INTEROPERABLE SUPPLY-CHAIN APPLICATIONS: MESSAGE METAMODEL-BASED SEMANTIC RECONCILIATION OF B2B MESSAGES

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

[1]*Manufacturing Engineering Laboratory*

National Institute of Standards and Technology

100 Bureau Drive, Gaithersburg, MD 20899, USA

{marko.vujasinovic, edward.barkmeyer, nenad.ivezic}@nist.gov

[2]*Department of Information Systems*

Faculty of Organizational Sciences, University of Belgrade

Jove Ilica 154, 11000 Belgrade, Serbia

marjanovic.zoran@fon.rs

Abstract. Supply-chain applications exchange numerous electronic business-to-business (B2B) messages of varied types. Often, the supply-chain applications need to interact even though the applications use different message specifications and different message-representation standards. The paper discusses an approach to achieve such interactions by using the Message Metamodel-based semantic reconciliation of messages conformant to different message specifications, yet supporting the same B2B scenario. The Message Metamodel is a novel, neutral form for representing B2B messages and their specifications that may be based on different message-representation standards , such as Electronic Data Interchange (EDI), Extensible Markup Language (XML) or Abstract Syntax Notation One (ASN.1). Experimental investigation showed that a semantic reconciliation architecture, discussed in this paper, if enhanced with the Message Metamodel, enables seamless interoperable message exchange in heterogeneous supply-chain environments. The Message Metamodel-enhanced semantic reconciliation architecture supports the reconciliation of B2B messages irrespective of the message exchange standard used and insulates reconciliation activities from the specific physical message-representation syntaxes.

Keywords: semantic reconciliation, business-to-business interoperability, message model, supply-chain integration

1. Introduction

In a typical business-to-business (B2B) scenario, the involved software applications interact with each other by exchanging business documents in a form of electronic messages that carry structured business data. The structure and content of the messages (i.e., message instances) are defined by message specifications (i.e., message schemas). The partners involved in such a business scenario implement respective application's message interfaces either by adopting standard message schemas, such as Universal Business Language (UBL) [3] and Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT) [4] multi-industry standard schemas, or such as Standards for Technology in Automotive Retail (STAR) [1] and Automotive Industry Action Group (AIAG) [2] industry-specific standard schemas, or by developing proprietary message schemas.

The B2B environment is heterogeneous as involved partners adopt different message schemas that use different vocabularies to name the message elements, different structures to organize the data, or different data types to represent the data. Also, the message schemas may differ in the syntax that represent and encode the message instances (e.g., XML [6], EDI [5], ASN.1 [7] syntaxes) which makes the B2B environment even more heterogeneous. Such syntactical, terminological, structural, and data-representational mismatches between message schemas lead to non-interoperable message exchanges between business partners, even though the message schemas are designed for the same B2B scenario.

Such a heterogeneous B2B environment requires a capability to reconcile these mismatches so the applications may exchange message instances regardless of their different specifications and representations. Presently, the prominent architectures for interoperable B2B message exchange use a reference ontology-based semantic reconciliation [10, 11, 12, 13, 23, 34]. The reference ontology is an explicit and formal representation of a set of business concepts and business concept relationships for a specific business domain; moreover, it is a shared vocabulary and a shared conceptual model of the data exchanged between collaborating business partners. The semantic reconciliation is a transformation of the content of message instances from the message instance form used by the sending application to the message instance form expected by the receiving application, using reconciliation rules that are based on a reference ontology as the mediating reconciliation point.

The semantic reconciliation of the mismatches between different message schemas unfolds at both design-time and run-time. The design-time semantic reconciliation is an activity of specifying the transformation of message instance content to and from the reference ontology form, while the run-time semantic reconciliation is the execution of the transformation of the message instance content.

The message instance content transformation may be specified either 1) *directly* – by defining the executable reconciliation rules (also called the mapping rules) that transform data from the terminology and structure of a message instance to the reference terminology and structure (i.e., to instances of the concepts from the reference ontology) and vice-versa; or (2) *indirectly* – by establishing explicit and machine-processable semantic expressions between the message elements and their business meanings as captured in a reference ontology (i.e., by annotating the meaning of message elements using the reference ontology concepts) and, then, by deriving the executable reconciliation rules automatically from the semantic annotation expressions.

Irrespective of whether the direct or indirect approach above is adopted, it is obvious that a model that abstracts the message from underlying specific message-representation syntax is needed to ensure that semantic reconciliation (both design-time and run-time) is independent of the physical message-representation syntax and message specification standards. Thus, the reconciliation rules and semantic annotations are to be defined on such model of a message. The model has to insulate message content transformation activities from the specific physical message-representation syntaxes and to allow the semantic reconciliation software to be reused with different physical message-representation syntaxes.

This paper presents one model of the message that provides a solid base for design-time and run-time semantic reconciliation of business messages. The next section presents the motivation for the research of such message model for semantic reconciliation and discusses issues with current approaches. The third section proposes and describes the Message Metamodel, which is the novel message model for semantic reconciliation. The fourth section describes the application of the Message Metamodel in the semantic-reconciliation architecture and details the activities and steps of the proposed semantic reconciliation. The fifth section presents an experimental scenario and an implementation of the model. The final sections of the paper discuss the related work and offer concluding remarks. Additionally, as detailed later, the implemented toolset and example schemas and messages used in the experimental scenario are available for download, so the reader can reproduce the example results presented in this paper.

2. Research motivation

Our investigation of the current semantic reconciliation approaches [10, 12, 19, 20, 22, 23, 33] showed that these approaches use a local conceptual model of the message elements as the model of the message for the reconciliation purpose.

The local conceptual model is a local ontology created by interpreting semantics of message elements from structural organization of elements given in a message schema and, if available, from message schema naming and design guidelines. This means that a *semantics extraction* software re-engineers the conceptual model of the message elements from message schemas and captures that model using an ontology representation language such as Web Ontology Language (OWL) [8] or Resource Description Framework Vocabulary Description Language (RDFS) [9]. For example, XML Schemas transformed to OWL local conceptual models [32] or to RDFS local conceptual models [25]. Once a local conceptual model of the message is available, the reconciliation rules are defined between the local conceptual model of the message and the reference ontology, and, at runtime, the output of the semantic reconciliation process produces a message as a set of instances of corresponding local conceptual model concepts.

However, a local conceptual model proved insufficient to capture required information about messages to enable effective mapping between syntactic elements and content concepts, which makes it insufficient for reconstructing message schema-conforming message instances from the output of the semantic reconciliation. As the local conceptual model of the message retains only the "semantic gist" of the message, it does not preserve information about the original message structure, elements order, names of message elements, namespaces definitions, data concept granularity (element vs. attribute), and formatting rules. Elsewhere, it was demonstrated and reported that a semantic reconciliation approach that uses a local conceptual model as a model of message needs an elaborate work-around to capture these structural and syntactic details in order to produce message schema-conforming message instance from the local ontology instance [11, 25].

When the local conceptual model represents a model of the message, a message syntax-specific transformation tool creates a local ontology from the message schema (e.g. EDI-to-RDFS, XML-Schema-to-RDFS, or ASN.1-to-RDFS transformations) and also transforms actual message instances to instances of the local ontologies (e.g., XML-to-RDF or EDI-to-RDF transformations) and vice-versa (e.g., RDF-to-XML or RDF-to-EDI transformations). In such syntax-specific transformation tools, the necessary information needed to produce schema-conforming message instances from the reconciliation output is usually maintained by (1) embedding the structural and concept granularity characteristics in naming conventions (e.g., a '*path-name*' naming convention for labeling the concepts of a local ontology and their instances to reflect message-structure definition, or, for an example, by adding *_attr* suffix to a concept name to distinguish attributes from elements), and (2) creating additional reconciliation rules that generate axioms for the reconciled message to carry purely data-representation and formatting rules through the entire semantic reconciliation. Significantly, creating such additional reconciliation rules requires undesired effort on behalf of a rule expert and, moreover, requires additional knowledge about message-representation and formatting rules, beyond understanding the message semantics, as reported in [11].

