A bottom up approach to build XML business document standards

Philipp Liegl, Christian Huemer
Vienna University of Technology
A-1040 Vienna, Austria
{liegl, huemer}@big.tuwien.ac.at

Christian Pichler
Research Studios Austria
A-1090 Vienna, Austria
christian.pichler@researchstudio.at

Abstract

XML has replaced traditional EDI standards in the field of business document standardization. Despite of the syntax, the principal approach to develop business document standards has not changed. A standardized business document is built by a superset of all elements that may appear in any business context, leading to overloaded and complex standards. However, in a particular partnership only a small percentage of the elements is used. This results in a top-down approach starting from a generic document and specifying partner-specific subsets. Such an approach is too costly for small and medium-sized enterprises (SME), because agreements on subsets must be implemented in their software systems. As an alternative we suggest a bottom-up solution that starts from a core set of elements, representing the intersection of all industry contexts. Thereby, the core set may be extended to incorporate the needs of a specific business context. In this paper we examine different mechanisms provided by XML Schema to realize such an extension. The applicability of the different mechanisms is evaluated by means of the Austrian e-Invoicing standard ebInterface, which we co-authored.

1. Motivation

Exchanging business documents by electronic means has been around in the IT domain for decades. One of the first and probably most successful non-XML business document standards is UN/EDIFACT [14]. The goal of EDIFACT was to create top-down business documents, allowing cross-country and cross-industry message exchanges. In order to meet the requirements of different industries and application domains, the document standards were designed as generic as possible with numerous conditional elements. With the introduction of XML new business document standards were introduced [7] [6] [9]. Although the syntax changed from a delimiter-based one to a mark-up one, the principal approach to develop business document standards did not change. More or less all dominant XML business document standards define a single invoice, purchase order, etc. The resulting generic business document type includes the superset of all elements that may be used in any business context. This leads to overloaded and complex business document standards.

In contrary to small and medium-sized enterprises (SME), large enterprises are able to implement their own ERP software or at least to customize their existing ERP software in order to handle these kind of business document standards and their bi-lateral agreements. Most of the SMEs do not have this flexibility, but rely on low cost commercial-off-the-shelf-software (COTS). In order for SMEs to participate in B2B scenarios the COTS systems should also allow the seamless import and export of business documents. Therefore, vendors of COTS have to provide appropriate import/export interfaces. However, they cannot foresee and implement partner-specific requirements. Thus, they require core business document standards covering only those elements that are common to all industries. If numerous COTS vendors provide interfaces for a core business document standard the users of these COTS systems may exchange business documents on the fly. By definition the core standard cannot consider specific elements required by a certain industry. To overcome this restriction a flexible extension mechanism for controlled domain-specific amendments is required. Depending on the target customer base, a COTS vendor may choose to implement certain domain-specific extension, but not necessarily all.

We followed the idea of a core business document standard plus domain-specific extension when starting the ebInterface initiative [1] for the Austrian Chamber of Commerce. The goal of ebInterface is to define an unambiguous e-Invoicing standard for the Austrian market. In the current status an agreement between twelve COTS vendors on the core elements has been established. However, an extension mechanism for domain-specific amendments (e.g., telecom industry) is still missing. In the paper at hand we introduce different XML extension mechanisms for defining domain-specific extensions in a bottom-up business document stan-

dard. The goal is to define a plugin-based solution in order to add industry and partner-specific extensions without altering the inner core of a bottom-up standard. Thus, interoperability of the core specification is provided at any time with any given partner. We specifically focus on the strengths and weaknesses of each extension mechanism and evaluate the applicability of each extension approach using the ebInterface standard. Thereby, we evaluate every approach in regard to four criteria: i) core schema integrity ii) core schema compatibility iii) extension control, and iv) guarantee of validity. We state that a successful bottom-up standard extension approach must meet all of the four criteria.

The rest of the paper is structured as follows. First, Section 2 introduces current research work in the field of XML Schema extension approaches. Section 3 introduces the ebInterface core standard and Section 4 evaluates the different extension mechanisms of XML Schema and presents our solution for a flexible bottom-up extension of the ebInterface standard. Finally, Section 5 concludes the paper.

2. Related Work

A literature review shows that in particular XML Schema has been subject to much controversy in regard to its expressiveness and complexity [8]. Nevertheless, it has become the de facto standard for defining data exchange formats in particular in the context of Web Services [15]. Pasley [11] examines the potential risks if wildcard extension mechanisms such as `xs:any` are used in XML Schema definitions. The author provides a set of best practices to XML Schema design, helping to cope with changing schema requirements and schema extensions. However, most of the recommendations aim at changing the core schema and thus they are not applicable to the scenario presented in this paper, where the core schema must remain unchanged. Another important field in particular in regard to business document standardization is the research area of XML evolution [13], [5], [4]. The developed methods aim at automatically adapting XML instance documents in case the associated XML Schema is extended or restricted by additional elements. Although several academic approaches for XML evolution exist, their integration level in B2B tools remains quite low. Generally, if schemas evolve and several versions of a schema are developed, a set of problems, e.g., revalidation issues occur [12], [3], [2]. Our presented approach aims at circumventing error-prone and time-consuming revalidation tasks of multiple schema versions by providing a single, but flexible solution for business document standard extensions.

