, core components are reusable building blocks for assembling business document definitions on a conceptual level. The conceptual business document representations may then be used to derive deployment artifacts such as XML Schema. The major challenge remains in mapping existing business document definitions to a common core component model. In this paper we evaluate the applicability of the core component approach by mapping ebInterface, an Austrian invoicing standard, to a common core component model. Thereby, we identify a set of challenges and provide pertinent solutions.

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).

II. Introduction to ebInterface

In the following we introduce the XML Schema based standard ebInterface [3], which is an Austrian standard for representing invoices electronically. Furthermore, we provide an accompanying example from the telecommunication industry. We then use the example throughout the article to evaluate the applicability of the different strategies for mapping the ebInterface standard to the concept of core components.

Figure 1 illustrates an excerpt from the main structure specified through the ebInterface standard including elements such as `InvoiceDate`, `Biller`, `Details`, and `PaymentMethod`. In the following we will elaborate on the elements `Biller` and `PaymentMethod` in more detail.

The element `Biller` is used for representing information about the party, such as a particular company, issuing an invoice. A more detailed view of the element `Biller` is provided in Figure 2. As illustrated in Figure 2, the element `Biller` contains further elements such as `VATIdentificationNumber`.

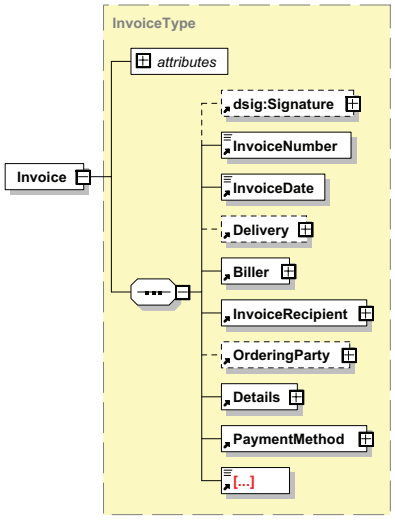

Figure 1: An excerpt of the ebInterface standard.

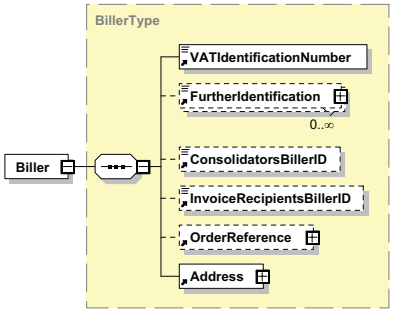

Figure 2: An excerpt of the biller section of the ebInterface standard.

`VATIdentificationNumber` represents a unique tax identification number of the party issuing the invoice. In the telecommunication industry, service providers typically issue their customers invoices on a regular basis. Applied to the example, the element `Biller` would be used to represent information about the provider, such as the provider's unique tax identification number.

Furthermore, the element `Biller` contains another element named `Address`, which is used to describe address information of the party issuing the invoice. The element `Address` is illustrated in more detail in Figure 3 and contains further elements such as `Street`, `Town`, and `Country`. Applied to our example from the telecommunication industry, the element `Address` would be used to provide the postal address of the service provider.

The element `PaymentMethod`, also illustrated in Figure 1, may be used to represent information regarding the payment of the invoice. We distinguish between three different kinds of payments including universal bank transactions, direct debit, and no payment at all. Applied to the example in the telecommunication industry, the payment method would be used to specify how a customer receiving the invoice is

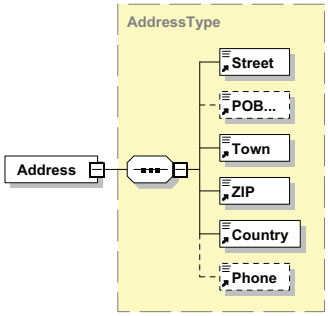

Figure 3: An excerpt of the address section of the ebInterface standard.

paying for the invoice. All other elements of the standard are not discussed any further, but can be found in the ebInterface specification [3].

III. UN/CEFACT's Core Components at a Glance

Our approach is based on the Core Components Technical Specification (CCTS) [4] maintained by UN/CEFACT. In a nutshell, core components are reusable building blocks for assembling business document definitions on a conceptual, implementation neutral level. Figure 4 gives an overview of Core Components Technical Specification's basic concepts. 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 given core component to the specific needs of a given business domain, the concept of business information entities (BIE) is used. Business information entities are derived from core components by restriction and may be used in a given business domain. Thus, a business information entity must not contain any attributes, which have not been defined in the underlying core component definition. Essentially, core components represent the common semantic basis for all business information entities. In a nutshell, business information entities are the same as core components except that they i) use a different naming convention in order to facilitate distinction from core components, ii) are used in a specific context, and iii) are based on core components and thus restrict an existing core component definition.

