CONCURRENT ENGINEERING: Research and Applications

Semantic B2B-integration Using an Ontological Message Metamodel

Marko Vujasinovic,[1,2,*] Nenad Ivezic,[1] Edward Barkmeyer[1] and Zoran Marjanovic[2]

[1]*Manufacturing Systems Integration Division, NIST, Gaithersburg, MD 20899-8260, USA*
[2]*Department of Information Systems, Faculty of Organizational Sciences, Belgrade, Serbia*

Abstract: E-Business applications are often required to use different, incompatible, message sets to implement message interfaces for a business-to-business (B2B) communication. This makes communication with every new partner a new interoperability problem. In this article, we present a semantic-mediation architecture that provides for interoperable B2B data exchange. The mediation process is based on an *Ontological Metamodel* of message schemas and messages, and uses existing *reference business ontologies*. An approach for message schema semantic reconciliation and annotation is devised to support the mediation. We demonstrate the approach on a scenario that involves two incompatible message interfaces, one UN/EDIFACT based and another OAGIS XML based.

Key Words: enterprise integration, semantic mediation, semantic reconciliation, semantic annotation, message model.

1. Introduction

Standard business-to-business (B2B) message schemas, such as UN/EDIFACT (www.unece.org/trade/untdid) or OAGIS BODs (www.oagi.org), standardize the B2B message definitions, and constrain the usage and relationships of message components – data types, elements, attributes, and their content. However, the standards-based integration has several shortcomings [1]. First, the business-concepts specification based on syntactic notations and diagrams leads to interpretation ambiguity. Second, informal annotation of message component semantics leads to the integration problems at the message semantic level. Third, the procedural mapping among application data concepts and standard message components leads to obscure and inflexible implementations. Additionally, there can be several *incompatible* standards for a particular message type, or several different customizations of a standard schema that use different terminologies to name message components, different structural organizations, or different data types. Also, the syntax, such as XML (Extensible Markup Language; www.w3.org/XML/) or EDI (Electronic Data Interchange), might differ.

To address these shortcomings, we explore a reference ontology-mediated semantic integration for B2B applications. Explicit and formal *reference ontologies*, as common business conceptual models for specific domains [2], are being recognized as the foundation for the B2B integration [3]. We base the reference ontology-mediated semantic integration on two assumptions: first, as long as peers are committed to a reference ontology, they can interact; second, the data concept mapping always takes place between the message schema concepts and reference ontology concepts. That decouples the peers. Here, we detail the proposed architecture, the integration steps, and the developed prototype tools. In [4] we reported on the initial set of tools that we first used in our architecture. Those tools' weaknesses motivated us to design and develop three novel foundational aspects of the architecture: (1) ontological message metamodel, (2) ontological message metamodel-based semantic reconciliation of messages, and (3) semantic annotation of message components definition. In the next section, we begin discussion of these essential advancements in the proposed architecture, which is further detailed through the rest of the article.

2. Semantic Mediation

2.1 Conceptual Architecture

Figure 1 shows the semantic-mediation architecture. *Reference ontology* formally captures the common business concepts and their relationships including the business meaning of the conceptual messages. We assume that a reference ontology is developed by the business community, represented using the OWL (Web Ontology Language; www.w3.org/2004/OWL/) or

*Author to whom correspondence should be addressed.
E-mail: marko.vujasinovic@gmail.com
Figures 1–5 appear in color online: http://cer.sagepub.com

1063-293X/10/03 0219–14 $10.00/0 DOI: 10.1177/1063293X10379764

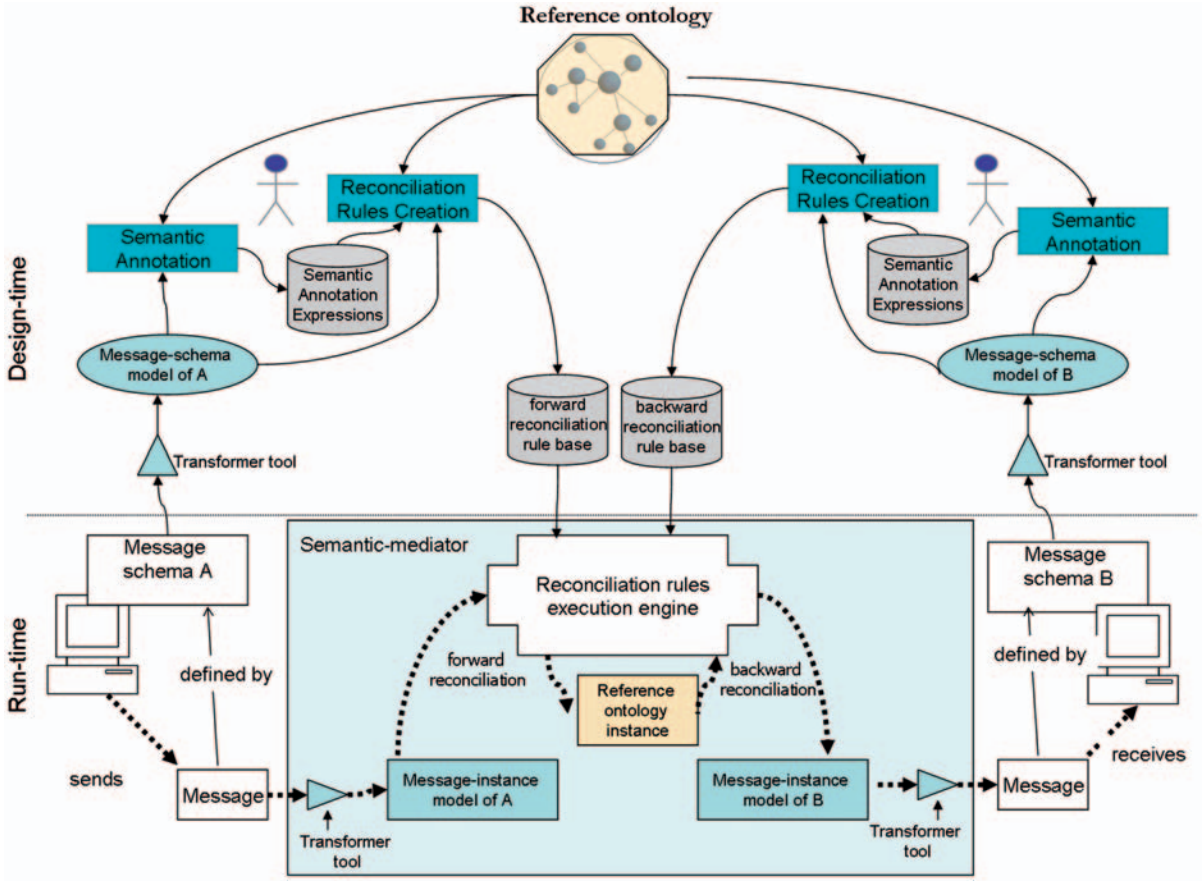

Figure 1. Semantic mediation: a conceptual architecture.

RDFS (Resource Description Framework Description Language; www.w3.org/TR/rdf-schema/), and publicly available. The *semantics annotation tool* provides for explicit and machine-processable *annotation expressions* of the message component definitions in terms of the reference ontology entities. The annotation tool requires a business expert to interpret the message component's meaning. The tool captures decisions made by the expert as formal expressions. The *reconciliation rules creation tool* provides the rule templates for executable reconciliation rules. Forward reconciliation rules specify the transformation of message content to reference ontology individuals, while backward rules specify the content transformation from the ontology individuals to a message. The reconciliation rules creation also requires human assistance, or at least oversight; however, the rules can be derived automatically in most cases from the annotation expressions. Deriving the rules from the annotation expressions steers the design time effort toward reconciliation. The *reconciliation rules execution engine* executes the forward reconciliation rules when the application is sending messages and the backward rules when the application is receiving messages. *Message model* abstracts underlying message and schema representation specifics, whilst providing a base for standard semantic annotation and reconciliation activities. *Message-representation transformation tools* provide transformation of schemas and messages into the corresponding *message-schema models* and *message-instance models,* respectively. *Semantic-mediator* component assembles the reconciliation engine and runtime message-representation transformation.