All these issues are shortcomings of a local conceptual model as a model of the message for semantic reconciliation. Furthermore, depending on a transformation strategy applied to the extraction of local conceptual model from message schemas, the transformation may produce different local conceptual models of the same message schema (as discussed in [25]). That practically means that the rule expert, when creating the rules, may be faced with unfamiliar message structure that is now captured in local conceptual model and that likely differs from the original message structure captured by the message schema.

Although the local conceptual model provides a model of the message that allows the semantic reconciliation software to be reused with different message-representation syntaxes, its insufficiency for reconstructing message schema-conforming message instances from the output of the semantic reconciliation process makes that model inadequate for reconciliation.

That outcome led us to the view that a model of the message, besides capturing the original message structure instead of interpreted one, should be also rich enough to accommodate the syntactic distinctions made in different message-representation standards (i.e., syntactic concepts of messages such are naming, structure, occurrences, and value representation) in order to reconstruct message schema-conforming message instances from the output of the semantic reconciliation. The model of a message should be represented using an ontology language as the ontological representation allows rule-based reasoning and inferencing over the model and its components. Ontological representation of the model of a message actually opens the door to defining semantic reconciliation between the model of a particular message on one side, and a reference ontology on other side, which is specified using an ontological language (e.g., OWL or RDFS).

We have developed a novel model of message that is a concise model for both actual message schemas and message instances that supports the primary syntactic concepts – naming, structure, occurrences, and value representation – and is just rich enough to accommodate the syntactic distinctions made in XML Schema, ASN.1, EDIFACT, North American EDI X12 [38], and EXPRESS [45] (as the model for Clear Text Exchange) standards. The novel model of a message is defined by the Message Metamodel. The Message Metamodel concepts are summarized in the next section and detailed in [18].

3. (Ontological) Message Metamodel

The Message Metamodel captures message structure and message content in a form that intermediates between well established message-representation standards (e.g. XML, ASN.1 or EDI), as shown in Figure 1. The Message Metamodel form is an abstract syntax for the messages, devoid of the specific representation rules for specific syntaxes.

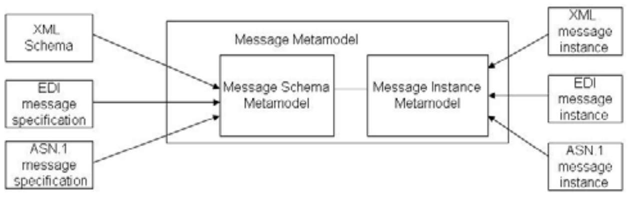


Figure 1 Message Metamodel as an intermediate form for representing messages of different syntactic varieties

Conceptually, the Message Metamodel has two parts: the schema part (the Message Schema Metamodel part), and the message instance part (the Message Instance Metamodel part). The schema part captures the naming, structure, and value concepts that are present in the corresponding message schema language (e.g. XML Schema) The message instance part captures the elements present in each runtime message instance, and their association to the modeled schema elements. (The Message Metamodel is presented in the Unified Modeling Language (UML) notation in Figures 2 and 3, but captured as an OWL ontology as we discuss later).

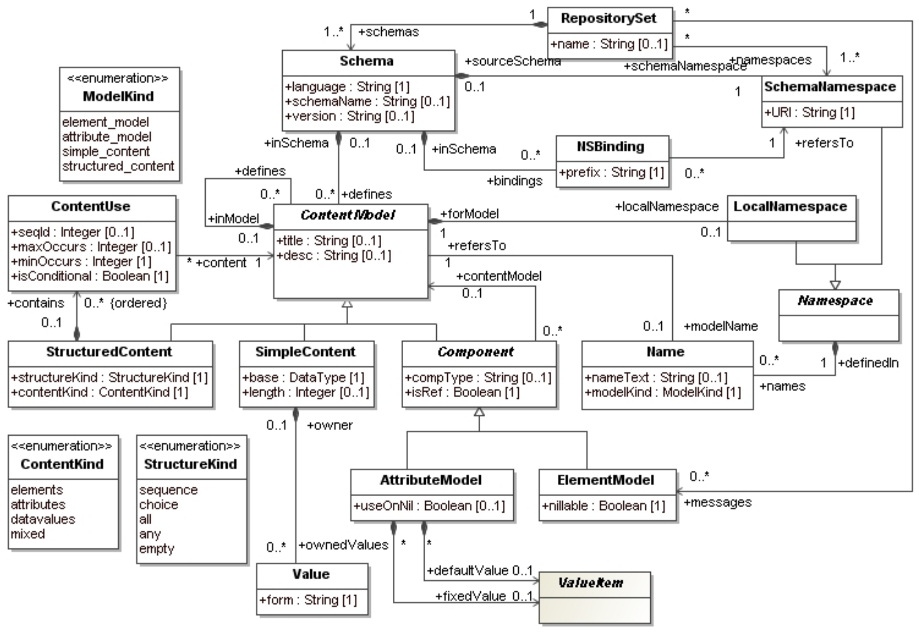


Figure 2 UML representation of Message Schema Metamodel part

The Message Schema Metamodel part defines concepts that are commonly used in the message schema representation standards in order to describe constraints on the message structure and content. The root concept is a `Schema`. `Schemas` define `ContentModels` (`StructuredContents` and `SimpleContents`) and `Components` (`AttributeModels` and `ElementModels`). `StructuredContent` represents a content that contains zero or more other `ContentModels` (e.g., xs:complexTypes, ASN.1 sequence and set types, and the structure of EDI segments and segment groups). `SimpleContents` represent datatypes (e.g., integer, string, date, float, enumeration, identifier, etc) defined or used in a message schema (e.g., xs:simpleType, or EDI data element). If a `SimpleContent` represents an enumeration (or an EDI Code list), it owns `Values`. `Value` represents and stores actual (enumerated or other) value. The `ElementModels` define the elements of a message. The content of an `ElementModel` can be either `StructuredContent` or `SimpleContent` (e.g., in XML Schema, xs:element has a type that is an xs:complexType or an xs:simpleType). The `AttributeModel` defines the element attributes. The `ContentModels` and Components may have names, which is captured by `Name` concept. The `Names` are defined either in a `SchemaNamespace` or locally (`LocalNamespace`). Also, `ContentModels` and `Components` can be defined locally by other `ContentModels` and `Components` (e.g., XML schema inner complexTypes or inner-defined xs:elements). The `RepositorySet` is a collection of schemas and message definitions that are used in a common application and typically refer to each other. A message definition is an `ElementModel` that defines a model of the root element of a message instance.

The Message Instance Metamodel part defines the concepts that are typically present in an actual runtime message instance. We consider that every message instance is an `Element`, more specifically a `StructuredElement`. A `StructuredElement` contains other `Elements` (either `StructuredElements` or `SimpleElements`) or structures of values (`ValueItems`). A `ValueItem` represents an occurrence of a `Value` in a given place (but the `Value` itself may or may not be modeled.) A `SimpleElement` contains a `SimpleValue`, while an empty element of either kind has no content (i.e., no `Items`). A `SimpleValue` is a value treated as atomic in the message definition. Its primary representation is the string that is the `text` attribute. `SimpleValues` store the message content. Every `SimpleValue` has a `SimpleContent` model (i.e., datatype), but it is usually unnecessary to specify it directly at runtime (It is stated in the model of the `Element` that has the `Value`). A `SimpleValue` is an occurrence of a `Value` in a given place, but that may not be specified directly at runtime, either. A `ListValue` is actually a sequence of `SimpleValues` of a given type. The only reason for having this concept is to support the idea that XML Attributes can have fixed or default values that are `ListValues`.

The definition of an `Element` is specified by its `ElementModel` (which provides the name/tag and expected properties). In some cases, the `ElementModel` does not completely (or actually) specify the content of the `Element` instance, and in that case, the `datatype` of the `Element` must be specified. `Elements` may contain `Attributes`, and each `Attribute` is identified by its `AttributeModel` (which owns the `Name`). `Attributes` may have assumed values, specified by the `AttributeModel`, even when they are not physically present.