Having these concepts at hand, it is possible to model business documents first in a conceptual, syntax neutral, and platform-independent way using core components. The created core component libraries may then be used to assemble different libraries of business information entities. Due to the derivation-by-restriction concept, all business information entities are based on a common base of core components. Therefore, interoperability between different business information entities is always provided, as long as the link to the underlying core component definition

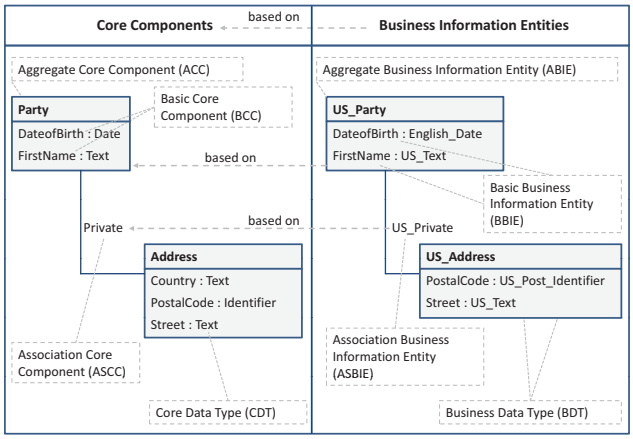

Figure 4: Overview of basic core component concepts.

exists. The conceptual business document models may then be used in order to derive deployment artifacts such as XML Schema.

Core Components.: The basic concept behind core components is the identification of objects and object properties. Thereby, object properties are divided into two subgroups: 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. On the left hand side of Figure 4 the ACC `Party` includes two simple object properties - `DateOfBirth` and `FirstName`. 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 `DateOfBirth` is `Date`. 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 is used (ASCC). In Figure 4 the ACC `Party` is connected to the ACC `Address` using the association core component (ASCC) `Private`. Thus, `Address` is a complex object property of the ACC `Party` and indicates the fact, that a party has a private address.

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. On top of Figure 4, a *based on* dependency between the business information entity artifacts on the right hand side and the core component artifacts on the left hand side underlines this strong relation. Due to this relationship, the underlying core component semantics of any business information entity may be retrieved easily.

Similar to the concept of 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 depicted using the concept of basic business information entities (BBIE). On the right hand side of Figure 4 the ABIE `US_Address` aggregates the two BBIEs `PostalCode` and `Street`. Please note, that the business information entity `US_Address` does not contain all attributes of the underlying core component `Address`. Thus, the ABIE restricts the underlying ACC. The value domain for a given basic business information entity is defined using the concept of business data types (BDT). The BBIE `PostalCode`, contained in the ABIE `US_Address` is of type `US_Post_Identifier`. Dependencies between different business information entities are shown using the concept of association business information entities (ASBIE). Please note, that a business information entity must contain the name of the underlying core component. However, so called qualifiers may be used in order to facilitate the distinction between core components and business information entities. E.g the qualifier `US_` is used in Figure 4 for all business information entity definitions. For a detailed discussion on core components and business information entities see [5].

A similar relationship as it is 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. Because we do not elaborate on data type mappings in this paper we do not further dwell on these concepts.

Since it is imperative that all business information entities are based on a core component, a consolidated library of predefined core components must be provided. A global library of core components is currently developed and maintained by UN/CEFACT and has become known as Core Component Library (UN/CCL) [6]. 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. The core component definitions in the Core Component Library represent the common semantic basis of all business information entities exchanged among business partners. 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 entitites are provided per different domain rather than on a global

level.

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 have created a UML profile, that allows using core components with standard UML modeling tools. Today the standard has become known as the UML Profile for Core Components (UPCC) [7]. The UPCC fully represents all necessary concepts from the Core Component Technical Specification, including business information entities and data types. We will use the UML representation of Core Components in this paper in order to elaborate our different mapping approaches.

IV. Related Work

In the field of business document engineering and integration we have identified three distinct kinds of related work. First, there is work concerning Web-Services and business data exchange aiming at semantic interoperability [8], [9] that perfectly relates to our very goal of business document integration and application domain. Second, we use concepts and tools originating from the database schema and model mapping field. There is significant work concerning mapping languages and mapping tools closely related to our mapping strategies [10], [11]. And last we perform semantic alignment, first between two schemas and then between the core components and the business information entity layer, which relates to the field of ontology and schema matching. See for example Rahm et al. [12].