The advances of the architecture over the traditional integration architectures are: (1) *formal specification of the business domain concepts* provides a basis for unambiguous interpretation of the data-exchange artifacts; (2) semantic annotation of message schemas provides *machine-processable annotation expressions* that formally and precisely describe message components; (3) *automated and consistent semantic integration through the executable reconciliation rules* moves the engineering effort away from the implementation details; (4) *semantics-mediation* reconciles mismatches of different message forms. The architecture also supports message exchange between independently developed applications that have nonstandard schema-based interfaces.

2.2 Semantic Integration Methodology

At *design time*, first, a schema-language-specific transformer tool is used to produce message-schema models from given message schemas. Then, integration

engineers annotate the message-schema model concepts to formally describe the message components' meaning. Next, the engineers use a reconciliation tool to create required reconciliation rules; either by using the message-schema models and the reconciliation tool to manually create the rules, or by loading the annotations into the reconciliation tool to derive the rules from annotations automatically. When an ontology concept is required for comprehension, but does not appear in a message, a value must be provided by manually creating the reconciliation rule, as it cannot be derived from the annotations. At *runtime*, an application sends a message and the message-transformer tool transforms the message to a corresponding message-instance model. Then, the reconciliation engine takes that message-instance model and executes the forward rules for a sending application. This generates reference ontology individuals. Next, the reconciliation engine takes the ontology individuals and executes the backward rules defined for a receiving application. This generates a message-instance model for the receiving application. Finally, from that model, a specific message-transformer tool produces a schema-conformant message in the syntax expected by the receiving application. Effectively, the semantic-mediator provides the semantic integration capability.

3. Model of a B2B Message

The semantic-mediation requires a message model form that (1) decouples the architecture from underlying message and schema syntax specifics, (2) provides for the semantic annotation of message component definitions, and (3) provides for the generation of schema-conformant messages from the reconciliation output. There are two alternative approaches: a local conceptual message-schema (LCM), and here proposed ontological message metamodel form.

3.1 Inadequacy of a Local Conceptual Message Schema as a Message Model

Other similar approaches (Section 8) use an LCM as a message model. The LCM is a conceptual model of the message elements re-engineered from the message schemas, and represented using the OWL or RDFS. An LCM proved insufficient to capture required message information about (a) naming, (b) structure, (c) granularity, (d) occurrences, (e) value representation, and (f) formatting. For that reason, the LCM is ineffective for message components annotation, and insufficient for reconstructing schema-conforming messages from the reconciliation output, as we elaborated in [8]. To fulfill these requirements, LCM-based approaches require additional reconciliation rules that carry message formatting data through the semantic-mediation. That imposes undesired human effort and expert knowledge about message-representation and formatting, beyond understanding the message semantics. Also, LCM-extraction software may produce different views of the message structure, depending on the extraction strategy applied. For that reason, an integration engineer may be faced with an unfamiliar message structure that differs from its original structure defined by the message schema.

3.2 Ontological Message Metamodel

Alternatively to the LCM, the proposed novel ontological message metamodel (MMM) form abstracts syntax-specific concepts but faithfully captures the message component definitions as well as the key message information (i.e., bullets (a)–(f) in Section 3.1). The MMM form is a set of common representational concepts of different message and schema languages, devoid of the too specific representation rules. It intermediates between well-established message-representation standards, such as XML, ASN.1 [5], EDI and EXPRESS as the model for Clear Text Exchange [6]. The MMM is detailed in [7]; here, it is briefly introduced. Conceptually, the MMM has two parts – the message-schema metamodel concepts and the message-instance metamodel concepts – as illustrated in Figure 2. We use OWL to capture the MMM concepts and their instances (individuals). So, the MMM is an OWL T-Box, while the actual message-schema and message-instance models are sets of MMM concept individuals (OWL A-Box axioms). OWL representation of the MMM is well-suited for reconciliation between the particular message model on one side, and OWL reference ontology on other side as well as for establishing machine-processable annotation expressions.

Message-schema metamodel part defines concepts commonly used in the schema languages, such as XML Schema (www.w3.org/XML/Schema). The root concept is `Schema`. `Schemas` define `ContentModels` (`StructuredContents` and `SimpleContents`) and `Components` (`ElementModels` and `AttributeModels`). `StructuredContents` contain other `ContentModels`. For example, `StructuredContents` are XML Schema complex types, ASN.1 sequences, and EDI segments. `SimpleContents` represent datatypes, such as integer, string, or enumeration. `Value` represents and stores actual enumerated or other values. `ElementModel` concept represents message element definition. `ElementModel` content is defined either by `StructuredContent` or `SimpleContent` concept. For example, an XML element has a type that is a complex type or simple type. `AttributeModel` represents an element's attribute definition. `Name` concept captures `ContentModels` and `Components` literal names. `Names` are defined either in a

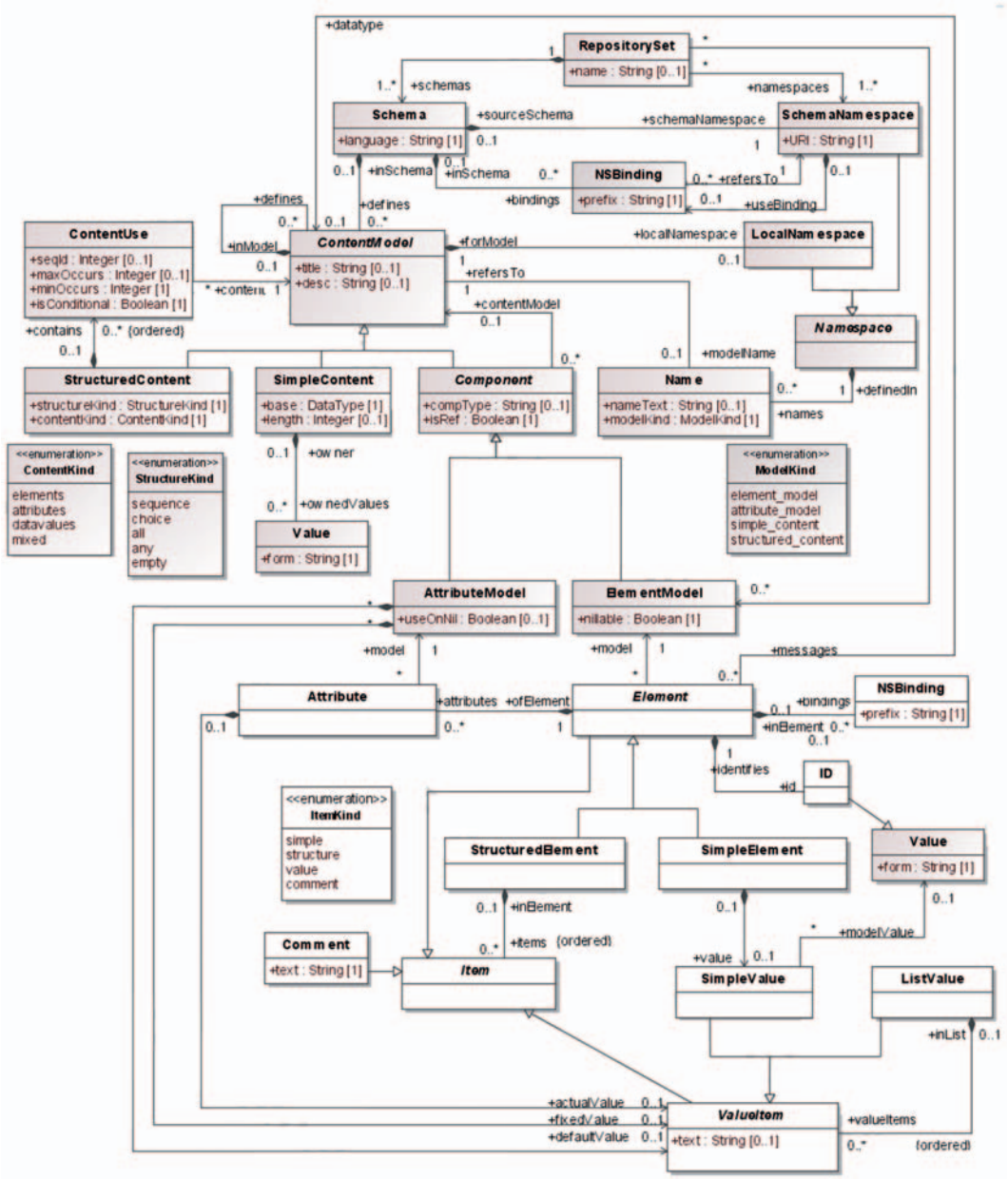

Figure 2. Ontological message metamodel; message-schema metamodel concepts are coded *grey*, while message-instance model concepts are coded *white*.