3. ebInterface - The Core

In the following we introduce the XML-based standard ebInterface and provide an accompanying invoice example from the telecom domain. We use the example throughout the article to evaluate the applicability of the introduced XML Schema extension mechanisms. Figure 1 illustrates

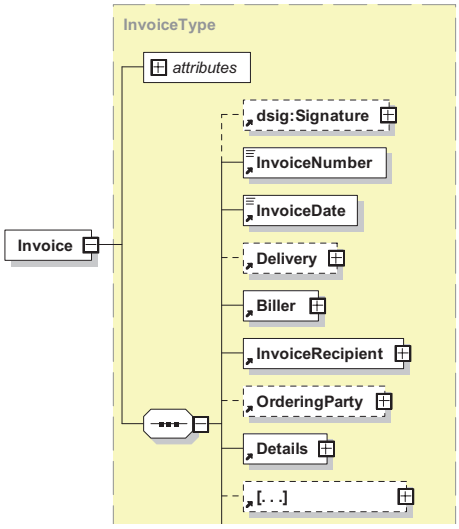

Figure 1: A cut-out of the ebInterface standard

the main structure specified by the ebInterface standard, including elements such as `InvoiceDate`, `Biller` and `Details`. A more detailed view of the element `Details` is given in Figure 2. The element `Details` may be used to represent items typically listed in an invoice, hence it is designed to contain one or more elements named `LineItem`. Each element `LineItem` contains further elements such as `PositionNumber` and `UnitPrice`. An excerpt of the ebInterface XML Schema representing an element `LineItem`, which is the target of extensions in the following sections, is provided in Listing 1. All other elements of the standard are not discussed any further, but can be found in the ebInterface specification [1]. To illustrate the

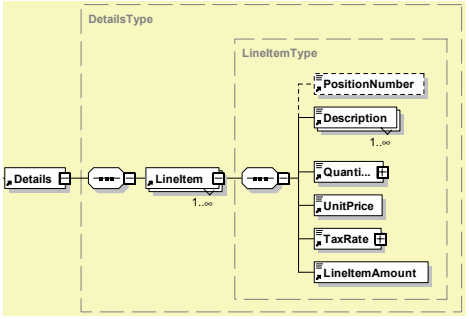

Figure 2: The details section

use of the ebInterface standard the following example is introduced. The example is based on invoices for end customers used in the telecom industry.

Listing 1: Line Item XML syntax

```
<xs:complexType name="LineItemType">
 <xs:sequence>
  <xs:element ref="PositionNumber" minOccurs="0"/>
  <xs:element ref="Description" maxOccurs="unbounded"/>
  <xs:element ref="Quantity"/>
  <xs:element ref="UnitPrice"/>
  <xs:element ref="TaxRate"/>
  <xs:element ref="LineItemAmount"/>
 </xs:sequence>
</xs:complexType>
```

Figures 3 and 4 illustrate excerpts from a real world telephone bill. The first excerpt illustrated in Figure 3 shows

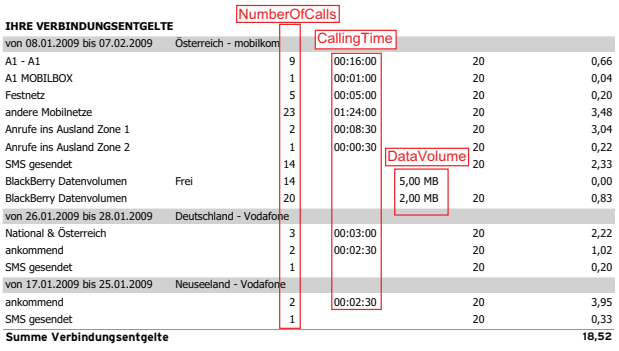

Figure 3: Excerpt from Example Invoice

different line items such as "A1 - A1" and "Festnetz". The line item "Festnetz" for instance represents a summary of all calls made to landline phones. In the ebInterface standard line items are typically represented using the element `LineItem` (cf. Figure 2 or Listing 1). However, the elements of `LineItem` are not sufficient to fully represent all information of a line item in the telephone bill. In fact, the number of calls, calling time, and data volume (cf. Figure 3) cannot be represented using the element `LineItem`. The

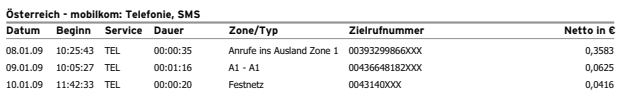

Figure 4: Excerpt from Itemized Bill

second excerpt illustrated in Figure 4 shows a detailed view of the call charges, listing every single call made including certain details such as the date or duration of the call. The detailed view is from now on referred to as itemized phone bill. Representing the itemized phone bill using the ebInterface standard is currently not possible either, since the standard neither provides suitable elements nor any mechanism to extend the standard itself. We will elaborate on different strategies for an extension of a bottom-up standard in the next section.