V. Mapping by Example

In the following we present our findings of a case study on mapping ebInterface to Core Components, as defined in the Core Component Library (CCL). These findings may contribute to the general aim of leveraging data exchange within heterogeneous information systems. To get an understanding of our notion of mappings and mapping operators we refere the interested reader to [13]. Generally, we define a mapping model map_{AB}, which aligns models M_A and M_B, such that a transformation may be executed in both directions taking the two models and the mapping model as input. In the case of aligning arbitrary XML Schema based models to CCTS, i.e. the standardized Core Component Library (CCL), we employ a more specific form of model transformation, where CCTS is not only some document model, but also a pivot model M_P. This pivot model is then involved in every single mapping task, to reduce the amount of mapping definitions. For a detailed description on our mapping and transformation scenarios see [14]. We also differ from the general model transformation description above in terms of the introduction of additional concepts and semantics to our mapping models apart from simple $1:1$, $1:n$ and $n:m$ mappings. These mapping constructs will be introduced successively and described by example in the following subsections.

The general mapping heuristic [15] for mapping arbitrary XML Schema models to both, core components and business information entities, comprises the following steps:
1. (a) Find an ACC, which includes most of the concepts modeled within a complex type. (b) Use the provided mapping operators to support Step 2 and hence ease the generation of XSD files needed for a full round-trip, i.e., the support of bidirectional transformations.
2. Create a corresponding ABIE, which restricts on those BBIEs needed.
3. In case 1. and 2. cannot be fully applied, a set of mapping heuristics and strategies may be considered.

Most of the concepts applied in this work have been defined in [15]. For the integration of ebInterface and CCL we introduce some new XSD to CCTS concept mappings like `xs:group`. There are three types of basic alignment scenarios that may occur. Concepts may be mapped fully equivalent, partially/conditionally equivalent, or not at all. For example, we perfectly map the concept of `Address` in our CCL model to the `Address` concept within the `Biller` composite in ebInterface. This mapping comprises the container only and may not be automatically applied to the sub-elements. For any sub-elements additional mappings have to be derived or specified. A partial equivalent mapping is for example defined for `ConsolidatorBillerID` $\leftrightarrow$ `Identification`. This mapping is only partially correct as `Identification.Identifier` is too general to express a `ConsolidatorBillerID` in the beginning, unless we put it into some business context. This issue will be discussed in more detail later in this section. Special care has to be taken of mappings like `VATIdentificationNumber` $\leftrightarrow$ `NULL`, where no corresponding concepts exist on one side.

To implement our mapping approach we base our work upon the VIENNA Add-In [16], which is an open source modeling tool for business documents and processes. As additional tool support we use the XML Schema mapping tool Map Force from Altova, that provides a smooth user interface for mapping schemas along with the ability to define custom mapping functions and operators.

In the following we firstly illustrate how XML Schema concepts not covered by the UML Profile for Core Components (UPCC) may be coped with during the mapping process involving ebInterface and CCTS. Secondly, we show how mappings on the core components level may impact the business information entity layer. Thirdly, we come up with two strategies to introduce new concepts to the Core Component Library (CCL) in order to prevent a loss of information.

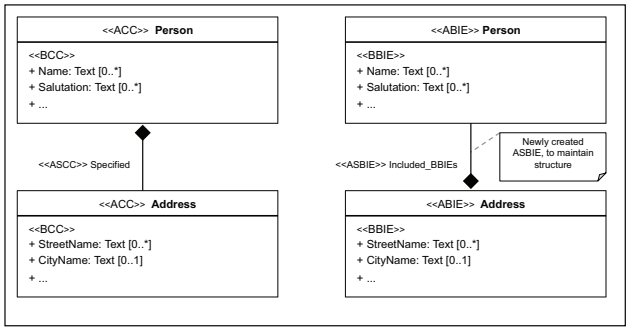

Figure 5: CCTS nesting issue.

A. Reusable Groups

sequence. Figure 7 shows an example of mapping a `sequence` to a *combination of ACCs* (see [15]). The `Biller`'s `Address` is mapped to the `SpecifiedAddress` of ACC `Party`. However, not all elements of the ebInterface `Address` can be mapped to ACC `Address`, because it does not contain semantically equivalent BCCs for some elements. Consequently, these elements must be mapped to other ACCs, such as `Person` and `Communication`.

This is also an example of *CCTS nesting*, because a new ASBIE must be created between `Address` and `Person`. The primary mapping of the ebInterface `Address` is to ACC `Address`, but we must include `Person` in `Address` in order to reflect the structure of ebInterface in the CCTS model (cf. Figure 5).

Figure 7 also contains a special case of mapping an element (`Street`) to two BCCs. For this purpose, we introduce a mapping function that splits the content of `Street` into appropriate parts that semantically match the target BCCs `BuildingNumber` and `StreetName`. While this function is not actually necessary on the conceptual level, it can later be used to generate stubs for instance transformation code.