`SchemaNamespace` or in a `LocalNamespace`. Also, `ContentModels` and `Components` can be defined locally in other `ContentModels` and `Components`. (e.g., XML Schema inner complex types). `RepositorySet` is a set of `Schemas`, for cases when some schemas import other schemas and message definitions. A *message definition* is an `ElementModel` that defines the root message element. *Message-instance metamodel* part defines the concepts typically present in a runtime message, and their association to the modeled schema elements. Every message is a `StructuredElement`. `StructuredElement` contains other elements, either `StructuredElements` or `SimpleElements`, or structures of `ValueItems`. `SimpleElement` contains a `SimpleValue` that is a value treated as atomic in the message definition. `SimpleValues` store the message content, in the `text` attribute. The `model` of runtime `Element` is specified by its `ElementModel`, which provides the element's naming, structural and content properties. In rare cases, such as an XML any-type element, the ElementModel does not actually specify the Element content, and the `datatype` of the `Element` must be specified. Finally, `Elements` may contain `Attributes`, and each `Attribute` is identified by its `AttributeModel`.

Hereafter, when we refer to the MMM individuals that represent a schema or a message, we will use a *message-schema model*, *message-instance model*, or

message model for both. We have developed supporting tools that transform actual schemas and messages to and from the OWL MMM form.

4. Syntactic Ontologization of Message Schemas

The message schema *syntax ontologization* is a design time transformation of the schemas into the corresponding message-schema models in the OWL form. For example, Listing 1 shows the fragments of the DELJIT EDI (unece.org/trade/untdid) specification, while Listing 2 shows fragments of the corresponding OWL message-schema model.

Listing 1 A fragment of the DELJIT EDI specification

```
DELJIT
 UNH - MESSAGE HEADER Segment
 BGM - BEGINNING OF MESSAGE Segment
 DTM - DATE/TIME/PERIOD Segment
 |...
 SG1 - Segment group 1
 | RFF - REFRENCE Segment
 |  C506 - REFERENCE Composite
 |    1153 - Reference code qualifier Data Element
 |    1154 - Reference identifier Data Element
 |    ...
 |   DTM - DATE/TIME/PERIOD Segment
 SG2 - Segment group 1
 | NAD - NAME AND ADDRESS Segment
 | LOC - PLACE/LOCATION IDENTIFICATION Segment
 |    ...
 SG3 - Segment group 3
 |...
 |UNT - MESSAGE TRAILER Segment
```

`RepositorySet` individual `repSet` holds DELJIT message-schema model, which is captured by the `deljit` individual of the `Schema` concept. The root DELJIT element definition (message definition) is transformed into the `elementModel2` individual of the `ElementModel` concept. Its `contentModel` is captured by the `structuredContent1` individual that `defines` and `contains` the `elementModel2` individual, which represents the definition of the DELJIT.SG1 segment group. By reading the Listing 2 in this way, we see that `elementModel2` individual contains `elementModel3` that represents the DELJIT.SG1.RFF reference segment, which contains `elementModel4` that represents DELJIT.SG1.RFF.C506 composite element definition, and so on. The `elementModel3`, `elementModel4`, `elementModel5`, and `elementModel6` individuals are defined in the DELJIT schema namespace, which is captured by `:schemaNamespace1` individual, and they are reusable building components for the other EDIFACT messages.

5. Syntactic Ontologization of Messages

The message syntax ontologization is the runtime transformation of the messages into the corresponding OWL message-instance models. Listing 4 shows the fragment of the message-instance model that corresponds to the fragments of the DELJIT EDI message given in Listing 3.

Listing 2 A fragment of the DELJIT message-schema model in RDF TURTLE (www.w3.org/2007/02/turtle/primer) syntax

```
@prefix p1: <http://local/MessageMetamodel.ecore#>.
:repSet  rdf:type p1:RepositorySet ;
        p1:schemas :deljit;
        p1:message :elementModel1
        p1:namespaces :schemaNamespace1 .

:deljit rdf:type p1:Schema ;
       p1:defines :elementModel1, .. ;
       p1:schemaNamespace :schemaNamespace1.

:schemaNamespace1 rdf:type p1:SchemaNamespace ;
      p1:URI  http://www.unece.un.org/EDIFACT/D05B
      ^^xsd:string ;

:elementModel1  rdf:type p1:ElementModel ;
      p1:contentModel :structuredContent1 ;
      p1:defines :structuredContent1;
      p1:inSchema :syncBODSchema;
      p1:modelName :name1 .

:name1  rdf:type p1:Name ;   p1:definedIn
:schemaNamespace1 ;

:structuredContent1  rdf:type p1:StructuredContent ;
      p1:contains :cu1,:cu2, :cu3 ... ;
      p1:defines :elementModel2 ;  p1:inSchema :deljit ;
      p1:structureKind :sequence ;
      p1:contentKind :elements ;
      p1:localNamespace :localNamespace1 .

:cu1  rdf:type p1:ContentUse ;
      p1:content :elementModel2 ;
      p1:seqId "1"^^xsd:int .

:elementModel2  rdf:type p1:ElementModel ; # SG1
       p1:contentModel :structuredContent2 ;
```

⇨

```
...
:structuredContent2  rdf:type p1:StructuredContent ;
      p1:contains :cu5,:cu6, ; p1:inSchema :deljit ;

:cu5  rdf:type p1:ContentUse ;
       p1:content :elementModel3 ; ...

:elementModel3  rdf:type p1:ElementModel ; # RFF
      p1:contentModel :structuredContent3 ;
      p1:nameText "DELJIT"^^xsd:string ;

:structuredContent3  rdf:type p1:StructuredContent ;
      p1:contains :cu8, ;

:cu8  rdf:type p1:ContentUse ;
      p1:content :elementModel4; ...

:elementModel4  rdf:type p1:ElementModel ; # C506
      definedIn :schemaNamespace1
      p1:contentModel :structuredContent4 ;

:structuredContent3  rdf:type p1:StructuredContent ;
      p1:contains :cu10, :cu11, ;

:cu10  rdf:type p1:ContentUse ; p1:content :elementModel5 ;
      p1:seqId "1"^^xsd:int .

:cu10  rdf:type p1:ContentUse ; p1:content :elementModel6 ;
      p1:seqId "1"^^xsd:int .

:elementModel5  rdf:type p1:ElementModel ; # 1154
       definedIn :schemaNamespace1
       p1:contentModel :simpleContent1, p1:base :string;

:elementModel6  rdf:type p1:ElementModel ; # 1153
       definedIn :schemaNamespace1
        p1:contentModel :simpleContent1,
        p1:base :enumeration;
...
```

Listing 3 The fragment of the DELJIT EDI message

```
UNA:+.?*'
UNB+UNOC:…
UNH+msg-refnum-abc+DELJIT:D:05B:UN…
BGM+242:::DELJIT+…+5'
DTM+137:20070709120000:204'
RFF+CT:2008-GJ4007'
RFF+ADZ:38-058941500'
NAD+BY+7656615::16++General Motors.+…+..
…
NAD+SF+498994588::16++Acme Auto Supplier GMBH+…
NAD+ST+964475835::16++Any Logistics Mngt, Inc+…
…
FTX+AAR+++…
FTX+AAA+++Cylinder Heads'
…
UNT+31+msg-refnum-abc'
UNZ+1+interchg-ref'
```