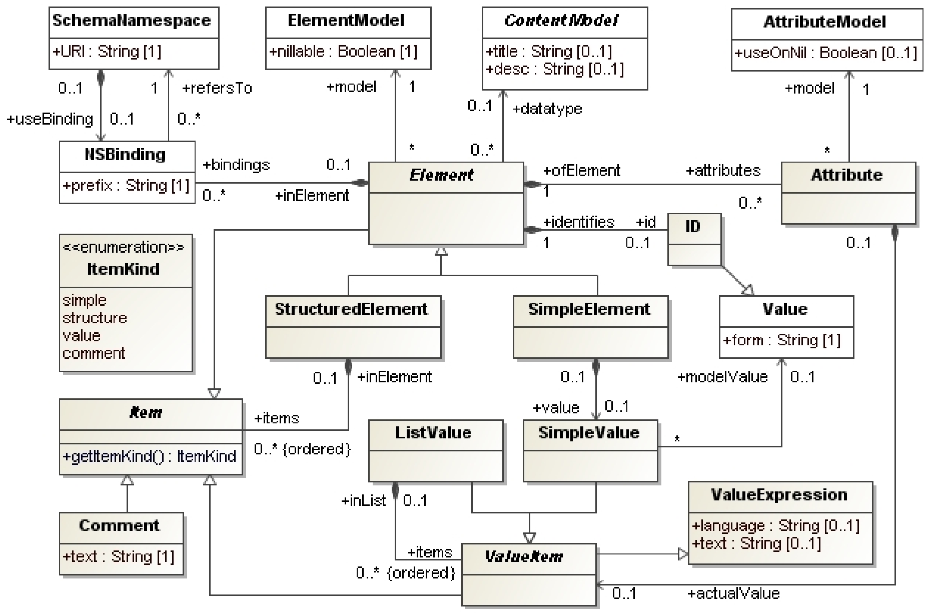


Figure 3 UML representation of Message Instance Metamodel part

A particular message schema information is captured as an instance of the Message Schema Metamodel while a particular runtime message instance and its content are captured as an instance of the Message Instance Metamodel.

In the terminology of Meta-Object Facility (MOF) [31], that means the particular message schema information is captured as an M1 instance of the M2 Message Schema Metamodel, which is shown in Figure 2, while the particular message instance and its content are captured as an M1 instance of the M2 Message Instance Metamodel, which is shown in Figure 3. (Hereafter, when we refer to the M1 instances of the Message Metamodel we will use a "message-schema model", "message-instance model", or "message model" for both).

A corresponding message model is one MOF-level higher than that actual message schema and message instance (e.g. an XML Schema message schema and XML message instance) as the model actually represents message concepts, structure and content model (Figure 4).[1] The example of a message-schema model is given in the Section 5.1, Listing 2; that model corresponds to the XML message schema in the Section 5.1, Listing 1. The example of a message-instance model is given in the Section 5.3, Listing 8; that model corresponds to the XML message instance given in the Section 5.3, Listing 7.

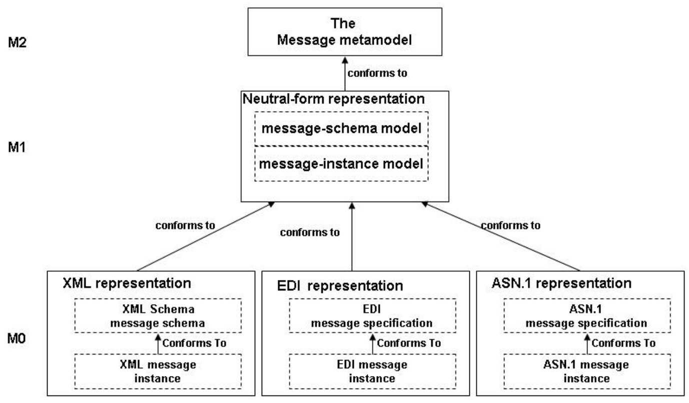


Figure 4 Message Metamodel within MOF layering architecture

It is important to mention that the information captured in a particular message-schema model is not sufficient to generate the original message schema, as some schema information is ignored (e.g., XML simple type minInclusive or maxInclusive restrictions are not captured) and some schema constructs are generalized (e.g., xs:group and xs:complexType are both StructuredContent). However, it would be possible to generate a message schema that contains the same validation requirements as the original schema with respect to conformance of the message structure, message elements names, elements order, and unrestricted datatypes, with the definitions given in the schema.

On the other hand, the message-schema model and the message instance-model together contain sufficient information to reconstruct the message schema-conforming and syntax-specific message instance from the message-instance model corresponding to that message instance. That is why this concept is named the Message Metamodel; its M1 instance represents a model of the message (because of the message-instance model part), and furthermore, captures all the primary message syntactic concepts – naming, structure, occurrences, and value representation - (because of the

[1] In the terms of an ontology, this means that Message Metamodel is a set of TBox statements (i.e., set of terms of controlled vocabulary) while message-schema model and message-instance model are a set of ABox statements (i.e., set of assertion associated with the controlled terminological vocabulary).

message-schema model part), which are needed to construct a message schema-conforming message instance in a target representation language (e.g. XML, EDI). That is, the intent of the Message Metamodel is to be a solid model of the message for the *semantic reconciliation* of *that message*.

The Message Metamodel as an intermediate model for representing messages and their schemas (for well-established syntactic varieties) insulates the content transformation activities from a specific message syntax, and allows the semantics reconciliation software to be reused with different physical representation syntaxes. As was the case with local conceptual model, the Message Metamodel also necessitates the construction of the software libraries to transform message schema and message content to and from the Message Metamodel form. For an example, the '*XML Schema to Message Schema Model transformer*' that would transform an actual XML Schema message schema to a corresponding message-schema model, and the '*XML to Message Instance Model transformation tool*' that would transform an actual XML message to a corresponding message-instance model. Yet, the Message Metamodel may allow other types of software, such as ones that interpret or annotate the message semantics, to be reused with different physical message representation syntaxes.

Finally, a particular message model (M1 instance), can be physically represented and exchanged in numerous ways, for an example as XML Metadata Interchange (XMI) file, set of OWL individuals, set of RDF statements, or MOF database population. In this work we use OWL representation for message models as that *ontologization* of the message model gives a form suitable for defining reconciliation between the particular message model on one side, and an OWL reference ontology on other side.

4. Application of the Message Metamodel in the semantic reconciliation architecture

A semantic-reconciliation architecture enhanced with the Message Metamodel is shown in Figure 5. The semantic-reconciliation architecture employs a reference ontology, the semantic reconciliation tools (reconciliation rules definition tool, which are not shown, and reconciliation rule engine) and message-representation transformation tools.

The reference ontology captures the shared conceptualization and business meaning of messages involved in a particular B2B scenario. The reconciliation rule engine provides the functionality to execute the forward reconciliation rules when applications are sending a message instance and backward reconciliation rules when applications are receiving a message instance.

The forward rules specify message content transformation from of a message instance to a reference ontology instance, while backward rules specify transformation from the reference ontology instance to a message instance.

The message-representation transformer tools provide transformations of the message schemas and message instances to corresponding models in a form of the Message Metamodel. The reconciliation rule engine and runtime message-representation transformer tools may be assembled into a single software component (e.g., *semantic mediator*). The architecture supports message content transformation between independently developed applications (applications A and B in Figure 5) that differ either due to differently specified message schemas (a proprietary or standard-based), or different message-representation standards, or both.