Figure 8 is an example of mapping a `sequence` with *semi-semantic loss* (see [15]). Two source elements, `ConsolidatorBillerID` and `InvoiceRecipientsBillerID` are mapped to the same BCC `IdentificationIdentifier`. Since the BCC represents a more generic semantic concept than the source elements, it must be qualified in the CCTS model in order to preserve the semantic information. The resulting ABIE therefore contains two BBIEs, `ConsolidatorBillerID_Identification` and `InvoiceRecipientsBillerID_Identification` (see Figure 6).[1]

choice. When mapping a `choice` to CCTS, we need to preserve this structural information in the resulting CCTS model and therefore must include it in the mapping model.

[1]Note, that a BCC's type (in this case `Identifier`) is part of the BCC's name only in the XML schema, but not in the UPCC model.

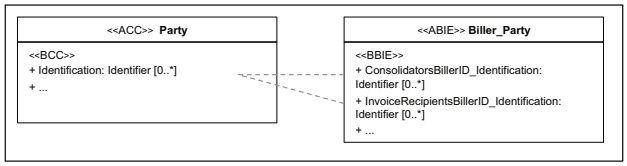

Figure 6: Semi-semantic loss issue.

Figure 9 shows the mapping of an invoice's delivery date to CCTS. The date can be specified either as a single `Date` or as a `Period`. In ebInterface this is defined as a `choice`, although the mapping definition does not explicitly show this information. We have defined the `xor` mapping function, which is applied to the mapping of each option of a `choice`. The function has an additional input parameter, namely an arbitrary ID used to a single `choice` to provide for the case, where a complex type contains more than one `choice`.

group. Model groups, such as `sequence` and `choice`, illustrated earlier are used to define a content model, which in turn may be used for defining complex types. The XML schema construct `group` allows the definition of content models separate from particular usages. The resulting advantage is, that a group of elements, once defined, may be reused in different complex type definitions. An example excerpt from the ebInterface standard, adapted for presentation purposes, is illustrated in Figure 11, letter A.

For content models, including `sequence`, `choice`, and `all`, it was already shown in the technical report [15], that an adequate model representation is feasible. Since model groups build upon these content models and the only extension of model groups is the separation of the content model from specific complex type definitions, an adequate model representation may be achieved similar to the representation for content models. In fact, creating the model representation consists of resolving the references from complex types to model groups and creating an appropriate model (see Figure 11, number 1).

However, when serializing the model representation to an XML schema not all structural information is preserved. Utilizing the approach proposed earlier, in the resulting XML schema all complex type definitions would have the content models incorporated, instead of referencing the model group. Therefore, we propose the introduction of an additional tagged value for each BBIE that an element of a particular model group is mapped to. In particular we introduce a tagged value named *modelGroup* which contains the name of the model group that the element mapped to the particular BBIE belongs to. Utilizing the proposed mapping strategy, the element declarations within the complex type definition are represented through BBIEs (see Figure 11, mark 1). The elements within the model group definition are represented as BBIEs as well, whereas for the element declarations from the modelGroup the tagged value *modelGroup* is used (see

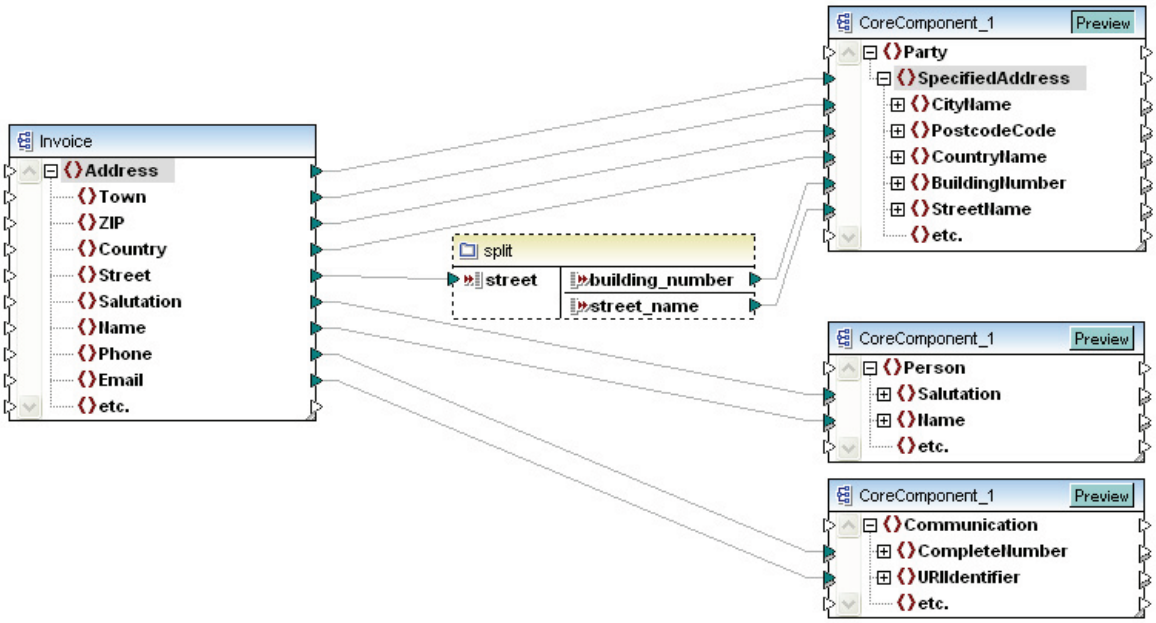

Figure 7: Mapping a sequence.