Listing 4 The fragment of the OWL DELJIT message-instance model

```
@prefix p1:      <http:///MessageMetamodel.ecore#> .
…
:structuredElement1 #captures DELJIT message
      rdf:type p1:StructuredElement ;
      p1:items :structuredElement2 ;
      p1:model :elementModel11 .
:structuredElement2 # captures SG1 segment group
      rdf:type p1:StructuredElement ;
      p1:items :structuredElement3 ;
      p1:inElement :structuredElement1
      p1:model :elementModel21 .
:structuredElement3 # captures RFF segment
      rdf:type p1:StructuredElement ;
      p1:items :structuredElement4 ;
      p1:inElement :structuredElement2
      p1:model :elementModel33 .
:structuredElement4 # captures C506 composite
      rdf:type p1:StructuredElement ;
      p1:items :simpleElement1, simpleElement2 ;
      p1:inElement :structuredElement3
      p1:model :elementModel44 .
:simpleElement1 # captures 1154 element
      rdf:type p1:SimpleElement ;
      p1:value :value1 ;
      p1:inElement :structuredElement4 ;
      p1:model :elementModel55 .
:value1 rdf:type p1:SimpleValue ;
      p1:text "2008-GJ4007"^^xsd:string .
:simpleElement2 # captures 1153 element
      rdf:type p1:SimpleElement ;
      p1:value :value2 ;
      p1:inElement :structuredElement4 ;
      p1:model :elementModel66 .
:value2 rdf:type p1:SimpleValue ;
       p1:text "CT"^^xsd:string . …
```

DELJIT root element is transformed into the `structuredElement1` individual of the `StructuredElement` concept. Importantly, the model *links* the `structuredElement1` individual with its naming, structuring, occurrences and value representation definition, which is captured by the `elementModel1` of the DELJIT message-schema model. All the individuals here are in fact RDF resources, so we use `rdf:id` to identify them. Further, `structuredElement1` has the `structuredElement2` item. Similarly, `structuredElement2` has `structuredElement3` that has `structuredElement4` item. Finally, `structuredElement4` has two `SimpleElement` individuals that actually capture the message data.

6. Semantic Reconciliation of B2B Messages

We developed a reconciliation rules creation tool that provides a graphical environment to visualize OWL message-schema models and OWL reference ontologies as well as several predefined reconciliation rules templates. The tool produces executable reconciliation rules in the Jena (http://jena.sourceforge.net/) language as Jena provides for the rule-reasoning over OWL and RDF documents. An engineer uses only the graphical environment (GUI) to formulate the rules and Jena expertise is not required. In Section 7, when we explain message semantics annotation, we will outline an approach that uses annotation expressions as a knowledge base from which most of the reconciliation rules are derived. In [8] we provided detail theoretical discussion of the message metamodel-based semantic reconciliation.

6.1 Demonstration and Prototype Implementation

Consider a business process that is managed by the Shipment Schedule message that regulates the flow of goods from the supplier to the customer to facilitate inventory management. Assume the customer's message interface is DELJIT EDI based, while the supplier's message interface is OAGIS SyncShipmentSchedule XML based. Both standards define the Shipment Schedule message. Besides syntactical, there are terminological, structural and representational mismatches between them. For example, DELJIT specifies 'DocumentReference' element as 'Reference Identifier' in the [DELJIT/SG1/RFF/C506/1154] data element's structure, as shown in Figure 3. On the other side, the SyncShipmentSchedule specifies 'DocumentReference' as a 'DocumentID' in the [SyncShipmentSchedule/DataArea/ShipmentSchedule/ShipmentScheduleHeader/DocumentReference/DocumentID/ID] structure. To enable interoperable DELJIT-to-SyncShipmentSchedule

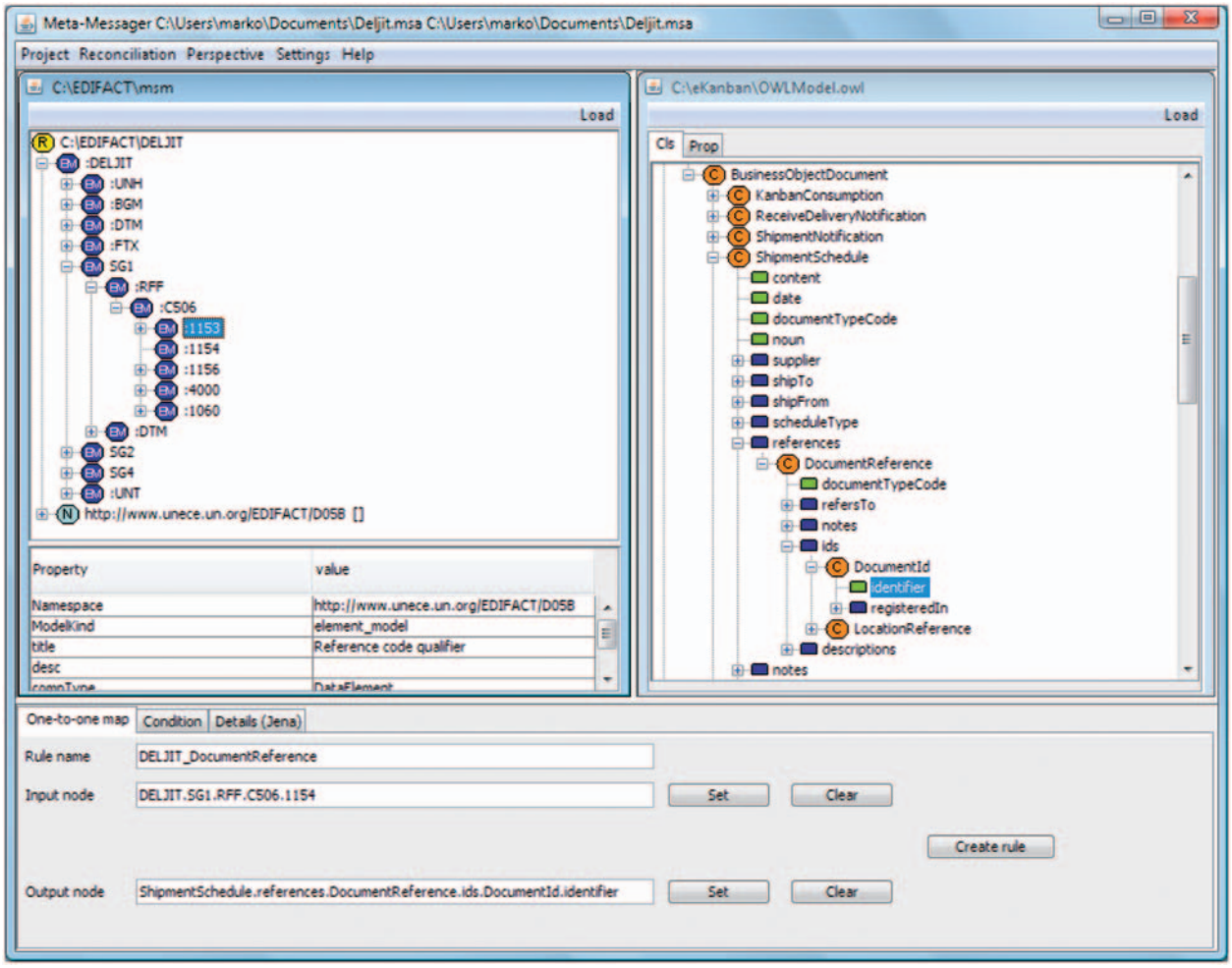

Figure 3. The tool in a 'reconciliation-forward' mode for DELJIT model.

message exchange, we employed the proposed semantic-mediation. As a reference ontology we used the eKanban Ontology [9] that provides a common conceptual data model for the Shipment Schedule.

6.2 Design time

First, we transformed the schemas into the DELJIT and SyncShipmentSchedule OWL message-schema models, respectively. Then, we used the design time tool to create forward reconciliation rules for the DELJIT and backward reconciliation rules for SyncShipmentSchedule message.