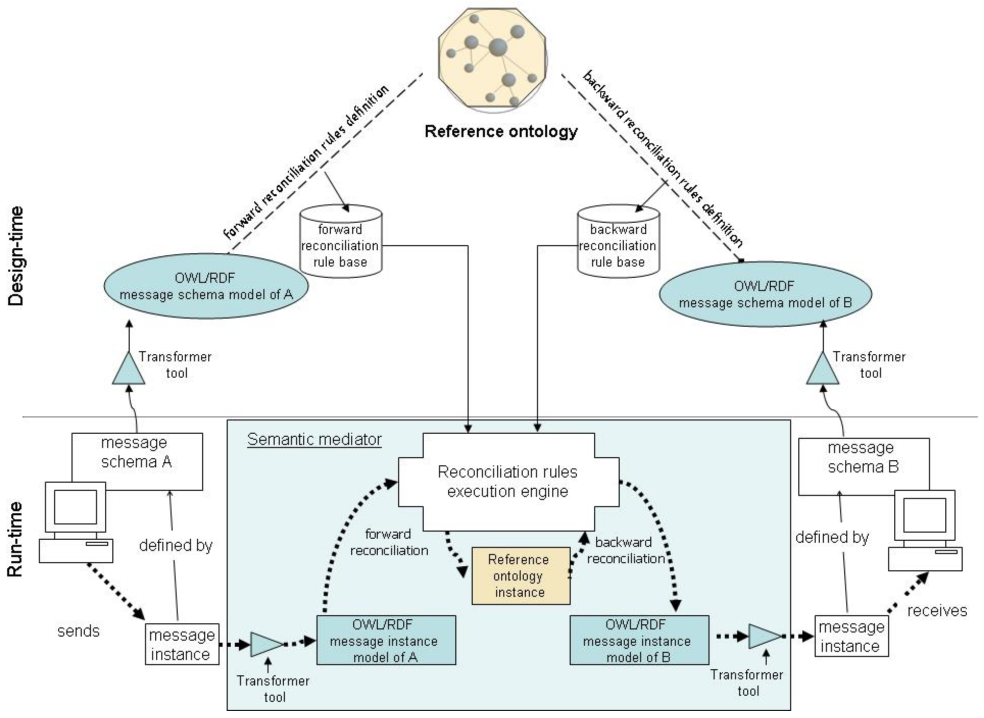

Reference ontology
forward reconciliation rules definition
backward reconciliation rules definition
forward reconciliation rule base
backward reconciliation rule base
Design-time
OWL/RDF message schema model of A
OWL/RDF message schema model of B
Transformer tool
Transformer tool
Run-time
message schema A
defined by
sends
message instance
Transformer tool
Semantic mediator
Reconciliation rules execution engine
forward reconciliation
Reference ontology instance
backward reconciliation
OWL/RDF message instance model of A
OWL/RDF message instance model of B
Transformer tool
message schema B
defined by
receives
message instance

Figure 5 A Message Metamodel enhanced semantic-reconciliation: A conceptual architecture

4.1. *The semantic-reconciliation methodology*

Within the architecture, the semantic-reconciliation activities take place at design-time and run-time. We assume here that the reference business ontology is already developed (and formally represented using an OWL or RDFS) by the business community and publicly available to the B2B participants.

- **At design-time**, message schemas of all participating applications are transformed to the corresponding message-schema models; a specific transformer tool is applied, depending on the message-representation standard used for a particular message schema (e.g., *XML Schema message schema to message-schema model* or *EDI message specification to message-schema model* transformation). The transformer tools produce OWL/RDF representations of the message-schema models, which is the representation form *syntactically normalized* with the language used to represent the reference ontology. Syntactical normalization of the representation of the message-schema models with the representation of the reference ontology provides an opportunity to use emerging Semantic Web technologies and rule-based reasoning over OWL/RDF documents. The rule-based reasoning over the OWL/RDF documents is used to define forward rules that access the message instance content from the OWL/RDF message-instance model and create instances of the OWL reference ontology concepts and populate the instances with the message instance content, as well as for the backward content transformation. For an example, we use the Jena rule language [16] to create executable rules that transform message instance content between the OWL reference ontology form and corresponding OWL/RDF message-instance model, as described in later sections. After the message schemas are transformed in the OWL/RDF message-schema models, a reconciliation rule engineer uses the OWL/RDF message-schema models and the reference ontology to define rules for the forward and backward reconciliation between OWL/RDF message-instance models and the

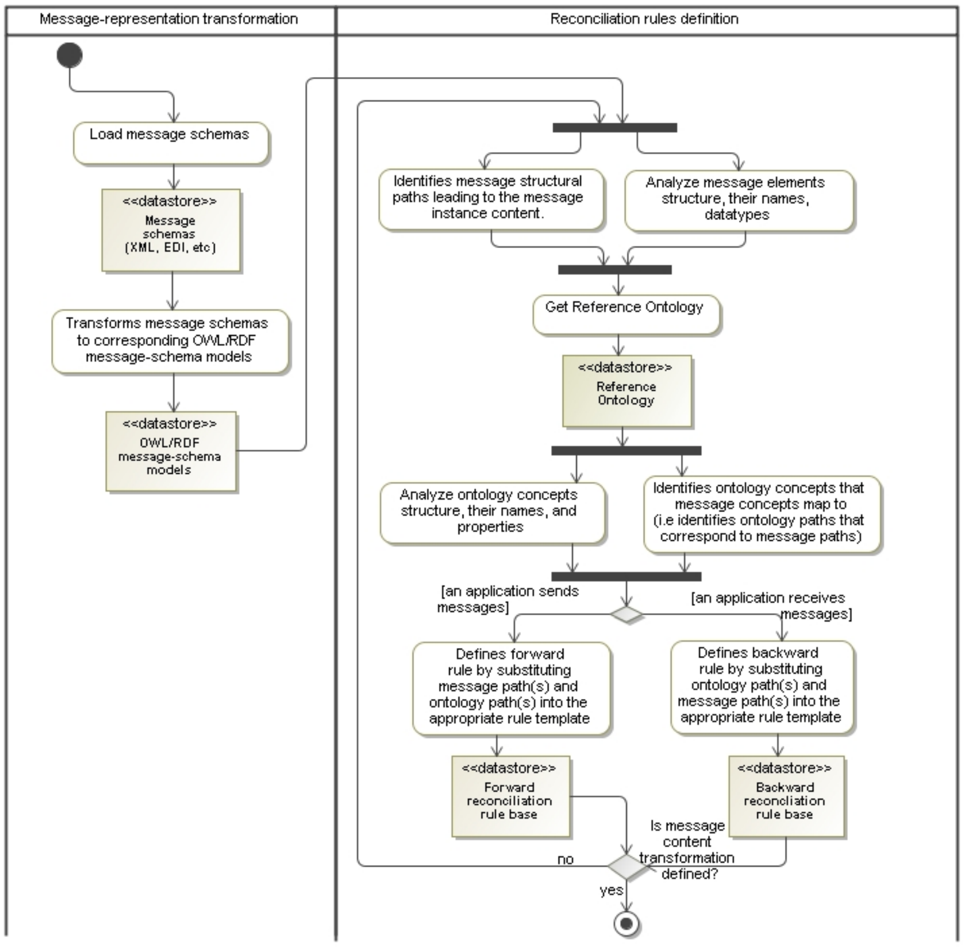

Message-representation transformation
Reconciliation rules definition
Load message schemas
<<datastore>> Message schemas (XML, EDI, etc)
Transforms message schemas to corresponding OWL/RDF message-schema models
<<datastore>> OWL/RDF message-schema models
Identifies message structural paths leading to the message instance content.
Analyze message elements structure, their names, datatypes
Get Reference Ontology
<<datastore>> Reference Ontology
Analyze ontology concepts structure, their names, and properties
Identifies ontology concepts that message concepts map to (i.e identifies ontology paths that correspond to message paths)
[an application sends messages]
[an application receives messages]
Defines forward rule by substituting message path(s) and ontology path(s) into the appropriate rule template
Defines backward rule by substituting ontology path(s) and message path(s) into the appropriate rule template
<<datastore>> Forward reconciliation rule base
<<datastore>> Backward reconciliation rule base
Is message content transformation defined?
no
yes

Figure 6 Design-time semantic-reconciliation steps from the perspective of reconciliation rule engineer.

- **At run-time,** when an application sends a source message instance (e.g., XML or EDI), the transformer tool (within a *semantic-mediator)* transforms the source message instance to the corresponding source OWL/RDF message-instance model. Then, the reconciliation rule engine takes the source OWL/RDF message-instance model and executes the forward ruleset for a sending application (Application 'A' in Figure 7). This step generates a reference ontology population of instances matching the *content* of the source message instance. Next, the reconciliation rule engine takes that reference ontology population, and executes the backward ruleset defined for a receiving application (application 'B' in Figure 7). This step generates a target OWL/RDF message-instance model for the receiving application. Then, the syntax-specific message-representation transformer tool transforms the target OWL/RDF message-instance model to a message-schema conforming message instance in the syntax expected by the receiving application.