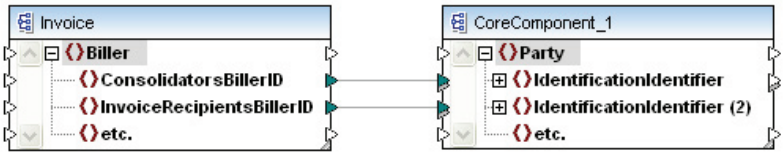

Figure 8: Mapping a sequence with semi-semantic loss.

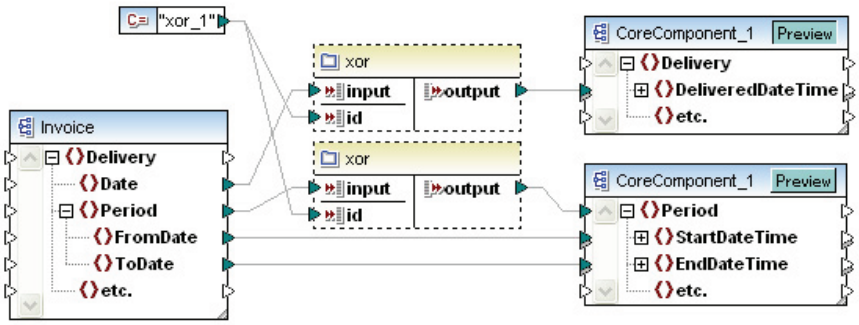

Figure 9: Mapping a choice.

Figure 10: Mapping a group.

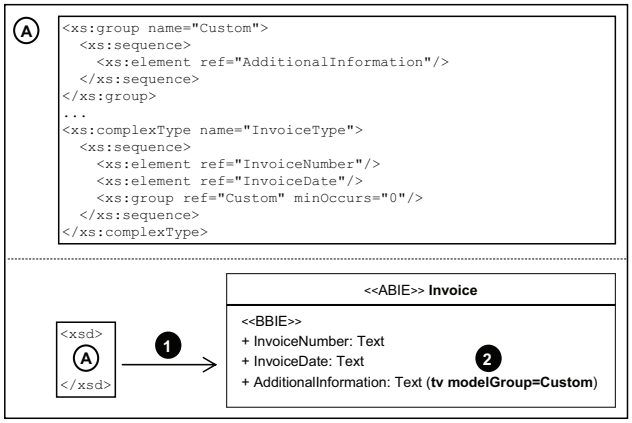

Figure 11: Reusable Group *group*

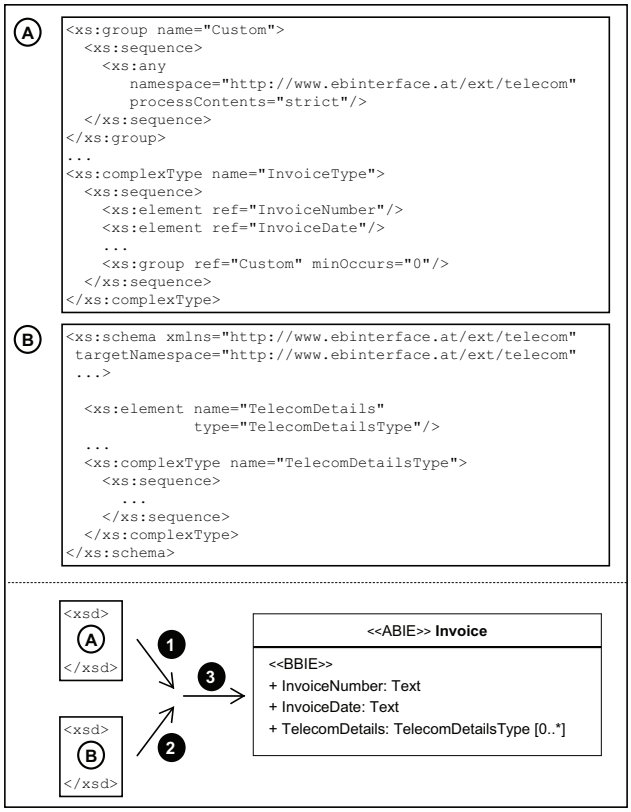

Figure 12: Element *xs:any*.