Figure 3 shows the tool in a 'reconciliation-forward' mode for DELJIT message-schema model. An engineer identifies DELJIT message path(s) leading to the message content, then identifies corresponding reference ontology concept(s) that message path(s) map to, and finally instantiates a rule template. For example, we created the Map rule for the reconciliation of the [DELJIT/SG1/RFF/C506/1154] element into the [ShipmentSchedule/references/DocumentReference/ids/DocumentId/identifier] reference ontology entity. Also, a condition, shortened as [DELJIT/SG1.RFF/C506/1153='CT'], was applied to this rule to precisely specify that the mapping executes only when DELJIT 'Reference Code Qualifier' element '1153' has value 'Contract Number', which is coded as a 'CT'. Listing 5 illustrates the automatically generated, executable Jena rule. So far, our implementation provides several reconciliation rule templates; One-to-one/Map, One-to-Many/Split, Many-to-One/Merge, SetValue and Convert.

The engineer creates backward rules for SyncShipmentSchedule in the same manner as DELJIT forward rules. The tool in a 'reconciliation-backward mode' renders message-schema model on the right and a reference ontology on the left panel. For example, we created the backward Map for the reconciliation of the [ShipmentSchedule/references/DocumentReference/ids/DocumentId/identifier] ontology entity into the

[SyncShipmentSchedule/DataArea/ShipmentSchedule/ShipmentScheduleHeader/DocumentReference/DocumentID/ID] element.

The tool supports automated generation of most of the backward rules for a particular message-schema model from forward rules of that message-schema model, by the rules inversion. This significantly reduces the effort needed to create all the required rules.

6.3 Runtime

The semantic-mediator orchestrates the runtime message transformations. First, the *EDIFACT-to-MessageInstanceModel* tool transforms a DELJIT message into the DELJIT OWL message-instance model. Then, the Jena rule engine executes the forward reconciliation rules on the DELJIT OWL message-instance model, which produces eKanban ontology individuals. Afterwards, the rule engine executes the backward reconciliation rules on those ontology individuals, which produces a SyncShipmentSchedule OWL message-instance model. Finally, from that model, the *MessageInstanceModel-to-XML* transformer generates a schema-conformant SyncShipmentSchedule message. Listings 3 and 4, and 6–8, respectively, show the fragments of intermediate message forms when reconciling between the DELJIT and SyncSipmentSchedule.

Listing 5 #DELJIT_DocumentReference Rule

```
(?s.201 rdf:type m:StructuredElement)  (?s.201 m:model d:elementModel11)
(?s.201 m:items ?s.201.d.1) (?s.201.d.1 m:inElement ?s.201) (?s.201.d.1 rdf:type m:StructuredElement)
(?s.201.d.1 m:model d:elementModel2) (?s.201.d.1 m:items ?s.205) (?s.205 m:inElement ?s.201.d.1)
(?s.205 rdf:type m:StructuredElement)  (?s.205 m:model d:elementModel3) (?s.205 m:items ?s.344)
(?s.344 m:inElement ?s.205) (?s.344 rdf:type m:StructuredElement)  (?s.344 m:model d:elementModel4)
(?s.344 m:items ?s.346)(?s.346 m:inElement ?s.344) (?s.346 rdf:type m:SimpleElement)
(?s.346 m:model d:elementModel5) (?s.346 m:value ?v1)(?v1 rdf:type m:SimpleValue) (?v1 m:text ?i_1txt)
--condition
(?s.344 m:items ?s.345)(?s.345 m:inElement ?s.344)(?s.345 rdf:type m:SimpleElement)(?s.345 m:model d:elementModel6)
(?s.345 m:value ?v1000)(?v_1000 rdf:type m:SimpleValue) (?v1 m:text ?i_1000txt)
equal(?i_1000txt , 'CT')
->
Map(?i_1txt ?o_1txt) (ro:ShipmentSchedule rdf:type ro:ShipmentSchedule)
(ro:ShipmentSchedule ro:references ro:ShipmentSchedule.references.DocumentReference)
(ro:ShipmentSchedule.references.DocumentReference rdf:type ro:DocumentReference)
(ro:ShipmentSchedule.references.DocumentReference ro:ids
    ro:ShipmentSchedule.references.DocumentReference.ids.DocumentId)
(ro:ShipmentSchedule.references.DocumentReference.ids.DocumentId rdf:type ro:DocumentId)
(ro:ShipmentSchedule.references.DocumentReference.ids.DocumentId ro:identifier ?o_1txt)
```

Listing 6 Fragment of the eKanban OWL individuals

```
@prefix rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix ro:
<http://referenceOntology.eKanban#> .
….
ro:e1  rdf:type ro:ShipmentSchedule;
      ro:references ro:e2 .
ro:e2 rdf:type ro:DocumentReference ;
      ro:ids ro:e3 .
ro:e3  rdf:type ro:DocumentId ;
      ro:identifier "2008-GJ4007"^^xsd:string
.
...
```

Listing 7 Fragment of the SyncShipmentSchedule message-instance model

```
@prefix p1: <http://local/MessageMetamodel.ecore#>.
@prefix msm: http://local/ShipmentScheduleMSM.owl#
...
:e1 #aiag:SyncShipmentSchedule
      rdf:type p1:StructuredElement ;
      p1:items :e2, … ;
      p1:model msm:elementModel1 .
:e2 #aiag:DataArea
      rdf:type  p1:StructuredElement ;
      p1:inElement  :e1 ;
      p1:items :e3, … ;   p1:model msm:elementModel2.
:e3 #aiag:ShipmentSchedule
      rdf:type p1:StructuredElement ;
      p1:inElement  :e2 ;
      p1:items  :e4, … ;
      p1:model  msm:elementModel3  .
:e4 #aiag:ShipmentScheduleHeader
      rdf:type p1:StructuredElement ;
      p1:inElement :e3 ;
      p1:items :e10, … ;
      p1:model msm:elementModel4.
:e10 #oa:DocumentReference
      rdf:type p1:StructuredElement;
      p1:inElement :e4 ;
      p1:items :e20, …;
      p1:model  msm:elementModel21 .
:e20 #oa:DocumentID
      rdf:type p1:StructuredElement ;
      p1:inElement  :e10 ;
      p1:items:e30, …;
      p1:model msm:elementModel22;
:e20 #oa:ID
      rdf:type p1:SimpleElement ;
      p1:inElement  :e20 ;
      p1:model msm:elementModel22;
      p1:items:a30, …; // attributes
      p1:value   :v1 .
:v1  rdf:type p1:SimpleValue ;
     :text    "2008-GJ4007"^^xsd:string ;
  ...
```

Listing 8 Fragment of the SyncShipmentSchedule XML message

```
<?xml version="1.0"?>
<aiag:SyncShipmentSchedule
xmlns:oa=
http://www.openapplications.org/oagis/9
xmlns:aiag="
http://www.openapplications.org/oagis/9/aiag/2
">
<aiag:ApplicationArea>..
  <aiag:DataArea>  …
    <aiag:ShipmentSchedule>  …
      <aiag:ShipmentScheduleHeader> …
        <oa:DocumentReference>
          <oa:DocumentID>
          <oa:ID…>2008-GJ4007</oa:ID>
           </oa:DocumentID>
          </oa:DocumentReference> …
      </aiag:ShipmentScheduleHeader> …
    </aiag:ShipmentSchedule> …
  </aiag:DataArea> …
</aiag:SyncShipmentSchedule>
```

7. Semantic Annotation of B2B Message Schemas

The semantic annotation is also performed on the OWL message-schema models. Formal and machine-processable annotation expressions link the message component definitions captured in OWL message-schema models with the reference ontology entities. Using the annotation expressions, one can recognize, assisted by an annotations-interpretation tool, when two message components represent the same business entity despite their different descriptions in respective message schemas.

7.1 The Semantic Annotation Process

We developed a Message Semantics Annotation Ontology (MSAO) that defines concepts for the annotation of OWL message-schema models with reference ontologies entities. In fact, the MSAO is a closed vocabulary of annotation expressions in the OWL form. Figure 4 illustrates the MSAO concepts.