4. Alternative Strategies for a Bottom-Up Approach

In the following we evaluate how different extension mechanisms may be used in a real-world environment in order to extend an existing business document definition. Thus, we demonstrate the different extension mechanisms by means of the ebInterface standard. However, it should be noted that the proposed extensions are valid in any given bottom-up business document standard approach.

4.1. Custom Section

The first approach to meet the requirements of different stakeholders, i.e., storing customized information, is achieved through introducing a so-called custom section in the ebInterface XML Schema. The definition of the custom section is shown in Listing 2. The XML Schema element `xs:any` used for the definition of the custom section is discussed in the following.

Listing 2: Custom Section

```
<xs:group name="Custom">
 <xs:sequence>
  <xs:any namespace="##other" processContents="strict"/>
 </xs:sequence>
</xs:group>
...
<xs:complexType name="InvoiceType">
 <xs:sequence>
  ...
  <xs:group ref="Custom" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
```

The wildcard `xs:any` may be used in an XML Schema for defining placeholders, enabling stakeholders to store additional, custom information in the actual XML document instances. Note that the exact structure of the custom information is usually not yet known at schema design time. The `xs:any` element contains two attributes namely `namespace` and `processContents`.

The attribute `namespace` is used for specifying the namespace that the content in the instance document must comply with. Allowed values of the attribute include `##any`, `##local`, `##other`, `##targetNamespace`, and a particular namespace. Assigning the attribute `namespace` the value `##any` determines that the content may be any well-formed XML from any namespace. Using `##local` defines that the content may be any well-formed and unqualified XML. `##other` specifies that the content may be any well-formed XML from any namespace other than the current target namespace. `##targetNamespace` defines that the content may be any well-formed XML as long as it belongs to the `##targetNamespace`. The fifth option is to list one or more namespaces such as `http://www.ebinterface.at/ext/telecom` in the attribute `namespaces`. Listing one or more namespaces restricts the placeholder the strongest and defines that

the content must be any well-formed XML and that it must belong to any of the namespaces listed.

The attribute `processContents` provides instructions regarding the validation of the custom section in the instance document. Thereby, the attribute may have one out of the three following values: `strict`, `lax`, and `skip`. The attribute value `strict` specifies that content stored in the custom section must be qualified through a namespace. Second, the value `lax` expresses that content in the custom section may be validated in case the backing XML Schema is available. Otherwise, in case the backing XML Schema is not present, the validation of the instance document is skipped and validation succeeds. The third attribute value `skip` specifies that content in the custom section in the instance document is not validated at all. In the following three different approaches for defining single and multiple custom sections are explained.

xs:any and any namespace. Listing 2 illustrates the definition of the custom section used to extend the ebInterface standard. As shown in line 13 of Listing 2 the content of the custom section must be well-formed XML, defined in a namespace other than the current target namespace. Furthermore, the value of the attribute `processContents`, also illustrated in line 13 of Listing 2, is set to `strict` specifying that any content stored in the custom section must be backed by an XML Schema.

Therefore, in order to add custom information it is necessary to first define a custom XML Schema, specifying the content of the custom section. The XML Schema used in this example is illustrated in Listing 3. The schema describes an itemized phone bill containing a detailed list of all calls made and a summary of the duration and cost of all calls made (cf. Figure 4).

Listing 3: Domain-specific extension: XML Schema

```
23 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.
       ebinterface.at/ext/telecom" targetNamespace="http://www.ebinterface.at/
       ext/telecom" elementFormDefault="qualified" attributeFormDefault="
       unqualified">
24  <xs:complexType name="ItemTelephonyType">
25   <xs:sequence>
26    <xs:element name="Date" type="xs:date" />
27    <xs:element name="Begin" type="xs:time"/>
28    <xs:element name="Service" type="xs:string" />
29    <xs:element name="Duration" type="xs:time" />
30    <xs:element name="Type" type="xs:string" />
31    <xs:element name="NumberCalled" type="xs:string" />
32    <xs:element name="Amount" type="xs:decimal" />
33   </xs:sequence>
34  </xs:complexType>
35  <xs:complexType name="SummaryTelephonyType">
36   <xs:sequence>
37    <xs:element name="TotalDuration" type="xs:time" />
38    <xs:element name="TotalAmount" type="xs:decimal" />
39   </xs:sequence>
40 </xs:complexType>
41  <xs:element name="ItemizedBillTelephony">
42   <xs:complexType>
43    <xs:sequence>
44     <xs:element name="ItemTelephony" type="ItemTelephonyType" maxOccurs="
           unbounded"/>
45     <xs:element name="SummaryTelephony" type="SummaryTelephonyType"/>
46    </xs:sequence>
47   </xs:complexType>
48  </xs:element>
49 </xs:schema>
```

Listing 4 illustrates an excerpt of an XML document instance representing the custom section of the updated ebInterface XML Schema in combination with the XML Schema defined in Listing 3.

