

## A Semantic based Solution for the Interoperability of UBL Schemas

Yalin Yarimagan and Asuman Dogac  
Middle East Technical University

**Abstract:** Universal Business Language (UBL) is an initiative to develop common business document schemas to provide document interoperability in the electronic business domain. However, businesses operate in different industry, geopolitical, and regulatory contexts and they have different rules and requirements for the information they exchange. As a result, several trading communities may prefer to tailor the UBL schemas to their needs. When these communities using different customizations wish to communicate, there is a need to translate their schemas to each other.

In this paper, we describe how to enhance UBL with semantics-based translation mechanisms to maintain the interoperability between documents conforming to different schema versions. For this purpose, we develop a Component Ontology for UBL using the Web Ontology Language to represent the semantics of individual components and their relationships within schemas. This ontology allows the utilization of descriptive logic reasoners for the discovery of similar components and the automation of the translation process.

### Universal Business Language

Businesses need to exchange data with their trading partners to execute transactions. Reusing well-understood standard patterns in documents makes business processes easier to implement, manage, and improve. Universal Business Language (UBL)<sup>1</sup> is an OASIS standard providing a library of reusable component schemas defined using the XML Schema Definition (XSD) language, such as Address, Price and a set of document schemas such as Order, Invoice.

UBL is being adapted by several communities around the world. Examples include Denmark, where UBL Invoice has been mandated by law for all public-sector businesses<sup>2</sup>; Sweden, where the National Financial Management Authority recommended UBL Invoice for all government use<sup>3</sup>, US Department of Transportation which is developing a UBL based pilot project for a demonstration of state-of-the-art electronic commerce in a real-world setting<sup>4</sup>. Denmark and Sweden define subsets of UBL 2.0 for customization by layering on business rules implemented in Schematron (<http://www.schematron.com>) which is a small language for making assertions about the presence or absence of patterns in XML documents.

However, since businesses operate in different industry, geopolitical, regulatory contexts and they have different rules and requirements for the information they exchange, sub setting and customization may not always serve the needs especially in the case of small and medium enterprises. In other words, with the increasing popularity of UBL, it is reasonable to expect that user communities of any size wishing to customize the UBL schemas according to their needs. However, as communities prefer using non-standard schemas, it becomes harder to maintain the interoperability within the UBL community.

In order to provide a solution to this interoperability problem, we propose a semantic translation mechanism that converts documents between schemas customized for different business contexts. Figure 1 displays our approach, in which the UBL community is divided into smaller communities based on their business context. Each community uses schemas tailored for their particular business needs. Parties within a community use the same version and can communicate with each other seamlessly. When parties from different communities need to make business with each other, they still use their schemas and the interoperability is provided by translating documents from one context to another.

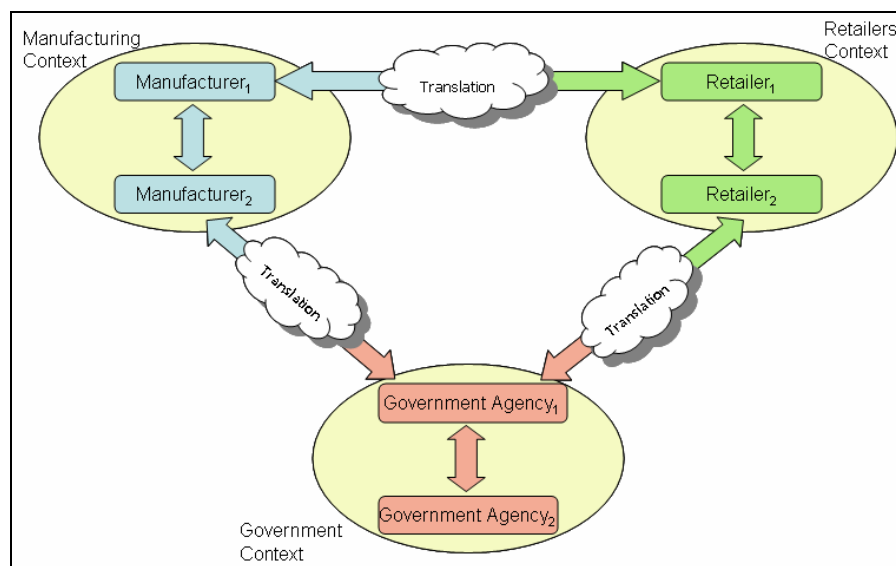


Figure 1 –Interoperability among different UBL communities through semantic-based translation mechanisms