The `Annotation` concept represents annotation expressions. It has two important properties: (1) `annotatedEntity` that connects to the `MMMEntity` concept, which encapsulates OWL message-schema model individuals (content models, components, and elements/attributes); (2) `annotater` that connects to the `OntologyEntity` concept, which encapsulates OWL ontology concepts, data types, properties, and individuals. `Annotation` can be either a simple `SingleAnnotation` or more complex `MultyAnnotation` that groups `Annotations`. Annotations are either `global` or `local`, as message schema defines elements globally as reusable components, or defines them locally in other elements. For example, to annotate the EDIFACT '1154' data element semantics we established a `global SingleAnnotation` expression as shown in Listing 9. It formally expresses semantics of the '1154' element in terms of the eKanban ontology *as an identifier, which is a datatype property of Document concept that is a domain of an object property ids of DocumentReference concept*; this is shortened as [DocumentReference/ids/Document/identifier] path expression. Further, `Annotation` may have a reconciliation `Functor` that specifies the content transformation from a message element into ontology individual(s), and this is crucial for the runtime reconciliation. So far, we provided support only for the transformation cases that occur most often Map, Merge, Split, Convert and SetValue. For the EDIFACT '1154' the Functor is `Map`. The functors such as currency or unit measure conversions could be provided as Web Services. Each `Annotation` can have `alternatives` annotations; this is to express the cases when a message component semantics is conditional. For example, if EDIFACT '1035' element has value 'ST', the 'NAD' segment semantics is ShipToParty; if the value is 'SF', the NAD semantics is ShipFromParty.

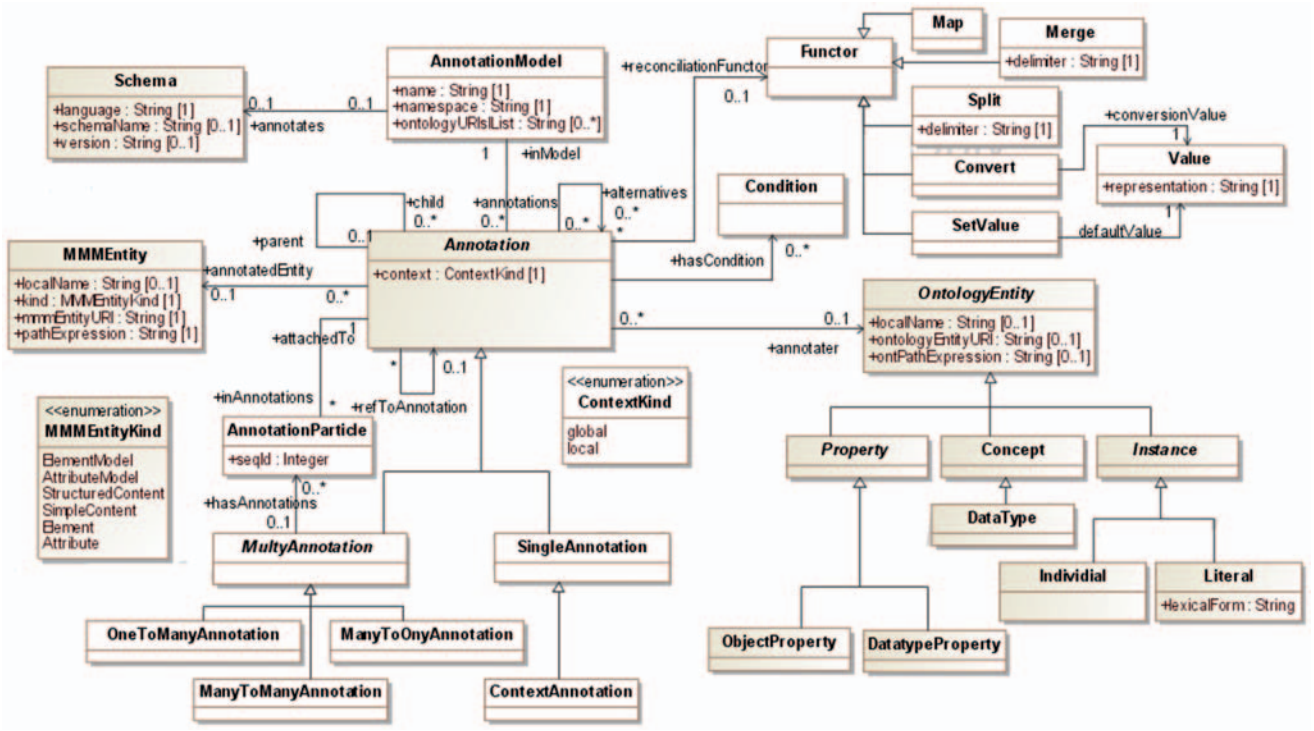

Figure 4. Message semantics annotation metamodel.

Listing 09 The single semantic annotation expression example in OWL RDF/XML syntax (some fragments of the expression are omitted for clarity). The expression is produced by the tool.

```
<ammo:SingleAnnotation rdf:ID="a1">…
  <ammo:context…> global</ammo:context>
  <ammo:annotatedEntity>
    <ammo:MMMEntity rdf:ID="ae>
     <ammo:mmmEntityURI …>
      file:/C:/EDIFACT/DELJIT_MSM.OWL#elementModel55 </ammo:mmmEntityURI>
      <ammo:pathExpression …> deljit/elementModel55 </ammo:pathExpression>
      <ammo:kind rdf:resource="mmm:ElementModel" />
      <ammo:localName …>1154</ammo:localName>
    </ammo:MMMEntity>
  </ammo:annotatedEntity>
  <ammo:annotater>
   <ammo:DatatypeProperty rdf:ID="atr1">
    <ammo:ontologyEntityURI …> http://nist.gov/amis2/eKanban#identifier </ammo:ontologyEntityURI>
    <ammo:localName …> identifier </ammo:localName>
    <ammo:ontPathExpression …>DocumentRefernce.ids.DocumentId.identifier </ammo:pathExpression>
   </ammo:DatatypeProperty>
  </ammo:annotater>
</ammo:SingleAnnotation>
```

The annotation process is highly human intensive and requires expert knowledge in the business domain and message specifications. When annotating, an annotator performs several steps: (1) message elements identification; (2) their intended meaning identification; (3) corresponding reference ontology entities identification; and (4) annotation expressions definition to specify the message elements meaning. Our tool supports the annotation steps, hides MSAO complexity, and exports the annotations into the OWL.

7.2 Demonstration and Implementation Status

Consider the case of the OAGIS schemas. The OAGIS schemas define reusable message components of three categories: (A) basic context free, such as a Name and NameType; (B) aggregated context free, such as PartyBaseType or LocationBaseType, that contain the basic or other aggregated components; and (C) business-context-specific components, such as ShipToParty, derived from the B category. Messages are defined by using reusable and additional message-specific components. The UBL (Unified Business Language; www.oasis-open.org/committees/ubl/) and EDI specifications utilize a similar methodology. We implemented a common annotation method, comprised of the following three steps, for all these specifications:

I. *Annotate context-free message components*. Annotation progresses from `simpleContents`, `attributeModels`, simple `elementModels`, `structuredContents`, to structured `element Models` message components. For example, by using the tool (Figure 5), the annotator identifies PartyBaseType in the SyncShipmentSchedule message-schema model, and then identifies a related eKanban ontology entity, which is the Party. Then, he adds a single annotation in the tool's Annotations table and populates 'Message Entity' and 'Ontology Entity' table fields, by *dragging-and-dropping* identified correspondents. That creates the `singleAnnotation` expression. Similarly, the annotator annotates PartyBaseType's containments. The annotator assigns a functor to the annotations. In this step, all reusable components are annotated; this will provide annotations reusability as we describe in the next step.

II. *Annotate context-specific components*. An annotator performs similarly as in the step I; however, the annotator now can reuse the annotations of context-free components that the context-specific components are based on. The tool supports the annotator when such a reuse case occurs. For example, the annotator annotates ShipToParty `ElementModel` with the ShipToParty ontology concept. As the annotator proceeds with the ShipToParty containments annotation, the tool notifies that the base model of the ShipToParty component – PartyBaseType – was already annotated and that reuse is possible for all the contained elements. The annotator decides either to reuse the annotations or to annotate the containments step-by-step. If he opts for reuse, the annotator specifies/refines a subpath of the ontology path that leads to the corresponding ontology entity; for the ShipToParty message element a refinement is 'add ShipToParty.participant before the Party'. Finally, the tool automatically generates annotations for the ShipToParty containments by reusing the PartyBaseType annotations. Annotations in the rows 5 and 6 in the annotation table are automatically created from annotations in rows 2 and 3.