Listing 4: Domain-specific extension: XML instance

```
50 <eb:Invoice xmlns:eb="http://www.ebinterface.at/schema/3p0/" xmlns:tco="http://
       www.ebinterface.at/ext/telecom" xmlns:xsi="http://www.w3.org/2001/
       XMLSchema-instance" xsi:schemaLocation="http://www.ebinterface.at/schema
       /3p0/ Invoice.xsd http://www.ebinterface.at/ext/telecom Telecom.xsd">
51   ...
52  </eb:PresentationDetails>
53
54  <tco:ItemizedBillTelephony>
55   <tco:ItemTelephony>
56    <tco:Date>2009-01-01</tco:Date>
57    <tco:Begin>10:25:43</tco:Begin>
58    <tco:Service>TEL</tco:Service>
59    <tco:Duration>00:00:35</tco:Duration>
60    <tco:Type>Anrufe ins Ausland Zone 1</tco:Type>
61    <tco:NumberCalled>002337774335</tco:NumberCalled>
62    <tco:Amount>0.3583</tco:Amount>
63   </tco:ItemTelephony>
64
65   <tco:ItemTelephony>
66    <tco:Date>2008-12-31</tco:Date>
67    <tco:Begin>22:31:03</tco:Begin>
68    <tco:Service>SMS</tco:Service>
69    <tco:Duration>00:00:00</tco:Duration>
70    <tco:Type>SMS gesendet</tco:Type>
71    <tco:NumberCalled>01235681295</tco:NumberCalled>
72    <tco:Amount>0.1666</tco:Amount>
73   </tco:ItemTelephony>
74
75   <tco:SummaryTelephony>
76    <tco:TotalDuration>00:00:35</tco:TotalDuration>
77    <tco:TotalAmount>0.5249</tco:TotalAmount>
78   </tco:SummaryTelephony>
79  </tco:ItemizedBillTelephony>
80 </eb:Invoice>
```

Extending the standard through adding a custom section adds great flexibility to the ebInterface standard. Furthermore, the implementer is forced to define an XML Schema describing the structure of the information stored in the custom section. The latter is achieved through the attribute `processContents`. The attribute's value is set to `strict` instructing any parser performing validation of an ebInterface XML document that, in case a custom section is present, a corresponding XML Schema must exist. Alternatively, the `processContents` attribute's value could be set to `lax`. If set to `lax` an XML Schema which is present will be used for validation. However, the presence of an XML Schema is then optional.

This flexibility also implies that the standards body creating the standard may not be able to control the content stored in custom sections. Theoretically, implementers of the standard may store information completely disconnected from the context of an electronic invoice. Moreover, it is necessary to modify the original ebInterface XML Schema. Thus, the approach is not desirable for defining a core schema with domain-specific extensions.

xs:any and defined set of namespaces. As outlined in Section 4.1, the attribute `namespace` of the element `xs:any` allows to define a set of namespaces containing one or more namespaces. Having a set of defined namespaces requires that the content stored in the instance document must be backed by an XML Schema whose namespace is listed in the `namespace` attribute. Therefore, it is necessary to modify the definition of the

custom section (cf. Listing 2) and specify a set of allowed namespaces. In the current example the namespace `http://www.ebinterface.at/ext/telecom` is specified as illustrated in Listing 5. Note that the namespace refers to the XML Schema defined in Listing 3.

Listing 5: Single Custom Section XML Syntax

```
<xs:group name="Custom">
 <xs:sequence>
  <xs:any namespace="http://www.ebinterface.at/ext/telecom" processContents="
       strict"/>
 </xs:sequence>
</xs:group>
...
<xs:complexType name="InvoiceType">
 <xs:sequence>
  ...
  <xs:group ref="Custom" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
```

Thus, the elements in the custom section must correspond to the structure defined in the domain-specific extension XML Schema, identified through the namespace `http://www.ebinterface.at/ext/telecom` (cf. Listing 3). One of the major advantages resulting from utilizing the `xs:any` attribute `namespaces` is the ability to restrict the content of the custom section through listing a set of namespaces.

On the contrary, to list a set of namespaces, it is also required to modify the original ebInterface XML Schema, hence using `xs:any` is not a desired extension mechanism.