### Web Ontology Language to Describe UBL Components and Schemas

The first step to develop such a translation capability is to provide a machine processable mechanism that can express the structure and the semantics of the components together with their correspondences in different versions. Considering the complexity of UBL schemas and the physically distributed nature of communities, it is not reasonable to expect manual specification and maintenance of all relationships among practically unlimited number of customized schemas.

As a solution, we developed a “Component Ontology” for UBL to provide a formal representation of components using the Web Ontology Language (OWL). This eliminates the maintenance burden as it only requires the specification of explicit relationships which can then be used for discovering implicit relationships through reasoning. The Component Ontology serves two major purposes:

- *Representing the semantics of UBL components:* UBL customization takes place at the level of individual types and elements; hence, translation needs to be done at the same level. When an automated process compares two versions of a schema, it needs to be able to identify corresponding elements in these schemas. When UBL elements are represented as classes of a common component ontology, it becomes possible to utilize that ontology for the computation of similarities between elements from different schemas.
- *Representing the structure of UBL schemas:* UBL document schemas are complex hierarchies including tens of types and elements any of which can be modified through customization. Even in the case of semantically equivalent components, comparison is a complicated task unless supported by proper tools. By representing the structural layout of types and elements using classes from a common component ontology, it becomes possible to utilize semantic reasoners for the comparison of these complex hierarchies.

The Component Ontology consists of classes representing UBL constructs such as type definitions, element declarations and business concepts. In order to represent various relationships between these constructs, we define object-type properties for classes and specify existential restrictions over object-type properties. When reasoners interpret these existential restrictions, they compute equivalence and/or subsumption relationships between classes that are restricted through similar expressions. In the following, we describe how the Component Ontology is defined and then explain how documents are translated using this ontology.

## Related Work about Ontology Based Interoperability of Information

Information society demands for complete access to available information, which is often heterogeneous and distributed. The problem of bringing together heterogeneous and distributed computer systems is known as the *interoperability problem*. Problems that might arise due to heterogeneity of the data are generally classified as structural heterogeneity and semantic heterogeneity. Structural heterogeneity means that different information systems store their data in different structures<sup>5</sup>. Semantic heterogeneity means that content and meaning of an information item is treated differently in different systems<sup>6</sup>.

In order to achieve semantic interoperability in a heterogeneous information system, the *meaning* of the information that is interchanged has to be understood across the systems. Semantic conflicts occur whenever two contexts do not use the same interpretation of the information. The use of ontologies for the explication of implicit and hidden knowledge is a possible approach to overcome the problem of semantic heterogeneity. Uschold and Gruninger mention interoperability as a key application of ontologies and point to several ontology based approaches for interoperability<sup>7</sup>.

A survey on the usage of ontologies for semantic interoperability summarizes the major role of ontologies as follows: "Ontologies are most frequently used in integration tasks to describe the semantics of the information sources and to make the content explicit. With respect to the integration of data sources, they are used for the identification and association of semantically corresponding information concepts by providing a vocabulary for the specification of the semantics of the relevant information systems"<sup>8</sup>.

## UBL Component Ontology

UBL provides several document schemas and a library of components consisting of elements and type definitions that are used in document schemas. Document schemas are collections of basic and aggregate components which recursively contain other components.

As an example, Figure 2-(a) displays a simplified Order schema and Figure 2-(b) provides corresponding XSD definitions (further details for IssueDate, SellerParty and OrderLine are intentionally omitted to keep the figure simple).