III. *Annotate a message definition*. Annotation proceeds from the root `elementModel`, which defines a message. Tool identifies the 'lower-level' message elements that are derived from reusable components, and offers to the annotator to reuse the annotations from (I) and (II) above. Such annotation expression reusability is a very important aspect as it reduces the time/effort for the message definition annotation. The annotator must annotate all the `elementModels` and `attributeModels` of message components that carry data, and to assign `functor` to their respective annotations. This is to prevent a message content loss during the semantic reconciliation, which happens if particular component definition is not annotated and reconciliation rules are derived from the annotations. Finally, the tool is used to generate the Jena reconciliation rules from the MSAO annotation expressions.

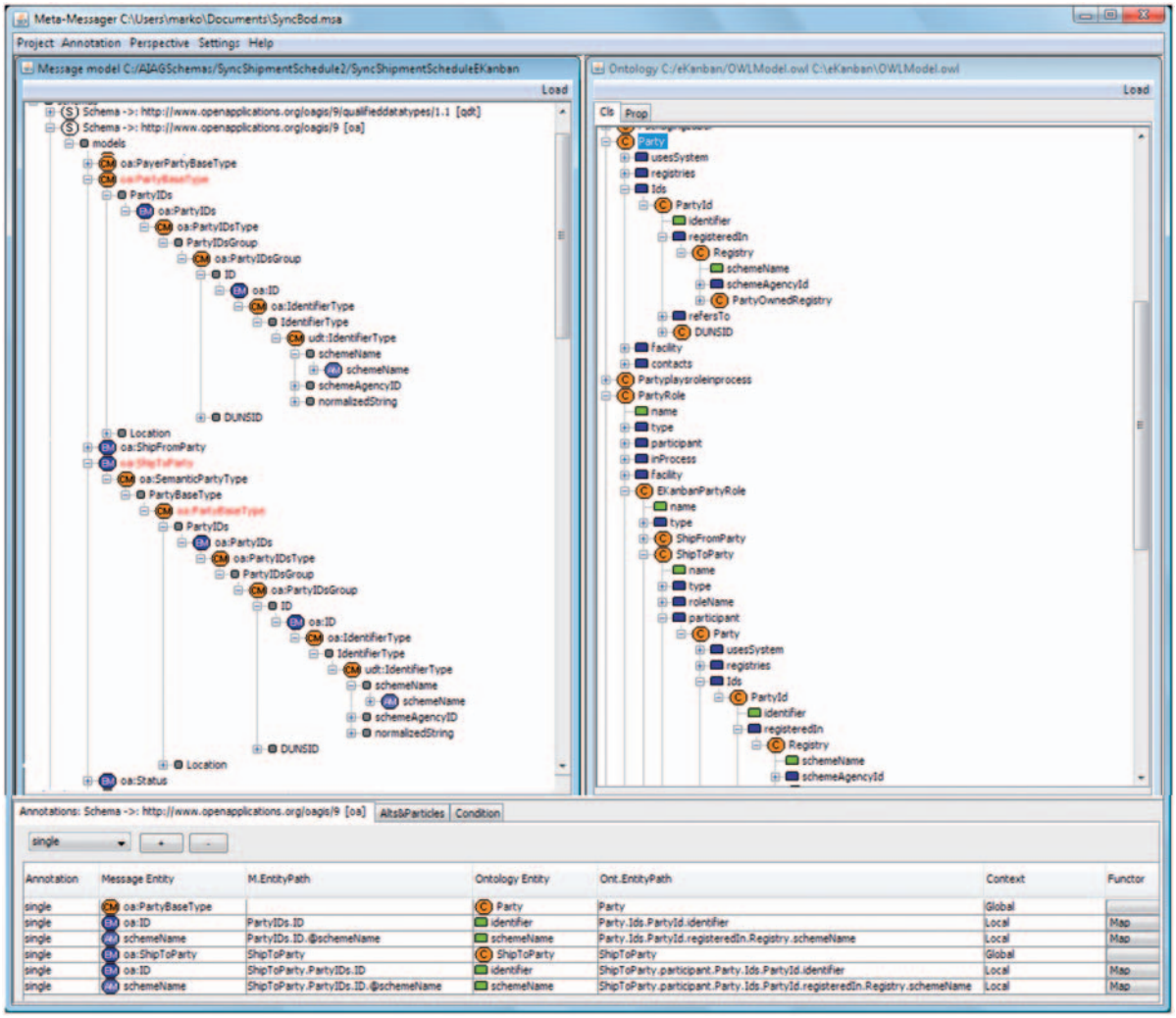

Figure 5. The tool in a semantic annotation mode. Illustrated is PartyBaseType and ShiptToParty annotation.

The initial experiment showed that a rule-generation algorithm is capable to generate reconciliation rules from the annotation expressions. As we still work on that algorithm, in Section 6 we chose to demonstrate reconciliation by directly creating the reconciliation rules. Importantly, there may be message transformation cases that annotation expressions cannot cope with. For example, if DELJIT document has no status element as the assumed default status is 'Authorized', then such message-to-ontology relationship cannot be described by annotations, and the integration engineer creates DELJIT forward rule that sets status to 'Authorized'. For such cases, an integration engineer uses the tool in both modes, the semantics annotation and reconciliation-rules creation.

8. Overview of Related Approaches

Alternative semantic-mediation models have been demonstrated before. Anicic et al. [10] demonstrated an any-to-any model that, first, merges local OWL ontologies (LCMs), and then classifies and transforms source ontology individuals using a description-logic (DL) reasoner into the target ontology individuals. The Artemis integration framework [11] used the OWLmt ontology mapping tool (http://sourceforge.net/ projects/ owlmt) to demonstrate crosswise mappings among local OWL ontologies. The any-to-any models employ no reference ontology as the mediation point; that increases the number of crosswise mappings or the size and complexity of the merged ontology. Oppositely, the any-to-one models, such as ours, employ reference ontologies and reduce the number of mappings. The Harmonise project [12] demonstrated an any-to-one model. In Harmonise, the Mafra tool (http://sourceforge.net/projects/mafra-toolkit) provided for mappings between RDFS ontologies and RDF-to-RDF documents transformation. The ATHENA project [13] introduced several semantic-mediation tools: Astar for RDFS concepts annotation; Argos for RDF-to-RDF documents reconciliation specification; and Ares for

RDF-to-RDF reconciliation execution. Astar annotations are OWL DL TBox axioms. To ontologize X12 EDI schemas (www.x12.org/), Foxvog and Bussler [14] defined ontological metamodel specific to the X12 constructs only. Yarimagan and Dogac [15] introduced a Component Ontology for UBL schemas. Yarimagan's tool transforms UBL schemas into Component Ontology TBox form, and reconciles messages of different customized UBL schemas by DL reasoning, similar to [10]. The MWSAF [16] provides W3C's SAWSDL (Semantic Annotations for XML Schema; www.w3.org/TR/sawsdl) annotation framework that defines XML Schema attributes (that 'add' semantics to XML components definitions) and associates them with executable procedures that translate XML documents to and from ontology individuals. Our work differs from Anicic et al. [10], Bicer et al. [11], Fodor and Werthner [12], and Berre et al. [13] in that we use the ontological MMM form for semantic-mediation activities. The DL reasoning used in Anicic et al. [10] and Yarimagan and Dogac [15] is incapable of nontrivial relationships computation, such as literal data conversions, while the rule-based reasoning, used in this work, supports such nontrivial transformations. In contrast to Foxvog and Bussler [14], Yarimagan and Dogac [15], and Patil et al. [16], our approach provides for semantic annotation and reconciliation of any message representations.