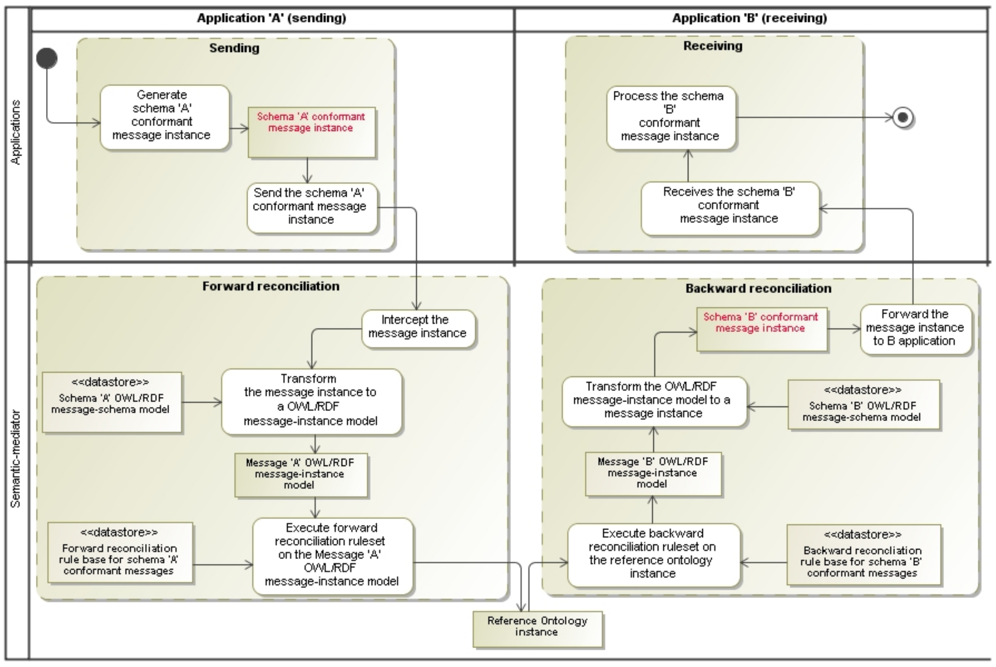

Application 'A' (sending)
Application 'B' (receiving)
Applications
Semantic-mediator
Sending
Generate schema 'A' conformant message instance
Schema 'A' conformant message instance
Send the schema 'A' conformant message instance
Receiving
Process the schema 'B' conformant message instance
Receives the schema 'B' conformant message instance
Forward reconciliation
Intercept the message instance
<<datastore>> Schema 'A' OWL/RDF message-schema model
Transform the message instance to a OWL/RDF message-instance model
Message 'A' OWL/RDF message-instance model
<<datastore>> Forward reconciliation rule base for schema 'A' conformant messages
Execute forward reconciliation ruleset on the Message 'A' OWL/RDF message-instance model
Backward reconciliation
Schema 'B' conformant message instance
Forward the message instance to B application
Transform the OWL/RDF message-instance model to a message instance
<<datastore>> Schema 'B' OWL/RDF message-schema model
Message 'B' OWL/RDF message-instance model
Execute backward reconciliation ruleset on the reference ontology instance
<<datastore>> Backward reconciliation rule base for schema 'B' conformant messages
Reference Ontology instance

Figure 7 Run-time semantic reconciliation steps from the perspective of sending and receiving application.

5. Experimental scenario and implementation

To assess representational capabilities of the proposed architecture and the application of the Message Metamodel within that architecture, we executed a scenario that involved two business applications whose interfaces are based on different message-schema standards but the same message-representation standard. The scenario is taken from the manufacturing industry, and deals with the supply-chain situation in which the electronic Kanban (eKanban) business process regulates the flow of goods from the supplier to match actual usage by the customer. In this scenario, “inventory visibility” applications support manufacturers and their suppliers, and communicate with each other by XML message instances to reach other suppliers.

A similar scenario with a semantic-reconciliation architecture, which, however, uses local conceptual models to represent model of messages, was executed in the Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications (ATHENA) B5.10 validation project [14, 11]. In [11], it was reported that additional effort for message reconciliation was needed because of the inadequacy of a local conceptual model as the model of message. So we chose this scenario to compare the performance of the model of messages we introduce here.

Our experimental scenario involved two business applications with one-way, single message communication between them. The proprietary version of the *AuthorizeKanban* business message used by the General Motors manufacturer’s application (“GM” in Figure 8), had to be transformed to the standard *AuthorizeKanban* Business Object Document (BOD) message used by the inventory visibility application (“IV”). Both *AuthorizeKanban* message schemas are specified in XML Schema, and there were several mismatches between the schemas, e.g., naming mismatches where message elements have the same content but different names (`gmSyncShipmentSchedule` vs.

SyncShipmentSchedule) or structural path mismatches where different sequences of message elements and sub-elements (i.e., paths) exist from the root element to the element where content is captured (*gmSyncShipmentSchedule.documentId* vs. *SyncShipmentSchedule.DataArea.ShipmentSchedule.ShipmentScheduleHeader.DocumentId*).

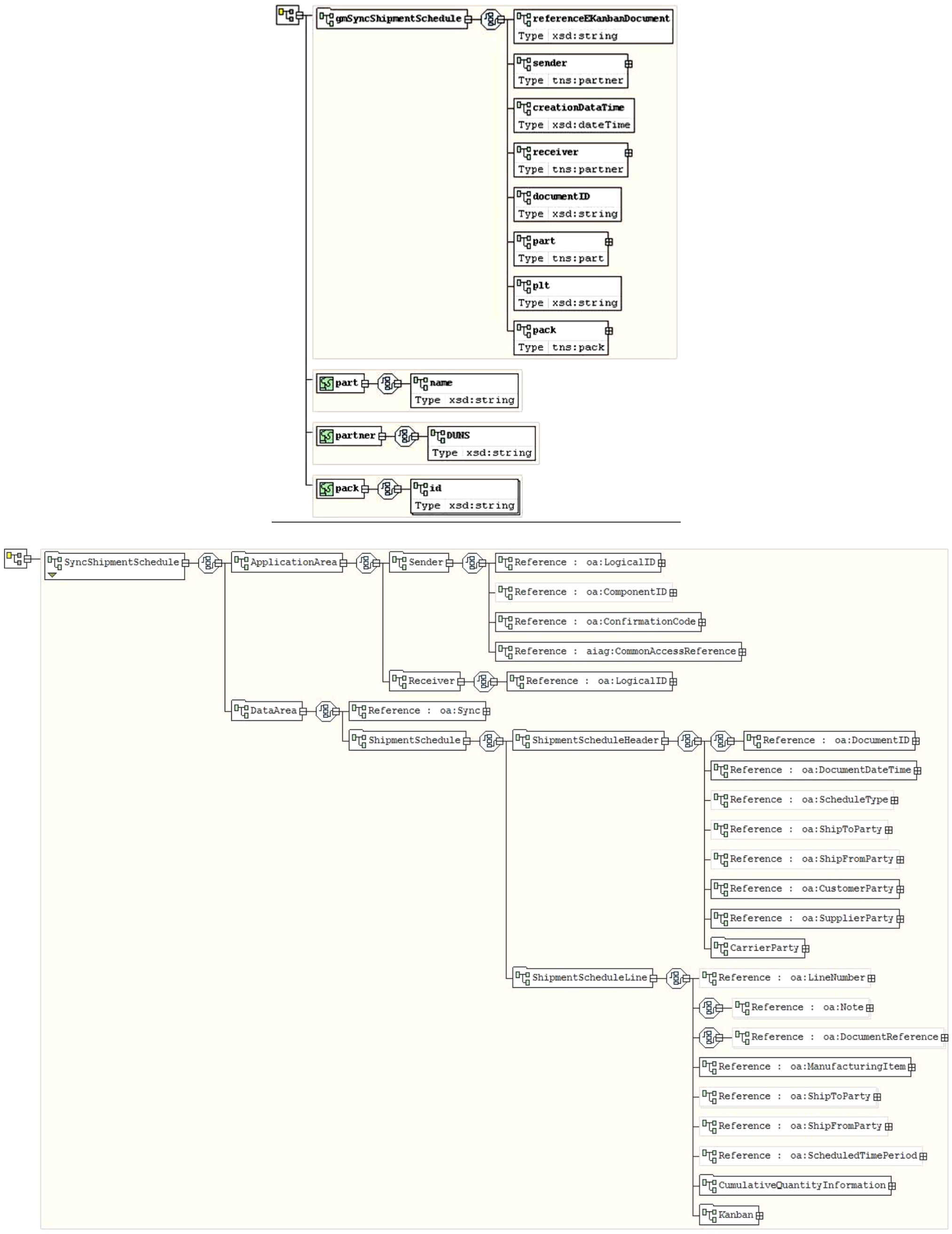


Figure 8 A fragments of the GM's and BOD *AuthorizeKanban* XML schemas, respectively (in a tree-like view)