Figure 11, mark 2).

At the bottom of Figure 10, a mapping model for the `group` from Figure 11 is shown. The `Invoice`'s element `AdditionalInformation` is mapped to the BCC `Description` of ACC `Document`. The `group` name is provided as an annotation of the mapping so that an appropriate tagged value may be created in the CCTS model.

```
<xs:group name="Custom">
  <xs:sequence>
    <xs:any namespace="##other" processContents="lax"/>
  </xs:sequence>
</xs:group>
```

Figure 13: Excerpt from the ebInterface standard.

B. *xs:any Element*

The `xs:any` construct in XML Schema is used for declaring placeholders in an XML schema, which in turn adds greater flexibility in regard to the structure of the XML schema. As a consequence, stakeholders representing information according to an XML schema, which utilizes the `xs:any` construct, may encode any content at the location where the construct was used (see Figure 12, letter A). The location where custom content may be stored is from now on referred to as custom section. The only condition that the content in the custom section must fulfill, is that it is well-formed. Furthermore, the construct allows the use of certain attributes that may be used to further restrict the content in the custom section. In particular, these attributes allow restrictions in regard to the namespace that the content must belong to, as well as in regard to an XML schema that the structure of the content must comply with.

Therefore, for creating an adequate model representation, it is necessary to address the namespace restrictions as well as the content restrictions. For reasons of simplicity we do not address namespace restrictions, but focus on content restrictions in more detail. Restrictions on the content are achieved through the `processContents` attribute of the `xs:any` construct. The attribute may have one out of the following three values: `skip`, `lax`, and `strict`. The attribute value `skip` specifies that the content stored in the custom section may be any content that is well-formed. Furthermore, `lax` specifies that if an XML schema describing the content is available, then the content may be validated against the particular schema. At last, the attribute value `strict` specifies that an XML schema must be available, which describes the content stored in the custom section.

To summarize, the only case where the structure in the custom section is foreseeable is the third case where the attribute `processContents` is set to `strict`. Therefore, the third case is also the only case where an adequate mapping strategy of the `xs:any` construct to core components is feasible. The mapping strategy in particular means that based on the XML schema utilizing the `xs:any` construct (see Figure 12, Letter A) and the XML schema providing an XML schema for the content stored in the custom section (see Figure 12, Letter B) it is possible to create a model representation.

Figure 13, shows an excerpt from the ebInterface standard where the `xs:any` construct is used. However, an adequate model representation for the `xs:any` construct cannot be created, since no XML schema for the structure of the

custom section is provided nor enforced through the attribute `processContents`. Therefore, it is not feasible to create an adequate model representation for the `xs:any` construct used in the ebInterface standard.

C. xsi:type Element

Complex types are used for defining types that contain other elements and attributes, while utilizing reusable groups. When defining an XML schema these complex type definitions may be used to typify element declarations. An example excerpt from the ebInterface standard demonstrating the definition of a complex type as well as an element declaration utilizing the complex type definition, is illustrated in Figure 14, mark 1 and 2.

Furthermore, the concept of derivation can be applied to complex types meaning that new complex types may be derived from existing complex types by extension (see Figure 14, mark 3). These derived types may then be used just as any other complex type. The consequence of the elements typified through a certain complex type is that elements in the actual XML document instances must adhere to the structure specified through the complex type. However, the `xsi:type` construct allows the substitution of the complex type typifying an element through any of it's derived complex types in the XML document instances.

For these reasons, creating an adequate model representation of the `xsi:type` construct is not feasible, which is discussed in more detail in the following. Basically, the concept of derivation may be realized in core components as it is described for the concept of substitution groups in [15]. However, core components support the concept of derivation by restriction only, whereas deriving complex types is achieved through derivation by extension. In addition, the concept of `xsi:type` is a construct in the XML Schema instance namespace, meaning that it is used, as mentioned earlier, in actual XML document instances for referencing a derived type. However, the mapping strategies presented in the technical report as well as in this paper are created for a mapping on the conceptual level. Therefore, a mapping of the `xsi:type` construct to core components is currently not feasible.

An example excerpt from the ebInterface standard, building upon the `xsi:type` construct, is illustrated in Figure 14. The standard defines an element named `PaymentMethod` utilizing the complex type `PaymentMethodType` (cf. Figure 14, mark 2). In addition, the complex type serves as a basis, where three additional complex type definitions are derived from, which includes the `UniversalBankTransactionType` (cf. Figure 14, mark 3), the `DirectDebitType`, and the `NoPaymentType`.