9. Industrial Perspective

We see a potential for use of our architecture in small and medium-size enterprises (SMEs). The SMEs usually have limited resources and insufficient knowledge to understand and implement industrial standard schemas (OAGIS, EDIFACT, etc.) into their e-business applications, which can be either developed in-house or bought from the application providers. The problem arises when each dominant partner that collaborates with the particular SME mandates a particular standard. For example, the US Walmart company mandates EDI-based messaging for the data exchange with its suppliers [17]. For that reason, instead of implementing and mapping several different standards that dominant partners mandate, the SMEs could benefit from our architecture, which for the SMEs means only one mapping and adoption of a single reference ontology. However, there are three requirements. *First*, reference ontologies for the business communities must be publicly available and provided to the SMEs. We hope that standards development organizations (SDOs) and business experts will work together to build such reference ontologies. Unquestionably, there will be a significant challenge in terms of needed expertise, ontology development tools, ontology evolution, and in getting the consensus of participants. *Second*, SMEs must annotate the proprietary message schemas used in their applications, and must create all the reconciliation rules for the proprietary schemas. We state, from our experience, it will be much easier for the SMEs to deal with their (familiar) proprietary schemas and reference ontology entities, than with the several different standard message specifications and implementation guides. *Third*, after the SDOs accept the reference ontologies as a common message conceptualization, they must annotate the standard schemas, by following the proposed annotation steps. Also, the SDOs must provide the reconciliation rules for standard messages publicly. Finally, the SMEs should configure the semantic mediator to execute their proprietary-message and provided standard-message reconciliation rules, according to the particular B2B scenario. At runtime, the semantic mediator will transform SME proprietary messages to and from the standard messages – in a manner similar to the one we demonstrated for the DELJIT-to-SyncShipmentSchedule scenario. For SMEs, such an infrastructure will provide for message standards-compliant applications and flexibility to the B2B partners change. However, to build such an infrastructure, primarily the SDO side will have to resolve several organizational and technical challenges, from the agreement on the business ontologies and their use, a repository that stores, manages and provides access to public standard message-schema models, annotations and reconciliation rules, to the semantic mediators as online software services.

10. Conclusion

Our successful demonstration was indeed small scale, but based on two real industrial schemas. For that reason, we believe the prototyped semantic-mediation tools would support the real e-business requirements, although the needed infrastructure could take several years to implement. We now plan to extend our approach towards context-taxonomy semantic annotation of the message components. This extension will add to the efficiency of our approach by providing a foundation for more efficient and scalable schema customization and development, based on the reuse of message components that are semantically described by ontologies and context-taxonomies, and categorized and accessed by reasoning tools. The component reuse is crucial for the B2B integration as it reduces different schema heterogeneities. Similarly, the semantically-managed access and reuse could be applied in an enterprise knowledge management context. Enterprise ontologies should be developed and then used to formalize the informal enterprise knowledge entities, such as organizational, business and technical

documents or best practices, by the semantic annotation. To enable that, we believe the metamodel of various enterprise entities as well as respective annotation vocabularies need to be developed, in a similar fashion to the Message metamodel and MSAO vocabulary. In the enterprise knowledge context, our tool could provide for XML-based enterprise document annotation.

Disclaimer: Certain companies or commercial and open source software products are identified in this article. This use does not imply approval or endorsement by NIST, nor does it imply that these products are necessarily the best available for the purpose.

References

1. Snack, P. (2007). Standards-based Interoperability: The Road Ahead, *AIAG Actionline*, July/August 2007, pp. 21–29
2. Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specification, *Knowledge Acquisition*, **5**(2): 199–220.
3. Ray, S. and Jones, A. (2006). Manufacturing Interoperability, *Journal of Intelligent Manufacturing*, **17**(6): 681–688.
4. Vujasinovic, M., Ivezic, N., Kulvatunyou, B., Barkmeyer, E., Missikoff, M., Taglino, F., Marjanovic, Z. and Miletic, I. (2010). Semantic Mediation for Standard-based B2B Interoperability, *IEEE Internet Computing*, **14**(1): 52–63.
5. International Standards Organization (2002). Abstract Syntax Notation One, ISO/IEC 8824-1:2002.
6. International Standards Organization (2004). Clear Text Encoding of the Exchange Structure, ISO/EIC 10303-21:2004.
7. Vujasinovic, M. and Barkmeyer, E. (2009). The Message Metamodel, NIST Internal Report, USA.
8. Vujasinovic, M., Barkmeyer, E., Ivezic, N. and Marjanovic, Z. (2010). Interoperable Supply-chain Applications: Message Metamodel-based Semantic Reconciliation of B2B Messages, *International Journal of Cooperative Information Systems*, **19**(1&2): 31–69.
9. Barkmeyer, E. and Kulvatunyou, B. (2006). Ontology for the e-Kanban Business Process, NIST Internal Report 7404, USA.
10. Anicic, N., Marjanovic, Z., Ivezic, N. and Jones, A. (2007). Semantic Enterprise Application Integration Standards, *International Journal of Manufacturing and Technology*, **10**(2–3): 205–226.
11. Bicer, V., Laleci, G.B., Dogac, A. and Kabak, Y. (2005). Artemis Message Exchange Framework: Semantic Interoperability of Exchanged Messages in the Healthcare Domain, *SIGMOD Record*, **34**(3): 71–76.
12. Fodor, O. and Werthner, H. (2004). Harmonise: A Step Toward an Interoperable E-Tourism Marketplace, *International Journal of Electronic Commerce*, **9**(2): 11–39.
13. Berre, A., Elvesæter, B., Figay, N., Guglielmina, C. Johnsen, S., Karlsen, D., Knothe, T. and Lippe, S. (2007). *The ATHENA Interoperability Framework. Enterprise Interoperability II*, New York: Springer, pp. 569–580.
14. Foxvog, D. and Bussler, C. (2005). Ontologizing EDI: First Steps and Initial Experience, In: *International Workshop on Data Engineering Issues in E-Commerce*, Tokyo, Japan, pp. 49–58.
15. Yarimagan, Y. and Dogac, A. (2009). A Semantic-based Solution for UBL Schema Interoperability, *IEEE Internet Computing*, **13**(3): 64–71.
16. Patil, A., Oundhakar, S., Sheth, A. and Verma, K. (2004). Meteor-s Web Service Annotation Framework, In: *International Conference on WWW*, New York, NY, USA, pp. 553–562.
17. Walmart Corporate Suppliers Requirements. Available at: http://walmartstores.com/Suppliers/248.aspx (accessed June 2010).

Marko Vujasinovic

Marko Vujasinovic is currently finishing PhD at Department of Information Systems, Faculty of Organizational Sciences, University of Belgrade. Until recently, he was a guest researcher at the Enterprise Systems Group of the National Institute of Standards and Technology (NIST). At NIST, he participated in several research projects of the supply-chains integration. His research interests are in enterprise application integration and interoperability, semantic web technologies, e-business, and model-driven development. He received his MS degree in computer science from University of Belgrade.

Nenad Ivezic

Nenad Ivezic is a staff researcher at the Enterprise Systems Group of the NIST. He received his MS and PhD degrees from Carnegie Mellon University and BS degree from University of Belgrade. His interests include artificial intelligence, distributed systems, supply chain management, and enterprise systems integration. He is currently working on advanced information systems for supply chain integration and participates actively in standards development for e-business technical specifications.

Edward Barkmeyer

Edward Barkmeyer has an MS degree in Applied Mathematics and 40 years experience in the computer sciences, covering a wide range of topics. Since 1981, Mr Barkmeyer has been a principal analyst and architect in information interchange among manufacturing software systems – engineering, planning, control, and supply-chain operations. He is currently working on automatic translation of business rules to information exchange tests using artificial intelligence methods. Mr Barkmeyer represents NIST on national and international standards bodies in the areas of interface specification, information modeling, process modeling, and data interchange.

Zoran Marjanovic

Zoran Marjanovic is a full professor at Faculty of Organizational Sciences, University of Belgrade, and a founder and president of the Breza Software Engineering company. His research interests are information systems development methodologies, databases, and semantic enterprise application interoperability. Professor Marjanovic is a lead on several on-going projects with government and commercial entities that address design, deployment, and testing of enterprise resource planning and other business systems. He received his MS and PhD degrees in Information Systems from University of Belgrade. He is a member of ACM and IEEE.