Multiple Custom Sections. Another possibility for utilizing the custom section is the use of references. Recall that the example used throughout the article assumes that the ebInterface instance document represents a telephone bill of a single telephone customer. The element `LineItem` specified through the element `LineItemType` is used to represent the different call charges. In addition an itemized phone bill was presented in the custom section of the ebInterface XML document instance (cf. Listing 4).

However, the example is further extended by assuming that the telephone bill represents the bill for two customers (e.g., in case of a partner tariff). Following the idea for a single customer, the element `LineItem` may be used to represent the call charges (cf. Figure 3) for both customers and the custom section may be used to store the itemized phone bill (cf. Figure 4) for both customers as well. However, it is not possible anymore to determine which itemized phone bill belongs to a particular call charge summary. To solve the problem it would be desirable to create appropriate references between call charges and itemized phone bills. One option to create references is to modify the element type `LineItemType` and add an additional element containing a unique identifier. Also, each section stored within the custom section must provide an element for storing references used in the call charge summaries. Through utilizing the reference mechanism it would be possible to assign each call charge summary particular itemized phone bills.

As shown by the example, it is a quite sophisticated process to use references to represent information properly. However, if it is desired to use references it is also necessary to modify the `LineItemType` of the ebInterface schema, which again is not a desired method to extend the ebInterface standard. An alternative approach to using references as described above may be achieved through introducing more than one custom section in the ebInterface XML Schema. The definition of a custom section would still be the same as shown in Listing 5.

A resulting advantage would be that the quite complex use of references can be avoided. Instead, custom sections may be added where needed. On the other hand, through a number of custom sections within an XML Schema, implementers may store customized information in any of the custom sections available. Hence, it is not possible to distinguish which customized section is used to store which kind of information. In the following we examine three more powerful extension mechanisms of XML Schema.

4.2. Redefine

A `redefine` element has a dual functionality. First, it implicitly includes the referenced schema file and thereby enables access to all of the elements of the referenced schema. Second, it enables the user to redefine zero or more of the components of the referenced schema. Using a redefine statement the user can extend or restrict an existing component. The redefine mechanism can only be applied if both schemas have the same target namespace or the included (redefined) schema has no target namespace.

Listing 6: Redefine XML Schema

```
93 <xs:schema xmlns="http://www.ebinterface.at/schema/3p0/" xmlns:xs="http://www.
       w3.org/2001/XMLSchema" targetNamespace="http://www.ebinterface.at/schema
       /3p0/" elementFormDefault="qualified" attributeFormDefault="unqualified"
       >
94  <xs:redefine schemaLocation="Invoice.xsd">
95   <xs:complexType name="LineItemType">
96    <xs:complexContent>
97     <xs:extension base="LineItemType">
98      <xs:sequence>
99       <xs:element name="NumberOfCalls" type="xs:integer" minOccurs="0"/>
100      <xs:element name="CallingTime" type="xs:integer" minOccurs="0"/>
101      <xs:element name="DataVolume" type="xs:decimal" minOccurs="0"/>
102     </xs:sequence>
103    </xs:extension>
104   </xs:complexContent>
105  </xs:complexType>
106 </xs:redefine>
107 </xs:schema>
```

As shown in line 94 of Listing 6 the redefine statement includes the main ebInterface schema and redefines the complex type `LineItemType` (line 95 to 105). The redefined `LineItemType` extends the original `sequence` and adds three elements: `NumberOfCalls`, `CallingTime`, and `DataVolume`. Note that the namespace of the redefined schema is the same as the namespace of the original ebInterface schema (line 93).

In the instance document shown in Listing 7 all elements of the `LineItem` element have the same namespace, since

the redefined schema does not apply a different namespace, but uses the original ebInterface namespace `http://www.ebinterface.at/schema/3p0/`. A different namespace prefix `tco` has been used in order to underline the fact that although the namespace `http://www.ebinterface.at/schema/3po` is still the same, the elements have telecom industry-specific extensions.

Using the redefine approach a new invoice definition is created for every domain-specific extension. The included elements from the original schema can be used as if they have been defined in the same schema. Furthermore, multiple namespaces are avoided, since all elements share the same `targetNamespace`.

Listing 7: Redefine XML instance document

```
108 <tco:Details>
109  <tco:LineItem>
110  <tco:PositionNumber>1</tco:PositionNumber>
111  <tco:Description>Calls to other providers</tco:Description>
112  <tco:Quantity tco:Unit="Units">60.00</tco:Quantity>
113  <tco:UnitPrice>0.1</tco:UnitPrice>
114  <tco:TaxRate>20.00</tco:TaxRate>
115  <tco:LineItemAmount>6.00</tco:LineItemAmount>
116  <tco:NumberOfCalls>43</tco:NumberOfCalls>
117  <tco:CallingTime>1800</tco:CallingTime>
118  </tco:LineItem>
119 </tco:Details>
```

Although smooth to implement, the redefine approach has a set of drawbacks. The redefined schema overwrites the element definitions of the original schema, thus making backward compatibility to the original ebInterface schema impossible. This means for example, that a system being capable of processing ebInterface schema `X` is not able to process schema `X'`, which is a redefined version of `X`. Furthermore, the possibility to redefine arbitrary `X'` schemas with domain-specific extensions leads to a multitude of different and incompatible business document definitions.