For the element `PaymentMethod`, the standard specifies that in the actual XML document instances, a derived type from the complex type `PaymentMethod` is to be used. An

```
<xs:complexType name="PaymentMethodType"> 1
  <xs:sequence>
    <xs:element ref="Comment" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
...
<xs:element name="PaymentMethod" 2
            type="PaymentMethodType"/>
...
<xs:complexType name="DirectDebitType">
  <xs:complexContent>
    <xs:extension base="PaymentMethodType">
      ...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="NoPaymentType">
  <xs:complexContent>
    <xs:extension base="PaymentMethodType">
      ...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
...
<xs:complexType name="InvoiceType">
  <xs:sequence>
    <xs:element ref="InvoiceNumber"/>
    <xs:element ref="InvoiceDate"/>
    ...
    <xs:element ref="PaymentMethod" minOccurs="0"/>
    ...
  </xs:sequence>
  ...
</xs:complexType>
...
<xs:complexType name="UniversalBankTransactionType">
  <xs:complexContent>
    <xs:extension base="PaymentMethodType"> 3
      <xs:sequence>
        <xs:element ref="BeneficiaryAccount" .../>
        <xs:element ref="PaymentReference" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute ref="ConsolidatorPayable" .../>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 14: Element *xsi:type*.

```
<eb:PaymentMethod xsi:type="eb:UniversalBankTransactionType">
  <eb:BeneficiaryAccount>
    <eb:BankName>Bank Name</eb:BankName>
    <eb:BankCode eb:BankCodeType="AT">99999</eb:BankCode>
    <eb:BIC>HHAUTKWZ</eb:BIC>
    <eb:BankAccountNr>11111111111</eb:BankAccountNr>
    <eb:IBAN>AT499999911111111111</eb:IBAN>
     <eb:BankAccountOwner>Max M</eb:BankAccountOwner>
  </eb:BeneficiaryAccount>
  ...
</eb:PaymentMethod>
```

Figure 15: Excerpt from XML document instance utilizing the `xsi:type` construct.

example from an XML document instance, utilizing the `xsi:type` construct is illustrated in Figure 15.

However, due to the reasons exemplified above, a mapping for the derived complex types as well as the `xsi:type` construct is not feasible. Therefore, an adequate model representation of the `xsi:type` construct used in the ebInterface standard cannot be created.

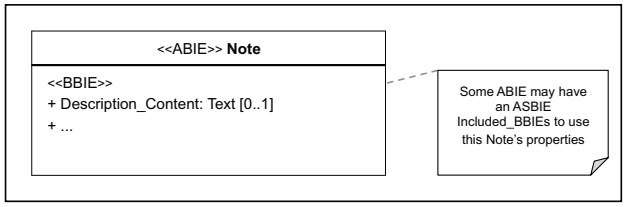

Figure 16: Generic content container issue.

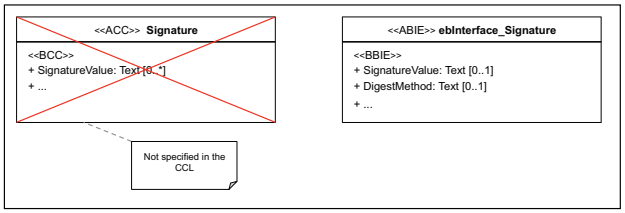

Figure 17: Negative mapping issue.

D. Further Mapping Issues

Generic Content Containers.: In the course of mapping ebInterface to the CCL we have sometimes not been able to map common descriptions or additional information of a complex type definition to the corresponding ACC. Therefore, in case of missing BCCs we recommend to map elements to generic BCCs of some generic ACC. As an example we mapped the ebInterface element `Description` to the BCC `Content` of ACC `Note` resulting in a corresponding BBIE `Description_Content` as shown in Figure 16.

Negative Mappings: Sometimes one may find elements or complex types in an XML Schema, which have not even in the broadest sense a counterpart in the more or less comprehensive CCL. In such cases we encounter a so called negative mapping. The solution proposed is to create an entirely new ABIE with proper BBIEs in the DOCLibrary [7] without the obligatory *based on* dependency. Note, that this option should only be used as a last resort as the advantage of a predefined semantic is completely lost. As an example for such a negative mapping consider Figure 17. In the CCL there is no adequate ACC, which may be mapped to the ebInterface concept of a digital signature with all its element declarations. As a result, we modeled an ABIE named `ebInterface_Signature` in the DOCLibrary, where the CCTS conforming ebInterface `Invoice` document is assembled.

VI. Conclusion

Throughout this paper, different strategies have been introduced for mapping arbitrary XML Schema constructs to core components. Moreover, the mapping strategies have been illustrated using the Austrian invoicing standard ebInterface. Furthermore, we evaluated the different mapping strategies in regard to their usability. The results of the evaluation are illustrated in Table I. In particular all the XML Schema constructs are listed, that we provided a mapping strategy for. Accordingly, the Table illustrates the core components concept that the XML Schema construct has been mapped to, as well as if necessary, an extension enabling successful mapping. Furthermore, the usability of the different mapping strategies is illustrated.

