

Contexts and interoperability in UBL2

Namespaces

To understand and parse an xml-instance, namespaces are used to distinguish where the content is defined. UBL 2.0 consists of approximately 29 business documents, the majority used in the procurement process. All shared components (BIEs) are placed in the common library, the transportation library or the procurement library. Due to the great number of business documents and the more comprehensive model (compared to UBL 1.0) the reusable components tend to be very complex and loosely defined.

The common libraries, when represented as xsd-schema, are defined in one schema file called common aggregate components. All common aggregate components share the same namespace (in the writing moment)

urn:oasis:names:draft:ubl:schema:xsd:CommonAggregateComponents-2.

Problems with reuse

A common component is the sum of the requirements from all documents and cases where it is reused. The reuse makes the documents more complex and comprehensive than they need to be in a non-technical point of view.

For example, let's take a look at the Order and OrderLine.

Order -->
 OrderLine -->
 QuotationLineReference -->
 AccountingDocumentReference -->
 PaymentMeans -->
 Payment -->
 ReceivedDateTime

There is no business need for having (zero to many) Payment ReceivedDateTime on a QuotationLineReference on an OrderLine. It is there because of reuse of some common components which in some other case or context need to be present.

If all elements and attributes in an UBL2 order are enumerated, we will find approximately 50 000 entities (only xml elements counted, attributes not included (but might be 100 000 if you count low). There are no business needs for 150 000 elements and attributes in an order. Actually, without very narrow restrictions, based on context, the order is useless.

Hence, we need some way of making a more narrow definition, a way of communicating what definition we use and a method of verifying/validating the narrow definition.

The definition can be made with a restricted xsd-schema where only the required information is included. It can also be achieved by the use of schematron rules where the rules replace the restrictions in the schema.

What makes an instance interoperable with another?

It is common that developers “hard-code” or, in their programming code, defines the namespaces that an xml-instance uses. This is not a problem if the namespace is always the same. This is also a main concern when deciding on versioning scheme on namespaces. In UBL only the major version number is stated in the namespace. Developers do not need to re-write their code just to change a minor version upgrade.

Namespace declaration in code can of course be made configurable and be changeable without re-writing any code but it is not the most common pattern. This could be an argument for using the same namespace regardless of the context or business process. But there is more to interoperability of course. What is achieved and made easier by just be able to read xml instances from other business processes than your system is designed to deal with?

Contexts

UN/CEFACT defines the following contexts in the core component specification (see: 5.6.1 in CCTS):

- Business Process Context
- Product Classification Context
- Industry Classification Context
- Geopolitical Context
- Official Constraints Context

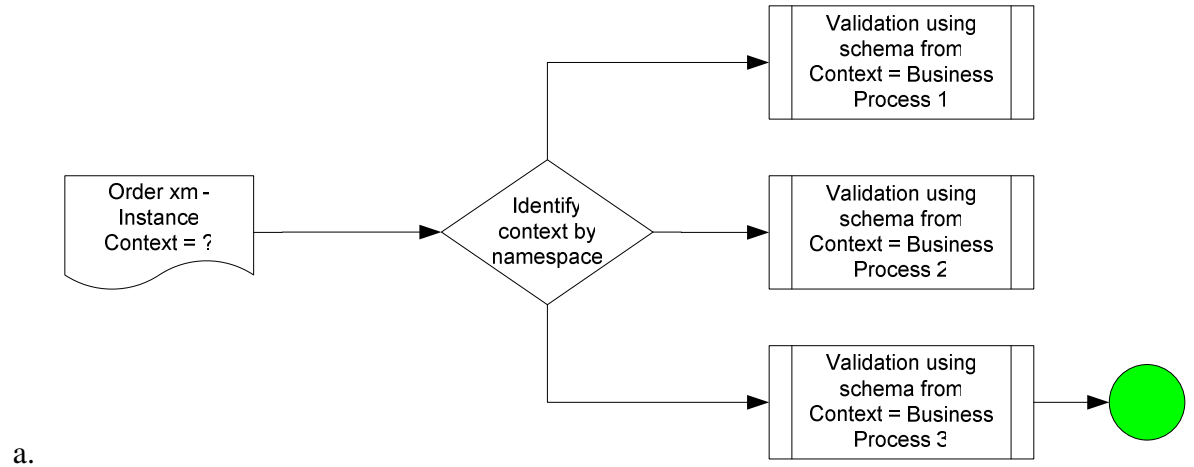
GS1 defines the following contexts

- Business Process Context
- Industry Sector Context
- Geopolitical Context
- Classification Context