4.3. Substitution Group

The concept of substitution groups allows the substitution of an existing element (called head element) using another element from a defined group of elements (substitution group). First, a head element is declared followed by a definition which elements may be used to substitute it. The substitutable elements must have the same type as the head element or must have an extended or restricted type of the head element's type. The existence of a substitution group does not imply that the use of the elements in the substitution group is mandatory nor does a substitution group prevent the use of the head element.

As shown in Listing 8 a new schema is created for the telecom application domain. In line 121 the original ebInterface schema is imported and assigned with its original `targetNamespace`. Consequently, line 122 defines a new `LineItem` element which may serve as a substitutable element for the original ebInterface element `eb:LineItem`. The new `LineItem` element is defined in the target namespace `http://www.ebinterface.at/ext/telecom` and adds the three elements `NumberOfCalls`, `CallingTime`, and `DataVolume` to the original sequence.

Listing 8: Substitution Group XML Schema

```
120 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:eb="http://www.
        ebinterface.at/schema/3p0/" xmlns:tco="http://www.ebinterface.at/ext/
        telecom" targetNamespace="http://www.ebinterface.at/ext/telecom"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
121  <xs:import namespace="http://www.ebinterface.at/schema/3p0/" schemaLocation="
         Invoice.xsd"/>
122  <xs:element name="LineItem" substitutionGroup="eb:LineItem">
123   <xs:complexType>
124    <xs:complexContent>
125     <xs:extension base="eb:LineItemType">
126      <xs:sequence>
127       <xs:element name="NumberOfCalls" type="xs:integer" minOccurs="0"/>
128       <xs:element name="CallingTime" type="xs:integer" minOccurs="0"/>
129       <xs:element name="DataVolume" type="xs:decimal" minOccurs="0"/>
130      </xs:sequence>
131     </xs:extension>
132    </xs:complexContent>
133   </xs:complexType>
134  </xs:element>
135 </xs:schema>
```

Listing 9 shows how the concept of a substitution group is reflected in an instance document. The namespace prefix `tco` in line 137 refers to the namespace `http://www.ebinterface.at/ext/telecom` and thus it becomes apparent that the redefined `LineItem` from the telecom domain is used in this instance and not the original `LineItem` from the ebInterface schema. The two elements in line 144 and 145 also have the telecom-specific namespace. All the other elements are in the original ebInterface namespace `http://www.ebinterface.at/schema/3p0/`, indicated by the prefix `eb`.

Listing 9: Substitution Group XML instance document

```
136 <eb:Details>
137  <tco:LineItem>
138  <eb:PositionNumber>1</eb:PositionNumber>
139  <eb:Description>Calls to other providers</eb:Description>
140  <eb:Quantity eb:Unit="Units">60.00</eb:Quantity>
141  <eb:UnitPrice>0.1</eb:UnitPrice>
142  <eb:TaxRate>20.00</eb:TaxRate>
143  <eb:LineItemAmount>6.00</eb:LineItemAmount>
144  <tco:NumberOfCalls>43</tco:NumberOfCalls>
145  <tco:CallingTime>1800</tco:CallingTime>
146  </tco:LineItem>
147 </eb:Details>
```

An advantage of the substitution group approach is the fact that the original ebInterface schema remains unchanged. All extensions or restrictions on existing types are defined in a separate schema. In doing so a flexible and module-based extension approach is enabled. In the instance document the new elements induced by the substitution group are labeled by their specific namespace prefix, e.g., `tco` which equals namespace `http://www.ebinterface.at/ext/telecom` in Listing 9. Therefore, the domain-specific extensions can be easily distinguished from the original ebInterface elements.

A major shortcoming of the substitution group approach is the fact that a refined element is placed in its own namespace. In Listing 9 the substituted `LineItem` element is assigned its own namespace `tco` (cf. line 137). Thus, an application which is only capable of processing original ebInterface compliant instances cannot process the instance with the telecom-specific extensions, since it does not know

what, e.g., the element <tco:LineItem> is. This undermines the idea of a modular approach, where all applications should be able to process the core schema, regardless of what is defined in any extension module.