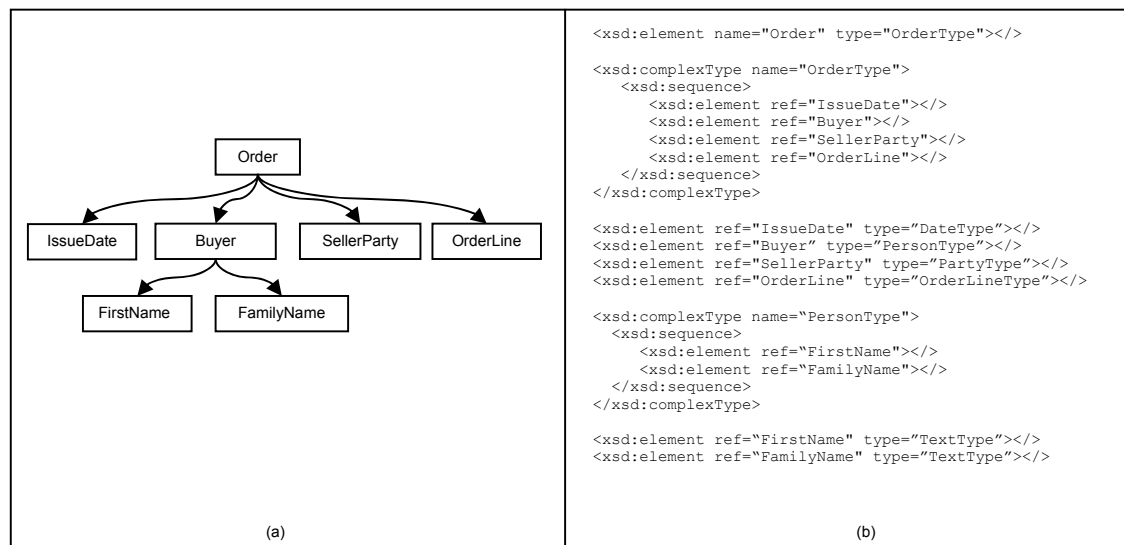


Figure 2 - (a) Simplified Order Schema (b) Corresponding XSD definitions

In order to represent relationships between entities, types and business concepts, we have developed the Component Ontology template shown in Figure 3. DataType, TypeDefinition,

ElementDeclaration and Concept are root classes of the Component Ontology. For each type and element declaration, we define corresponding subclasses under these root classes. Every UBL element represents a unique business concept or an entity. There are no redundancies among elements provided by the UBL and the reuse of existing elements is mandated whenever possible. However, as mentioned earlier, implementations prefer to define their own elements at the expense of redundancy. To be able to avoid semantic redundancy in schemas while giving the implementers the freedom to add elements as they need, we define the Concept class in our template and add a corresponding subclass for each unique business concept and/or entity represented by elements. This allows the definition of multiple elements representing the same business concept/entity and their correspondence is expressed through their relation to the same Concept class.

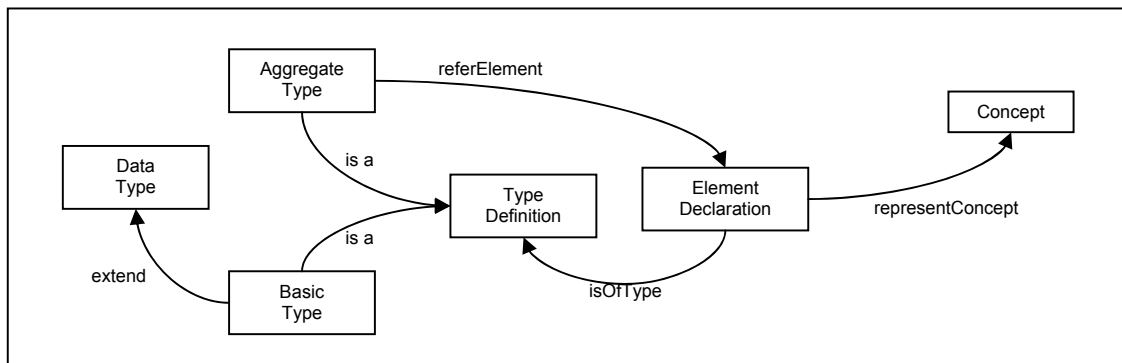


Figure 3 – Proposed Component Ontology

In the following, we describe how we relate classes to each other through object properties:

- Basic UBL types are defined through extending simple data types such as text, integer, date. There may be multiple basic types extending the same data type, such as `FirstNameType`, `FamilyNameType`, `AddressLineType` all being defined through extending the `TextType`. In order to represent the relationship between classes representing basic types defined by extending the same data type, we use the `extend` object property.
- UBL defines aggregate types as sequences of element declarations. In order to represent the relationship between classes representing aggregate types that refer to a similar set of elements, we use the `referElement` object property.
- Every UBL element has a type and we use the `isOfType` object property to represent the relationship between classes representing type definitions and element declarations.
- As described previously, we allow the definition of multiple elements that represent identical business concepts and relate element declaration classes to corresponding business concept classes through the `representConcept` object property.

Every class is defined together with existential restrictions over object properties to specify relationships distinguishing that particular class. This allows the utilization of reasoners to classify classes that are described (restricted) through similar relationships. In the following, we introduce the different types of existential restrictions and then present how reasoners interpret those restrictions through an example.

1. Two basic types that are derived from the same data type can be translated to each other. In order to represent the relationship between classes representing such types, following existential restrictions are defined for the `extend` property between `BasicType` classes and corresponding `DataType` classes.

$$aType \equiv (BasicType \sqcap (\exists extend. aDataType))$$

When such expressions are defined for all `BasicType` classes, reasoners classify those that have `extend` relation with the same `DataType` class to be equivalent to each other.

2. Two aggregate types that refer to the same set of elements can be translated to each other. In order to be able to compute the relationship between classes representing such types, following existential restrictions are defined over `referElement` properties between `AggregateType` classes and `ElementDeclaration` classes.

$$aType \equiv (AggregateType \cap (\exists referElement. (anElement_1 \cap anElement_2 \cap \dots \cap anElement_n)))$$

When such expressions are defined for all `AggregateType` classes, reasoners classify those that have `referElement` relation with the same set of `ElementDeclaration` classes to be equivalent to each other.

3. Every element declaration in UBL represents a business concept. The “type” of the element defines how the content of the element is structured. Hence, elements that represent the same business concept and have the same structure can be translated to each other. To be able to compute the relationship between classes representing such elements, following existential restrictions are defined for `ElementDeclaration` classes:

$$anElement \equiv (ElementDeclaration \cap (\exists representConcept. aConcept) \cap (\exists isOfType. aType))$$

When such expressions are defined for all `ElementDeclaration` classes, reasoners classify those that have `representConcept` and `isOfType` relation with the same `Concept` and `TypeDefinition` classes to be equivalent to each other.

### Computing Translations through Reasoning

Even though these expressions seem trivial, their expressiveness increase when they are used together. Relationships between basic constructs trigger the computation of relationships between higher level constructs which recursively trigger the computation of additional relationships. As an example, consider the `CustomOrder` schema in Figure 4 which is a possible customization of the `Order` schema in Figure 2.