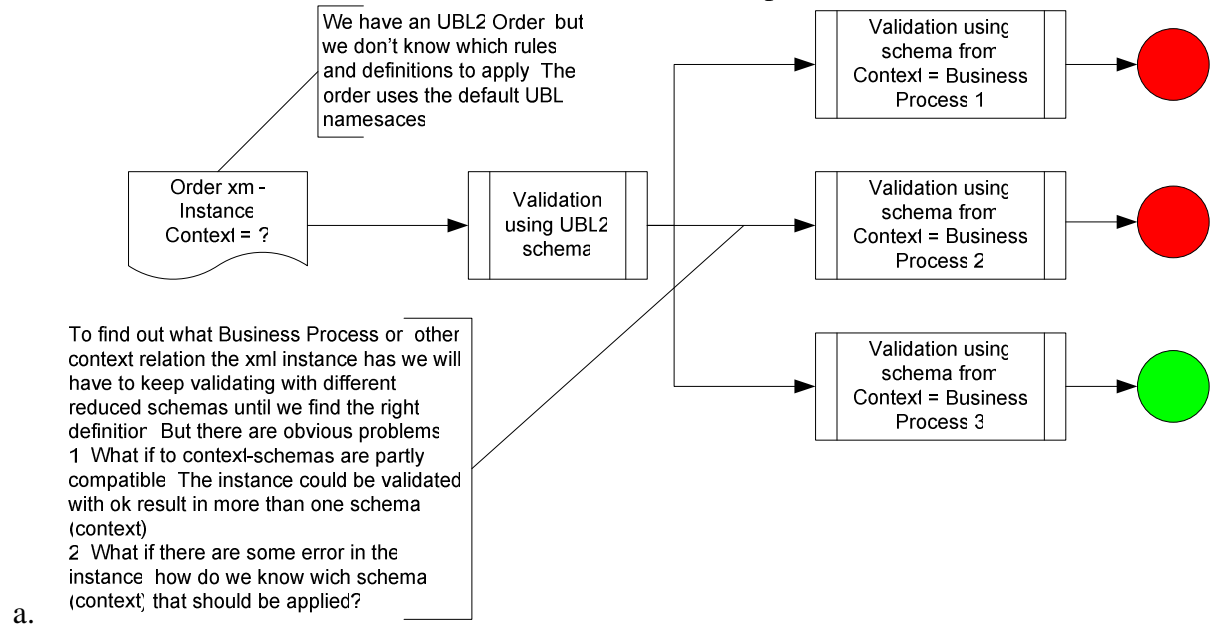
How do I know what context or business process an instance belongs to?

I can see five solutions on this problem.

1. Define new namespaces where the context is reflected



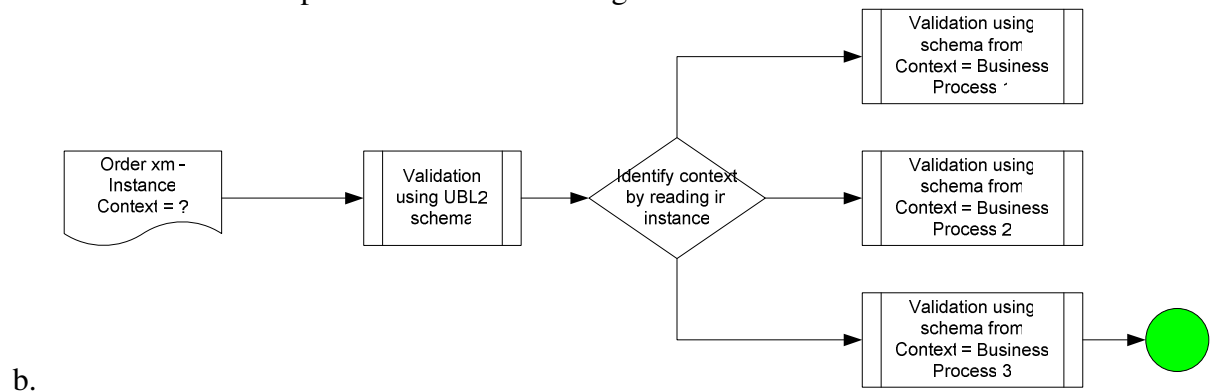
2. Use reduced schemas and validate to see if the instance is compatible



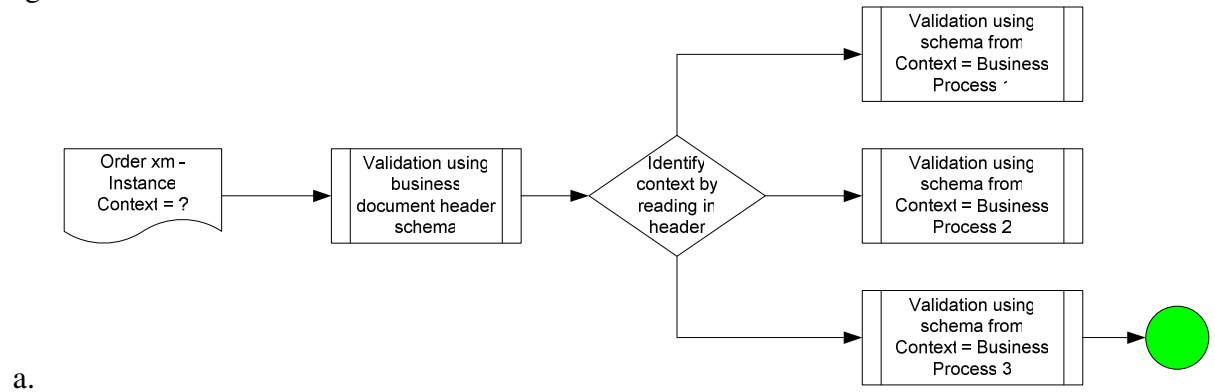
3. Use a BIE in the instance where the context is defined

a. <Invoice>

<Context>EuropeanSubset:FrameworkAgreementProcurementBP</Co..



4. Using a standard business document header where the context can be defined



5. Using envelope from e.g. ebXML where context is defined.