The experiment involved following steps:

(a) transformation of the GM *AuthorizeKanban* and the BOD *AuthorizeKanban* XML message schemas to the GM and BOD *AuthorizeKanban* message-schema models, respectively,
(b) definition of the forward reconciliation rules for the GM *AuthorizeKanban* message-instance model,
(c) definition of the backward reconciliation rules for the BOD *AuthorizeKanban* message-instance model, and
(d) execution of the message-exchange scenario.

In the experiment we used the eKanban Reference Ontology [17] that was also used in the ATHENA B5.10 scenario. Figure 9 shows the fragment of the eKanban Reference Ontology. In the following sections we detail steps (a-d).

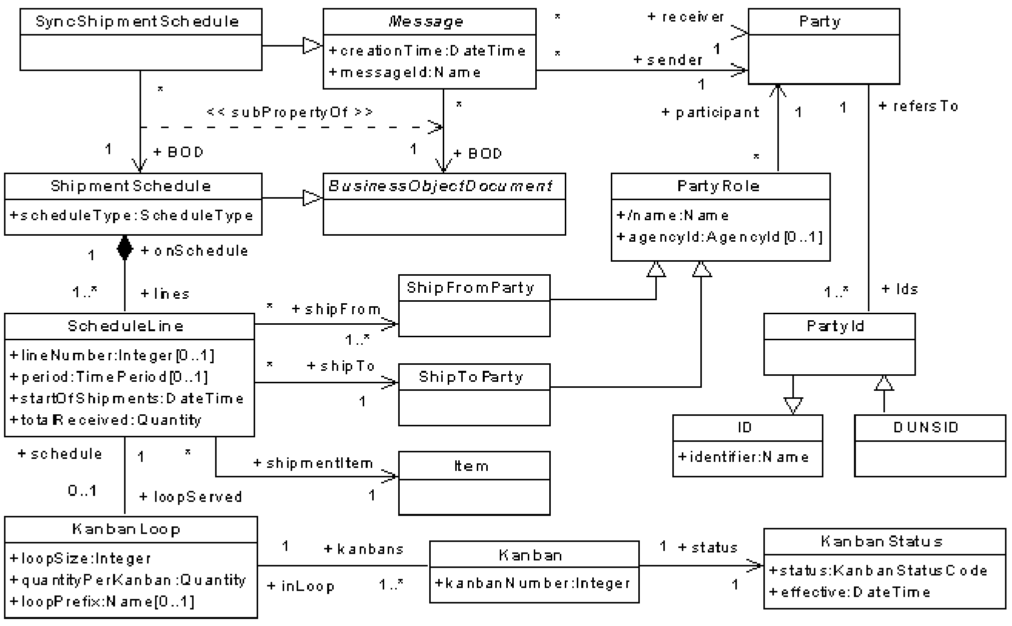


Figure 9 A fragment of the eKanban Reference Ontology

5.1. *Transformation of XML message schemas to OWL/RDF message schemas models*

The transformation of the XML message schemas to the corresponding OWL/RDF message-schema models is accomplished by the *XML Schema to Message Schema Model transformer* tool. The Message Metamodel represented in Figures 1 and 2 is actually implemented as a set of Java classes that represent Message Metamodel concepts and as an OWL Message Model Ontology[2]. Transformation rules are defined at the MOF M2 level between XML Schema Definition concepts

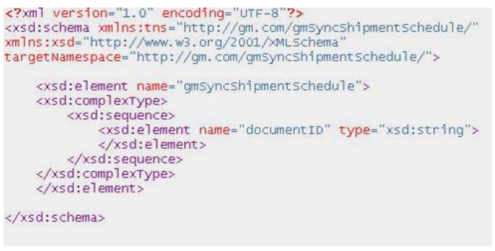
[2] The Message Metamodel Java API and Message Model OWL Ontology as well as the design-time and run-time 'message to model' transformer tools are available at http://meta-messager.sourceforge.net. Also provided are the XML schemas, XML message files, and reconciliation rules to recreate example transformations and message reconciliations shown in this paper.

and Message Schema Metamodel concepts, which provide transformation of any given XML message schema to the corresponding message-schema model.

The transformation from a message schema to an OWL/RDF message-schema model involves two phases. First, a given message schema is transformed to a message-schema model captured as in-memory Java objects. Then, if an OWL/RDF form of a message-schema model is needed, the transformer tool creates an OWL/RDF representation of the message-schema model, by creating an equivalent OWL individual (an instance of the concept from the Message Model Ontology) for each message-schema model object.

For the *gmSyncShipmentSchedule* and *documentID* elements of the GM XML message schema, which fragment is shown in Listing 1, the corresponding OWL/RDF message-schema model, in the RDF Turtle syntax [15], is shown in Listing 2. Note that only a part of the OWL/RDF message-schema model is shown here.

Listing 1 A fragment of the GM XML schema



```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:tns="http://gm.com/gmSyncShipmentSchedule/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://gm.com/gmSyncShipmentSchedule/">

    <xsd:element name="gmSyncShipmentSchedule">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="documentID" type="xsd:string">
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    </xsd:element>

</xsd:schema>
```

Listing 2 A fragment of the GM message-schema model that corresponds to the GM XML schema in Listing 1.

```
@prefix p1: <http:///MessageMetamodel.ecore#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#>.

:gmSchema
       rdf:type p1:Schema ;
       p1:defines :elementModel1 ;
       p1:schemaNamespace :gmSchemaNamespace.

:gmSchemaNamespace
      rdf:type p1:SchemaNamespace ;
      p1:URI  http://gm.com/gmSyncShipmentSchedule/
      ^^xsd:string ;
      p1:names :name1 .

:nsBinding1
      rdf:type p1:NSBinding ;
      p1:prefix "tns"^^xsd:string ;
      p1:refersTo :gmSchemaNamespace .

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

:name1
      rdf:type p1:Name ;
      p1:definedIn :namespace1 ;
      p1:nameText
"gmSyncShipmentSchedule"^^xsd:string ;
      p1:refersTo :elementModel1 .

:structuredContent1
      rdf:type p1:StructuredContent ;
      p1:contains :cu1 ;
      p1:defines :elementModel2 ;
      p1:inModel :elementModel1 ;
      p1:localNamespace :localNamespace1 .

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

```
:elementModel2
      rdf:type p1:ElementModel ;
      p1:contentModel :simpleContent1 ;
      p1:inModel :structuredContent1 ;
      p1:modelName :name2 .