4.4. xsi:type Overloading

An approach similar to substitution groups is introduced with the concept of xsi:type. xsi:type uses the concept of type hierarchies, where a sub-type inherits features from a supertype by extension. First a supertype is created, followed by multiple specialized subtypes, meeting different requirements. Similar to the concept of polymorphism in object-oriented technologies, a derived type can be used wherever a base type is expected. Using the xsi:type a type of an element can be explicitly specified. Through this mechanism it is possible to specify a certain type of an element, although the specified type is not defined in the actual schema, but defined in another schema. The XML parser validates that the type which is specified in the xsi:type attribute is derived from the originally expected base type. Thus, the concept of xsi:type fits very well for extending a core schema with domain-specific amendments.

Before we can apply an xsi:type construct we have to extend a given base type. As shown in Listing 10 we define a new complex type tco:LineItemType (cf. line 150) by extending the base type eb:LineItemType (cf. line 152). Before the complex type is refined, the necessary type definitions are made available by importing the original ebInterface schema in line 149. Note that the new complex type is defined in the target namespace http://www.ebinterface.at/ext/telecom, specific to the telecom application domain. In the actual XML instance document the xsi:type attribute in a given element will be used to indicate of which type the given element is.

Listing 10: xsi:Type XML Schema

```
148 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:eb="http://www.
        ebinterface.at/schema/3p0/" xmlns="http://www.ebinterface.at/ext/telecom
        " targetNamespace="http://www.ebinterface.at/ext/telecom"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
149  <xs:import namespace="http://www.ebinterface.at/schema/3p0/" schemaLocation="
         Invoice.xsd"/>
150  <xs:complexType name="LineItemType">
151   <xs:complexContent>
152    <xs:extension base="eb:LineItemType">
153     <xs:sequence>
154      <xs:element name="NumberOfCalls" type="xs:integer" minOccurs="0"/>
155      <xs:element name="CallingTime" type="xs:integer" minOccurs="0"/>
156      <xs:element name="DataVolume" type="xs:decimal" minOccurs="0"/>
157     </xs:sequence>
158    </xs:extension>
159   </xs:complexContent>
160  </xs:complexType>
161 </xs:schema>
```

Listing 11 shows an XML document instance using an xsi:type construct. Using the xsi:type attribute in line 163 it is indicated that LineItem is of type tco:LineItemType. The two additionally used elements defined in tco:LineItemType are defined in the telecom-specific namespace http://www.ebinterface.at/ext/telecom as indicated by the prefix tco (cf. 170 line and 171).

Listing 11: xsi:Type XML instance document

```
162 <eb:Details>
163  <eb:LineItem xsi:type="tco:LineItemType">
164  <eb:PositionNumber>1</eb:PositionNumber>
165  <eb:Description>Calls to other providers</eb:Description>
166  <eb:Quantity eb:Unit="Units">60.00</eb:Quantity>
167  <eb:UnitPrice>0.01</eb:UnitPrice>
168  <eb:TaxRate>20.00</eb:TaxRate>
169  <eb:LineItemAmount>6.00</eb:LineItemAmount>
170  <tco:NumberOfCalls>43</tco:NumberOfCalls>
171  <tco:CallingTime>1800</tco:CallingTime>
172  </eb:LineItem>
173 </eb:Details>
```

By using the xsi:type approach a clear distinction between the core elements and the extension elements of the ebInterface schema is made. The extension elements are indicated using the tco namespace (line 170 to 171), but the extended element LineItem remains in the original namespace 163. The only indication that LineItem is a specialized type is given by the xsi:type attribute in line 163. In regard to schema modularity this approach is superior to all the other introduced approaches. If a software application has been designed to process only the core ebInterface standard, it is still able to receive a document including extensions, but it processes only the core and ignores the extensions.

The xsi:type extension mechanism has a set of minor shortcomings as well. First of all, it assumes that the receiver has the XML Schema for the instance document. In case the receiver uses alternative XML instance validation mechanisms such as RELAX NG [10], the xsi:type mechanism does not work. However, since the ebInterface standard relies on a well-defined XML Schema and every recipient of an instance document is supposed to have the relevant ebInterface schema definition including any necessary extension element definitions, this argument does not hold here. Second, some experts critically argue that the inclusion of the abstract type (i.e., the xsi:type) into the XML instance violates the paradigm of the separation of data (XML instance document) and data definition (XML Schema). While this argument might hold for XML purists, the superiority of the xsi:type approach in regard to XML Schema extension is evident. The last and probably major criticism of the xsi:type approach is in terms of processability of such schema constructs by tools, because not all parsers support xsi:type.

5. Conclusion

In the previous sections we have analyzed four different approaches in order to extend the existing ebInterface standard with domain-specific extension in a bottom-up manner. Thereby, we evaluated every extension approach in regard to four key criteria: i) core schema integrity ii) core schema compatibility iii) extension control, and iv) guarantee of validity. *Core schema integrity* refers to the fact,

whether an extension alters the original core schema definition. If an instance document with domain-specific extensions is still compatible with the core schema definition, it meets the criteria of *core schema compatibility*. *Extension control* refers to the fact, whether the extension mechanism allows for a governance of different extensions by a standardization body. If the core schema together with the domain-specific extensions may still be validated it meets the criteria of *guaranteed validity*.