XSD Concept	CCTS Concept	Tagged Value	Usability
xs:all	BCC/BBIE	modelgroup = all	+
xs:any	ABIE/BBIE	-	-
xs:choice	BCC/BBIE	modelgroup = choice	+
xs:group	BCC/BBIE	modelgroup = custom	+
xs:redefine	BCC/BBIE	-	o
xs:sequence	BCC/BBIE	-	++
xs:substitutionGroup	ACC/ABIE	super	-
xsi:type	-	-	- -

Table I: Overview on the Usability of the different Mapping Strategies

Also illustrated in Table I, the only XML Schema construct that cannot be mapped to core components is the `xsi:type` construct. Successful mapping of the construct is inhibited due to various reasons including the following. First of all, the `xsi:type` meta construct is used in actual XML document instances. Furthermore, the `xsi:type` construct cannot be comprehended unless having the ebInterface specification at hand. Also, the construct has in inherent derivation-by-extension mechanism which is not supported by core components.

On mapping aggregate core components (ACC) we may recognize two different strategies, if structural differences such as missing ASCCs prevail. First, we could choose an ACC, which for a given complex type covers most of the concepts. Later on we introduce a completely new association business information entity (ASBIE) on the BIE layer in order to maintain the structural requirements, when generating XML Schema files. Second, we could choose another ACC, which covers some or none of the basic core components required, but maintains the structure naturally because of proper ASCCs. There may however be a loss in semantics for the ACC.

References

[1] J. Berge, *The EDIFACT Standards*, 2nd ed. Cambridge, MA, USA: Blackwell Publishers, 1994.

[2] H. Li, "XML and Industrial Standards for Electronic Commerce," *Knowledge and Information Systems*, vol. 2, no. 4, pp. 487–497, 2000.

[3] AustriaPro, *ebInterface 3.0*, http://www.ebinterface.at, 2008.

[4] UN/CEFACT, *Core Components Technical Specification 3.0*, http://www.untmg.org/ccts/spec/3_0, 2009.

[5] P. Liegl, "Conceptual Business Document Modeling using UN/CEFACT's Core Components," in *Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009), January 20-23, Wellington, New Zealand*, 2009, pp. 59–69.

[6] UN/CEFACT, *UN/CEFACT's Core Component Library UN/CCL*, http://www.unece.org/cefact/codesfortrade/codes_index.htm.

[7] UN/CEFACT, *UML Profile for Core Components Technical Specification 3.0*, http://www.untmg.org/upcc/spec/3_0, 2009.

[8] M. Nagarajan, K. Verma, A. P. Sheth, J. Miller, and J. Lathem, "Semantic Interoperability of Web Services - Challenges and Experiences," in *Proceedings of the IEEE International Conference on Web Services (ICWS '06), September 18-22, Chicago, IL, USA*, Washington, DC, USA, 2006, pp. 373–382.

[9] J. Guo and C. Sun, "Context Representation, Transformation and Comparison for Ad Hoc Product Data Exchange," in *Proceedings of the 2003 ACM symposium on Document Engineering (DocEng '03), November 20-22, Grenoble, France*, 2003, pp. 121–130.

[10] A. Blouin, O. Beaudoux, and S. Loiseau, "Malan: A Mapping Language for the Data Manipulation," in *Proceedings of the 2008 ACM Symposium on Document Engineering, September 16-19, Sao Paulo, Brazil*, 2008, pp. 66–75.

[11] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth, "Clio Grows Up: From Research Prototype to Industrial Tool," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD '05), June 13-16, Baltimore, MD, USA*, 2005, pp. 805–810.

[12] E. Rahm and P. A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[13] P. A. Bernstein, "Applying model management to classical meta data problems," in *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003), January 5-8, Asilomar, CA, USA*, 2003, pp. 227 – 232.

[14] C. Eis, C. Huemer, P. Liegl, C. Pichler, and M. Strommer, "A Framework for Managing the Complexity of Business Document Integration," in *Proceedings of the 2009 eChallenges e-2009 Conference, October 21-23, Istanbul, Turkey. To be published*, 2009.

[15] P. Liegl, C. Pichler, and M. Strommer, *On Mapping Business Document Models to Core Components*, Research Studio Austria Forschungsgesellschaft mbH, http://code.google.com/p/vienna-add-in/mapping.pdf.

[16] *VIENNA Add-In - Visualizing Inter ENterprise Network Architectures*, Public Domain, http://code.google.com/p/vienna-add-in/.