:name2
      rdf:type p1:Name ;
      p1:nameText "documentID"^^xsd:string ;
      p1:refersTo :elementModel1 .
      p1:definedIn :localNamespace1.
```

GM XML schema is captured by `':gmSchema'` which is an instance of `Schema` concept. `gmSyncShipmentSchedule` XML element definition is transformed in `':elementModel1'` which `modelName` is `':name1'`. `':name1'` is an instance of `Name` concept, and its `'nameText'` has literal value *`gmSyncShipmentSchedule`*. `':name1'` is `definedIn` `':gmSchemaNamespace'`, which is a `schemaNamespace` of `':gmSchema'` and represents *`http://gm.com/gmSyncShipmentSchedule`* namespace. `':elementModel1'` defines and contains `':structuredContent1'`. *`documentId`* XML element is captured by `':elementModel2'` `ElementModel` which is defined in `':structuredContent1'`. `':structuredContent1'` represents complex type declared locally in the definition of XML *`gmSyncShipmentSchedule`* element.

The *XML Schema to Message Schema Model transformer* tool successfully transformed the GM *AuthorizeKanban* and BOD *AuthorizeKanban* XML Schemas to the GM *AuthorizeKanban* and BOD *AuthorizeKanban* message-schema models, respectively. Ultimately, two OWL/RDF documents were created: an OWL representation of the GM message-schema model, and an OWL representation of the BOD message-schema model.

5.2. *Reconciliation rules definition using OWL/RDF message-schema models and a reference ontology*

The definition of reconciliation between two differently specified messages involves defining the forward and backward message content transformation rules for each message-schema model.

Forward rules formally describe how to construct instances of concepts in the reference ontology by operational transformation of the message content of one or more message elements appearing in the source message instance. The rules are defined based on elements in the message-schema model, but they operate on corresponding elements of the message-instance model. *Backward rules* formally describe how to construct message elements and message content of a target message instance by an operational transformation on the instances of the reference ontology. There can be several message content transformation patterns, such as one-to-one, many-to-one, one-to-many, or more complex patterns including conversion functions.

In the proposed architecture, semantic reconciliation is a transformation of data in one OWL/RDF document to another OWL/RDF document; it is a transformation of a message-instance model in OWL/RDF form to reference ontology instances in OWL/RDF form, or vice-versa.

The transformation between two OWL/RDF documents is a transformation schema (TS_x) that consists of a set of forward chaining rules (*r*), where the rules are defined over a set (*S*) of OWL/RDF triplets ($\mathcal{T}$ *(subject, predicate, object)*) of a particular knowledge base L. Knowledge base L corresponds to an OWL/RDF document. Each rule maps a set of $\mathcal{T}_i$ triples of one knowledge base L_{OLD} into a set of $\mathcal{T}_n$ triples of other knowledge base L_{NEW}, whilst retaining the rest of L ($\mathcal{V}_k$).

$TS_X : L_{OLD} \rightarrow L_{NEW}$ by: for each r in TS_X,
r: $L_{OLD} = \{\mathcal{T}_i\ (subject, predicate, object)\} \cup \{\mathcal{V}_k\ (subject, predicate, object)\}$
$\rightarrow L_{NEW} = \{\mathcal{T}_n(subject, predicate, object)\} \cup \{\mathcal{V}_k\ (subject, predicate, object)\}$
TS_F x L_{MM} -> L_{RO} {transformation schema for forward reconciliation from source message model (MM, for short) to reference ontology (RO, for short)}
TS_B x L_{RO} -> L_{MM} {transformation schema for backward reconciliation from reference ontology to target message model}

When a set of triplets $\mathcal{T}_i$ in the body of a rule holds, which means that set of triplets $\mathcal{T}_i$ belongs to the knowledge base L_{OLD}, then new knowledge L_{NEW} is derived as a set of triplets $\mathcal{T}_n$ defined in the head of the rule. This applies for any transformation pattern; however, in non-trivial cases (e.g., literal value splitting or merging), it is necessary to include built-in and custom functions in the body of a rule. Ultimately, executing the TS_F and TS_B transformation schemas accomplish reconciliation between two message models. The transformation may involve vocabulary substitution, property deletion or creation, structural abstraction (replacing a sub-graph by a node), structure introduction, structural rewriting, and literal value transformation.

A reconciliation rules engineer uses the OWL/RDF message-schema models to define the forward and backward reconciliation rules that operate on OWL/RDF message-instance models. As the reconciliation rules actually transform the message instance content captured in the OWL/RDF message-instance models (forward rules) or populate the message instance content of the OWL/RDF message-instance models (backward rules) it is necessary to produce a set of triples $\mathcal{T}$ *(subject, predicate, object)* that formulates the structural path leading to the message instance content in the OWL/RDF message-instance model. The set is named *Spath*. For the forward reconciliation, in the case of one-to-one or one-to-many content transformation, only one *Spath* exists in a rule body, while in the case of many-to-one or many-to-many content transformation, several different *Spath* sets may exist in a rule body to access the content. For the backward reconciliation, in the case of many-to-one or one-to-one content transformation, only one *Spath* set of triples exists in a rule head, while in the case of one-to-many or many-to-many content transformations several different *Spath* sets may exist in a rule head to populate the content depending. For an example, Listing 3, in the rule body, shows the *Spath* for the `gmSyncShipmentSchedule.documentId` XML message path that leads to the value of a `documentId` element.

To generate *Spath* sets needed to specify rules, a reconciliation rules engineer uses a tool that generates an *Spath* set for a given message element path. The tool generates the *Spath* from corresponding OWL/RDF message-schema models as the message-schema models capture original message structure, and thus, information about the structural paths leading to a message content. Starting from the root `elementModel` that represents the `Message`, the *Spath* is created by following `Component.contentModel.StructuredContent.contains.ContentUse.content` relationship for `Components` which `content` is `StructuredContent`, and by `Component.contentModel.SimpleContent` relationship for `Components` which content is `SimpleContent`.

In practice, this means:

- if `elementModel` has a `structuredContent` then a `[(?xi rdf:type m:StructuredElement)(?xi m:model modelUri)]` is generated; further, that structured element have items as defined by `Component.contentModel.StructuredContent.contains.ContentUse.content` relationship, which generates `(?xi m:items ?xi+1)`.

- If the `ContentUse.content` is another `elementModel` that has a `structuredContent` then a `[(?xi+1 rdf:type m:StructuredElement)(?xi+1 m:model modelUri)(?xi+1 m:inElement ?x)]` is generated and the triples-generation-algorithm is executed again for that `elementModel`.
- if the `ContentUse.content` is an `elementModel` that has a `simpleContent` then a `[(?xi+1 rdf:type m:SimpleElement)(?xi+1 m:model modelUri)(?xi+1 m:inElement ?x)(?xi+1 m:value ?v)(?v rdf:type m:SimpleValue)(?v m:text ?txt)]` is generated.
- if the `ContentUse.content` is an `attributeModel` then `[(?xi+1 rdf:type m:Attribute)(?xi+1 m:model modelUri) (?xi+1 m:ofElement ?x) (?xi+1 m:value ?v)(?v rdf:type m:SimpleValue)(?v m:text ?txt)]` is generated.

The reconciliation rule engineer generates needed *Spaths*, chooses corresponding reference ontology concepts that message concepts map to, and then defines executable reconciliation rules. To define executable reconciliation rules, Jena rule language is used as Jena provides inference over OWL/RDF triples by executing the forward chaining rules.

The rule in Listing 3 defines the transformation of the GM *`gmSyncShipmentSchedule.documentId`* concept to the *`SyncShipmentSchedule.ids.DocumentId.identifier`* concept defined in the eKanban reference ontology. There was a path-naming mismatch between these two concepts and simply one-to-one mapping was needed.

Listing 3 Jena forward reconciliation rule for the GM *`gmSyncShipmentSchedule.documentId`* element; one-to-one map to the *`SyncShipmentSchedule.BOD.ShipmentSchedule.ids.DocumentId.identifier`* ontology concept