The results of our study are aggregated in Table 1. The first three examined extension mechanisms used the concept of custom sections (A1-A3).

	A1	A2	A3	B	C	D
Core schema integrity	-	-	-	+	+	+
Core schema compatibility	-	-	-	-	-	+
Extension control	-	+	-	+	+	+
Guarantee of validity	+/-	+/-	+/-	+	+	+

Table 1: Comparison matrix
A1 - xs:any and any namespace; A2 - xs:any and defined namespaces; A3 - multiple custom sections; B - Redefine; C - Substitution Group; D - xsi:type

As clearly shown in Table 1 neither A1 (xs:any and any namespace), nor A2 (xs:any and defined namespaces), nor A3 (multiple custom sections) preserve the integrity of the core schema or ensure backward compatibility to the original schema. Extension control, that is the ability of a standardization organization to prescribe what XML elements to use in an extension, is only possible if using A2. Whether the overall validity of the core schema plus the extension may be guaranteed depends on how the `processContents` attribute is set. If it is set to `strict`, validity can be ensured. Thus, neither of the three approaches (A1-A3) using the concept of custom sections meets the requirements for an appropriate extension mechanism of a bottom-up schema.

Using `redefine` (B) it is possible to guarantee extension control and validity of the overall schema. Since a redefine statement imports the original schema and alters its elements in a new file, the original schema remains untouched. However, a `redefine` statement may alter any of the elements of the original schema, thus, obstructing backward compatibility to the core standard schema.

The same problem occurs if using the concept of substitution groups (C) for the extension of a core bottom-up schema. Although integrity of the original schema as well as extension control and validity is ensured, backward compatibility is violated. Eventually, the only remaining extension mechanism meeting all four requirements for the successful extension of a bottom-up business document standard is `xsi:type`.

In this paper it has been shown how a bottom-up business document approach can be extended to meet domain-specific requirements, while still maintaining interoperability at the core level. We examined different XML Schema extension mechanisms and analyzed their applicability for defining domain-specific XML Schema extensions by the example of the Austrian e-Invoice standard ebInterface. We evaluated every approach using four key indicators and concluded, that the concept of `xsi:type` is the only competitive solution for bottom-up schema extensions. Thus, we provided important input for the further development of XML-based business document standards.

References

[1] AustriaPRO. *ebInterface 3.0*, 2008.

[2] A. Balmin, Y. Papakonstantinou, and V. Vianu. Incremental validation of XML documents. *ACM Trans. Database Syst.*, 29(4):710–751, 2004.

[3] D. Barbosa, A. O. Mendelzon, L. Libkin, L. Mignet, and M. Arenas. Efficient Incremental Validation of XML Documents. In *Proceedings of the 20th International Conference on Data Engineering*, pages 671–682. IEEE Computer Society, 2004.

[4] G. Guerrini and M. Mesiti. X-Evolution: A Comprehensive Approach for XML Schema Evolution. In *Proceedings of the 19th International Conference on Database and Expert Systems Application*, pages 251–255, 2008.

[5] G. Guerrini, M. Mesiti, and D. Rossi. Impact of XML schema evolution on valid documents. In *Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 39–44. ACM, 2005.

[6] D. J. Kim, M. Agrawal, B. Jayaraman, and H. R. Rao. A Comparison of B2B e-Service Solutions. *Communications of the ACM*, 46(12):317–324, 2003.

[7] H. Li. XML and Industrial Standards for Electronic Commerce. *Knowledge and Information Systems*, 2(4):487–497, 2000.

[8] W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML Schema. *ACM Trans. Database Syst.*, 31(3):770–813, 2006.

[9] J.-M. Nurmilaakso, P. Kotinurmi, and H. Laesvuori. XML-based e-Business Frameworks and Standardization. *Computer Standards and Interfaces*, 28(12):585–599, 2006.

[10] OASIS. RELAX NG. http://relaxng.org/.

[11] J. Pasley. Avoid XML schema wildcards for Web service interfaces. *Internet Computing, IEEE*, 10(3):72–79, 2006.

[12] M. Raghavachari and O. Shmueli. Efficient Revalidation of XML Documents. *Knowledge and Data Engineering, IEEE Transactions on*, 19(4):554–567, April 2007.

[13] H. Su, D. K. Kramer, and E. A. Rundensteiner. XEM: XML Evolution Management. Technical report, Worchester Polytechnic Institute, 2002.

[14] UN/CEFACT. United Nations Electronic Data Interchange For Administration, Commerce and Transport (UN/EDIFACT). http://www.unece.org/trade/untdid/welcome.htm.

[15] E. Wilde. What are you talking about? In *Proceedings of the International Conference on Services Computing*, pages 256–261. IEEE, 2007.