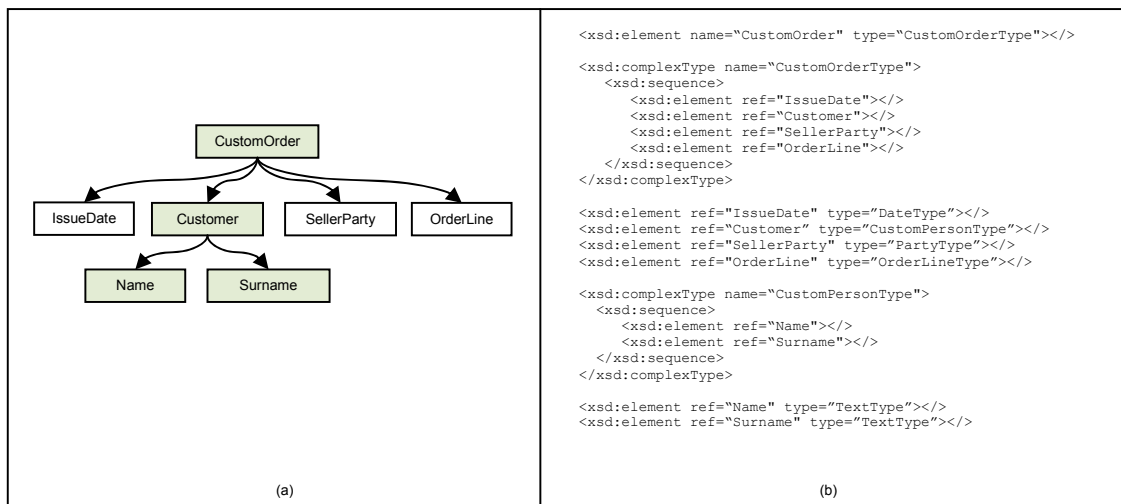


Figure 4 - (a) An example customization of `Order` schema in Figure 2 (b) Corresponding XSD Definitions

Figure 5-(a) lists the set of existential restriction expressions corresponding to both schemas; expressions 1 through 8 are defined for the `Order` schema in Figure 2 and expressions 9 through 16 are defined for the `CustomOrder` schema in Figure 4.

<ol style="list-style-type: none"> <li>1. Order <math>\equiv</math> (ElementDeclaration <math>\cap</math> (<math>\exists</math>representConcept. OrderConcept) <math>\cap</math> (<math>\exists</math>isOfType. OrderType))</li> <li>2. OrderType <math>\equiv</math> (AggregateType <math>\cap</math> (<math>\exists</math>referElement.(IssueDate <math>\cap</math> Buyer <math>\cap</math> SellerParty <math>\cap</math> OrderLine)))</li> <li>3. Buyer <math>\equiv</math> (ElementDeclaration <math>\cap</math> (<math>\exists</math>representConcept. BuyerConcept) <math>\cap</math> (<math>\exists</math>isOfType. PersonType))</li> <li>4. PersonType <math>\equiv</math> (AggregateType <math>\cap</math> (<math>\exists</math>referElement.(FirstName <math>\cap</math> FamilyName)))</li> <li>5. FirstName <math>\equiv</math> (ElementDeclaration <math>\cap</math> (<math>\exists</math>representConcept. FirstNameConcept) <math>\cap</math> (<math>\exists</math>isOfType. FirstNameType))</li> <li>6. FirstNameType <math>\equiv</math> (BasicType <math>\cap</math> (<math>\exists</math>extend. TextType))</li> <li>7. FamilyName <math>\equiv</math> (ElementDeclaration <math>\cap</math> (<math>\exists</math>representConcept. FamilyNameConcept) <math>\cap</math> (<math>\exists</math>isOfType. FamilyNameType))</li> <li>8. FamilyNameType <math>\equiv</math> (BasicType <math>\cap</math> (<math>\exists</math>extend. TextType))</li>   <li>9. CustomOrder <math>\equiv</math> (ElementDeclaration <math>\cap</math> (<math>\exists</math>representConcept. OrderConcept) <math>\cap</math> (<math>\exists</math>isOfType. CustomOrderType))</li> <li>10. CustomOrderType <math>\equiv</math> (AggregateType <math>\cap</math> (<math>\exists</math>referElement.(IssueDate <math>\cap</math> Customer <math>\cap</math> SellerParty <math>\cap</math> OrderLine)))</li> <li>11. Customer <math>\equiv</math> (ElementDeclaration <math>\cap</math> (<math>\exists</math>representConcept. BuyerConcept) <math>\cap</math> (<math>\exists</math>isOfType. CustomPersonType))</li> <li>12. CustomPersonType <math>\equiv</math> (AggregateType <math>\cap</math> (<math>\exists</math>referElement.(Name <math>\cap</math> Surname)))</li> <li>13. Name <math>\equiv</math> (ElementDeclaration <math>\cap</math> (<math>\exists</math>representConcept. FirstNameConcept) <math>\cap</math> (<math>\exists</math>isOfType. NameType))</li> <li>14. NameType <math>\equiv</math> (BasicType <math>\cap</math> (<math>\exists</math>extend. TextType))</li> <li>15. Surname <math>\equiv</math> (ElementDeclaration <math>\cap</math> (<math>\exists</math>representConcept. FamilyNameConcept) <math>\cap</math> (<math>\exists</math>isOfType. SurnameType))</li> <li>16. SurnameType <math>\equiv</math> (BasicType <math>\cap</math> (<math>\exists</math>extend. TextType))</li> </ol> <p style="text-align: right;">(a)</p>
<ol style="list-style-type: none"> <li>17. FirstNameType <math>\equiv</math> NameType <span style="float: right;">(6 and 14)</span></li> <li>18. FirstName <math>\equiv</math> Name <span style="float: right;">(5, 13 and 17)</span></li> <li>19. FamilyNameType <math>\equiv</math> SurnameType <span style="float: right;">(8 and 16)</span></li> <li>20. FamilyName <math>\equiv</math> Surname <span style="float: right;">(7, 15 and 19)</span></li> <li>21. PersonType <math>\equiv</math> CustomPersonType <span style="float: right;">(4, 12, 18 and 20)</span></li> <li>22. Buyer <math>\equiv</math> Customer <span style="float: right;">(3, 11 and 21)</span></li> <li>23. OrderType <math>\equiv</math> CustomOrderType <span style="float: right;">(2, 10 and 22)</span></li> <li>24. Order <math>\equiv</math> CustomOrder <span style="float: right;">(1, 9 and 23)</span></li> </ol> <p style="text-align: right;">(b)</p>

Figure 5 – (a) Existential restrictions corresponding to Order and CustomOrder (b) Computed relationships

Once these expressions are defined, similarly restricted classes are interpreted by a reasoner to compute additional relationships between classes. As an example, expressions 6 and 14 in Figure 5-(a) specify the same restriction for the `FirstNameType` and the `NameType` classes. As a result of this, they are computed to be equivalent to each other as shown in the expression 17 of Figure 5-(b). Following that computation, expressions 5 and 13 become equivalent; hence the `FirstName` and `Name` classes as shown in the expression 18 of Figure 5-(b). All computed equivalences resulting from expressions in Figure 5-(a) are shown in Figure 5-(b) to demonstrate how asserted relationships trigger the computation of additional relationships.

An equivalence relationship between classes expresses the structural and semantic similarity between corresponding constructs. Hence, we consider types/elements to be translatable to each other provided that the corresponding Component Ontology classes are equivalent. Note that the expression 18 in Figure 5-(b) expresses the equivalence between root elements of our example schemas, a result apparent for a human being. Yet, the example demonstrates how information is to be formalized so that an automated process can compute that same conclusion about the `Order` and the `CustomOrder` schemas.

### Components with Common Content

As can be expected, real life scenarios can be more complicated than this simple example and we do not always expect a one-to-one mapping between schemas to be able to translate them. In the following, we address how such cases are handled.

Even though original UBL specification includes no redundancy, tailored schemas might introduce redundancy as they define additional components resulting in multiple types with the same or very similar content. As an example, consider `NewOrderType1` in Figure 6-(a) and `NewOrderType2` in Figure 6-(b). Assume both are new types defined in different customizations as alternatives to the `OrderType` in Figure 2. We have already demonstrated how equivalent elements trigger relationships among higher level constructs, therefore for the sake of this example, assume new types either directly include common elements `Buyer`, `SellerParty`, `OrderLine` or they include other equivalent elements.

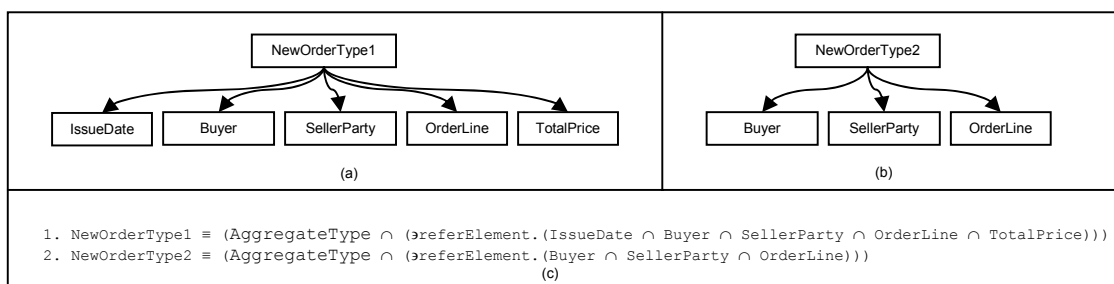


Figure 6 – (a), (b) Example custom types (c) Corresponding existential restriction expressions

Compared to the `OrderType` in Figure 2, `NewOrderType1` in Figure 6-(a) has an additional element, namely the `TotalPrice`. When the expression 1 in Figure 6-(c) is refactored as:

1. `NewOrderType1` = (AggregateType  $\cap$  ( $\exists$ referElement.(IssueDate  $\cap$  Buyer  $\cap$  SellerParty  $\cap$  OrderLine))  $\cap$  ( $\exists$ referElement. TotalPrice))
2. `OrderType` = (AggregateType  $\cap$  ( $\exists$ referElement.(IssueDate  $\cap$  Buyer  $\cap$  SellerParty  $\cap$  OrderLine)))
3. `NewOrderType1` = (`OrderType`  $\cap$  ( $\exists$ referElement. TotalPrice))

It can be seen that any class that satisfies the condition for `NewOrderType1` satisfies the condition for `OrderType` as well. Hence, when a reasoner interprets these expressions, it computes a subclass relationship between the `NewOrderType1` and the `OrderType`. Similarly, the expression 2 in Figure 6-(c) for `NewOrderType2` requires one less condition compared to the expression 2 in Figure 5-(a) and a superclass relationship is computed between the `NewOrderType2` and the `OrderType`.

This example demonstrates that class-subclass relationships are computed for types including common elements. Based on this, when a type has no equivalent class applicable to the target context we use an applicable superclass/subclass for translation.

More complicated cases result when two types have not only a common set of elements but also different sets of additional elements. For such types, common elements may or may not be relevant and/or sufficient for a proper translation; therefore we do not consider it feasible to define generic expressions that would drive automatic translation between such components with arbitrary number of common elements.

## Implementation

For the generation of the Component Ontology, we have developed a conversion tool that reads UBL schemas and creates corresponding class, object property and existential restriction definitions in OWL. The ontology we have generated for UBL 2.0 can be accessed at our web site ([http://www.srdc.metu.edu.tr/ubl/UBL\\_Component\\_Ontology.owl](http://www.srdc.metu.edu.tr/ubl/UBL_Component_Ontology.owl)).

In order to be able to translate documents exchanged between different communities, their customizations need to be reflected in the Component Ontology. For this purpose, we developed a web-based “Custom Component Registration Tool” to provide a GUI that generates necessary class definitions for the Component Ontology.

Our Translation Engine uses RacerPro (<http://www.racer-systems.com>) as the reasoner. Given a UBL document instance and target context, Translation Engine recursively traverses the document and for each component included in the original document, computes a corresponding component using the Component Ontology. For this computation, it first searches for an equivalent class, then a sub-class and then a superclass. Candidate components found in any of these steps are checked against parent components to make sure component schemas are not violated. Further details of our translation algorithm are provided at our web site (<http://srdc.metu.edu.tr/ubl/translation/src>). Once corresponding components are computed, they are appended to the target document instance to generate a translated version of the original document.

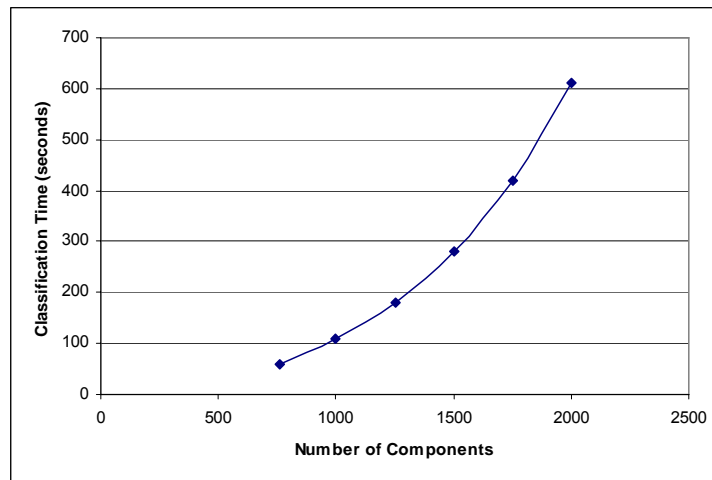


Figure 7 – Performance of the Translation Engine for computing equivalence and subsumption relationships between classes of the Component Ontology

Reasoning performance is an important aspect of our Translation Engine. Whenever a new UBL component is defined, corresponding classes are automatically added to the Component Ontology and the ontology is classified by the reasoner to compute equivalence and subsumption relationships between classes. Figure 7 provides performance figures for this classification operation measured on a Pentium 4 CPU 3.4 GHz Windows workstation with 1GB of RAM. Considering that the frequency of defining additional components is fairly low (possibly in the order of once per day), this once per component cost penalty is well within practically acceptable limits even on our modest test setup. Note that, after this computation, all subsequent requests are handled using the classified ontology until a new component is defined. The response time of the translation algorithm for such cases is in the order of milliseconds.

## Biography

**Yalin Yarimagan** is a PhD candidate at the Department of Computer Engineering at Middle East Technical University (METU). His research interests include semantic web, eBusiness and distributed systems. Yarimagan has an MSc in Computer Engineering from METU. Contact him at [yalin@srdc.metu.edu.tr](mailto:yalin@srdc.metu.edu.tr).

**Asuman Dogac** is a professor at the Department of Computer Engineering at METU and the CEO of the Software Research, Development and Consultancy company. Her research interests include semantic interoperability, eBusiness and eHealth. Dogac has a PhD in Computer Engineering from METU. She is a member of the IEEE and the ACM. Contact her at [asuman@srdc.metu.edu.tr](mailto:asuman@srdc.metu.edu.tr).

## References

- <sup>1</sup> Universal Business Language v2.0,  
<http://docs.oasis-open.org/ubl/os-UBL-2.0/UBL-2.0.html>
- <sup>2</sup> Adoption of UBL in Denmark - Business Cases and Experiences,  
<http://www.idealliance.org/proceedings/xttech05/papers/03-05-02/>
- <sup>3</sup> Svefaktura (SwedInvoice),  
[http://www.svefaktura.se/SFTI\\_Basic\\_Invoice20051130\\_EN/index.html](http://www.svefaktura.se/SFTI_Basic_Invoice20051130_EN/index.html)
- <sup>4</sup> The Electronics Freight Management White Paper,  
<http://www.itsdocs.fhwa.dot.gov/JPODOCS/REPTS TE/14246 files/14246.pdf>

- 
- <sup>5</sup> Won Kim and Jungyun Seo. "Classifying schematic and data heterogeneity in multidatabase systems." *IEEE Computer*, 24(12):12–18, 1991.
  - <sup>6</sup> Vipul Kashyap and Amit Sheth. "Semantic heterogeneity in global information systems: The role of metadata, context and ontologies.", *Cooperative Information Systems: Current Trends and Applications*. 1996.
  - <sup>7</sup> Mike Uschold and Michael Grüninger. "Ontologies: Principles, methods and applications." *Knowledge Engineering Review*, 11(2):93–155, 1996.
  - <sup>8</sup> H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner, "Ontology-based Integration of Information - A Survey of Existing Approaches," In: Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing, Seattle, WA, 2001, Vol. pp. 108-117.