```
@prefix m: <http://MessageMetaModel.core#>
@prefix ro: <http://referenceOntology.eKanban#>
[
 (?e1 rdf:type m:StructuredElement) (?e1 m:model d: elementModel1)
 (?e1 m:items ?e2) (?e2 m:inElement ?e1) (?e2 rdf:type m:SimpleElement)
 (?e2 m:model d: elementModel2) (?e2 m:svalue ?v) (?v rdf:type m:SimpleValue)
 (?v m:text ?txt)
 ->
 (ro:e1 rdf:type ro:SyncShipmentSchedule) (ro:e1 ro:BOD ro:e2)
 (ro:e2 rdf:type ro:ShipmentSchedule)(ro:e2 ro:ids ro:e3)
 (ro:e3 rdf:type ro:DocumentId)(ro:e3 ro:identifier ?txt)
]
```

The body of the rule in Listing 3 refers to *`gmSyncShipmentSchedule.documentId`* content by referring to the literal value `(?txt)` of all the `StructuredElements` (?e3) whose `model` is `‘d:_elementModel1’`, and whose `items` are `SimpleElements (?e2)` whose `model` is `‘d:_elementModel2’` and that have a `SimpleValue (?v)` with the `text` value `?txt`. The `model` is provided from corresponding message-schema model, and it is a link between an element of the message-instance model and its definition captured in the message-schema model (`Element.model.ElementModel` relationship shown in Figure 2). The link is established using the resource’s description identifier `rdf:ID`, which is a unique identifier of the an actual `elementModel` defined in the OWL/RDF message-schema model. For example, on the right side of Figure 7, the `‘d:_elementModel1’` is the `rdf:ID` of the `ElementModel` that represents *`gmSyncShipmentSchedule`* XML element, and `‘d:_elementModel2’` `rdf:ID` of the `ElementModel` that represents *`documentId`* XML element.

The rule in Listing 4 defines a transformation for the Kanban Status concept, which was missing in the GM message, by setting the corresponding element of the eKanban ontology to the 'Authorized' literal value. This actually was a coverage mismatch, which occurs either when a concept in a message has no match in the reference ontology or when reference ontology has a concept not used in a message. In the first case, the corresponding information will be lost in outbound message instances and missing from inbound message instances. When that information is optional and seldom used, it may not be a problem; but, when it is important to the application, it means that the reference ontology is inadequate. The reverse case, in which the reference ontology has a concept not used in the application message, is usually harmless – the reference ontology may well support many different messages, only some of which use any given concept. But if it is a mandatory property of a required object, the application may not be suitable for the use envisaged in the standard.

Listing 4 Jena forward reconciliation rule for the GM message where Kanban Status concept is missing; sets default 'Authorized' value to the *SyncShipmentSchedule_Message.ShipmentSchedule.lines.ScheduleLine.loopServed. KanbanLoop.hasKanban.Kanban.status.KanbanStatus.KanbanStatusCode* ontology concept.

```
@prefix m: <http://MessageMetaModel.core#>
@prefix ro: <http://referenceOntology.eKanban#>
[
->
(ro:e1 rdf:type ro:SyncShipmentSchedule)
(ro:e1 ro:BOD ro:e2)
(ro:e2 rdf:type ro:ShipmentSchedule)
(ro:e2 ro:lines ro:e3)
(ro:e3 rdf:type ro:ScheduleLine)
(ro:e3 ro:loopServed ro:e4)
(ro:e4 rdf:type ro:KanbanLoop)
(ro:e4 ro:hasKanban ro:e5)
(ro:e5 rdf:type ro:Kanban)
(ro:e5 ro:status ro:e6)
(ro:e6 rdf:type ro:KanbanStatus)
(ro:e6 ro:KanbanStatusCode 'Authorized')
]
```

The rule in Listing 5 defines the one-to-many transformation of the GM *gmSyncShipmentSchedule.part.name* concept to the *SyncShipmentSchedule.ShipmentSchedule.lines.ScheduleLine.shippedItem.Item.description.ItemDescription.Description.Text* and *SyncShipmentScheduleMessage.ShipmentSchedule.lines.ScheduleLine.shippedItem.Item.ids.ItemID.PartID* concept. There was a path-name and an attribute-granularity mismatch between these two concepts and the literal value of the GM's concept needed to be split into two literal values. Attribute-granularity mismatches exist when the granularity of data is different. The split operation is supported by the Jena Split built-in function.

Listing 5 Jena forward reconciliation rule for GM *gmSyncShipmentSchedule.part.name* concept; one-to-many transformation.

```
@prefix m: <http://MessageMetaModel.core#>
@prefix ro: <http://referenceOntology.eKanban#>
[
(?e1 rdf:type m:StructuredElement)(?e1 m:model d:elementModel1)
(?e1 m:items ?e2)(?e2 m:inElement ?e1)
(?e2 rdf:type m:StructuredElement) (?e2 m:model delementModel5)
(?e2 m:items ?e3)(?e3 m:inElement ?e2)
(?e3 rdf:type m:SimpleElement)(?e3 m:model d:elementModel8)
(?e3 m:svalue ?v)(?v rdf:type m:SimpleValue)(?v m:text ?txt)
->
 Split(?txt,?y1, ?y2, '-')
(ro:e1 rdf:type ro:SyncShipmentSchedule) (ro:e1 ro:BOD ro:e2)
(ro:e2 rdf:type ro:ShipmentSchedule) (ro:e2 ro:lines ro:e3)
(ro:e3 rdf:type ro:ScheduleLine)(ro:e3 ro:shippedItem ro:l1)
(ro:l1 rdf:type ro:Item) (ro:l1 ro:description ro:l2)
```

```
 (ro:12 rdf:type ro:Description)(ro:12 ro:Text ?y1)

 (ro:e1 rdf:type ro:SyncShipmentSchedule) (ro:e1 ro:BOD ro:e2)
 (ro:e2 rdf:type ro:ShipmentSchedule) (ro:e2 ro:lines ro:e3)
 (ro:e3 rdf:type ro:ScheduleLine) (ro:e3 ro:Item ro:l1)
 (ro:l1 rdf:type ro:Item) (ro:l1 ro:ids ro:l2)
 (ro:l2 rdf:type ro:ItemId) (ro:l2 ro:partId ?y2)
 ]
```

The rules shown in Listings 3, 4, and 5 are examples of forward reconciliation rules for the GM message model. Listing 6 shows an example of a backward rule, which is defined for an element of the BOD message model.

Listing 6 Jena backward reconciliation rule for BOD's *`SyncShipmentSchedule.DataArea.ShipmentSchedule.ShipmentScheduleHeader.DocumentID.ID`* concept; one-to-one map from the *`SyncShipmentSchedule.BOD.ShipmentSchedule.ids.DocumentId.identifier`* ontology concept. The rule head corresponds to the *Spath* for the BOD's *`ID`* element.

```
@prefix m: <http://MessageMetaModel.core#>
@prefix ro: <http://referenceOntology.eKanban#>
[
(?e1 rdf:type ro:SyncShipmentSchedule)
(?e1 ro:BOD ?e2) (?e2 rdf:type ro: ShipmentSchedule)
(?e2 ro:ids ?e3)(?e3 rdf:type ro:DocumentId)
(?e3 ro:identifier ?txt)
->
(m:e1 rdf:type m:StructuredElement) (m:e1 m:model d:elementModel10)
(m:e1 m:items m:e2) (m:e2 rdf:type m:StructuredElement)
(m:e2 m:model d:elementModel20)(m:e2 m:inElement m:e1)
(m:e2 m:items m:e3)(m:e3 rdf:type m:StructuredElement)
(m:e3 m:model d:elementModel30) (m:e3 m:inElement m:e2)
(m:e3 m:items m:e4) (m:e4 rdf:type m:StructuredElement)
(m:e4 m:model d:elementModel40)  (m:e4 m:inElement m:e3)
(m:e4 m:items m:e10)(m:e10 rdf:type m:StructuredElement)
(m:e10 m:model d:elementModel50)  (m:e10 m:inElement m:e4)
(m:e10 m:items m:e20)(m:e20 rdf:type m:SimpleElement)
(m:e20 m:model d:elementModel60)  (m:e20 m:inElement m:e10)
(m:e20 m:svalue m:wsv1) (m:wsv1 rdf:type m:SimpleValue)
(m:wsv1 m:text ?txt)
]
```

5.3. *Semantic-reconciliation execution*

At run-time, several message-representation transformations and a message content transformation occur when applications exchange message instances. In the experimental scenario, at first, the GM *AuthorizeKanban* XML message instance is transformed to the OWL/RDF GM message-instance model using the *XML message to Message Instance Model* transformer tool. Then, the Jena rule engine executed the TS_F x L_{GM_MM} -> L_{RO} transformation schema on the OWL/RDF GM message-instance model (GM_MM), which produced an instance of the reference ontology (RO). Afterwards, the rule engine executed the TS_F x L_{RO} -> L_{BOD_MM} transformation schema on the instances of the reference ontology, which produced an OWL/RDF BOD message-instance model (BOD_MM). Finally, the *Message Instance Model to XML transformer* tool generated a BOD *AuthorizeKanban* XML message instance. Figure 10 shows message instance transformation and reconciliation flow.

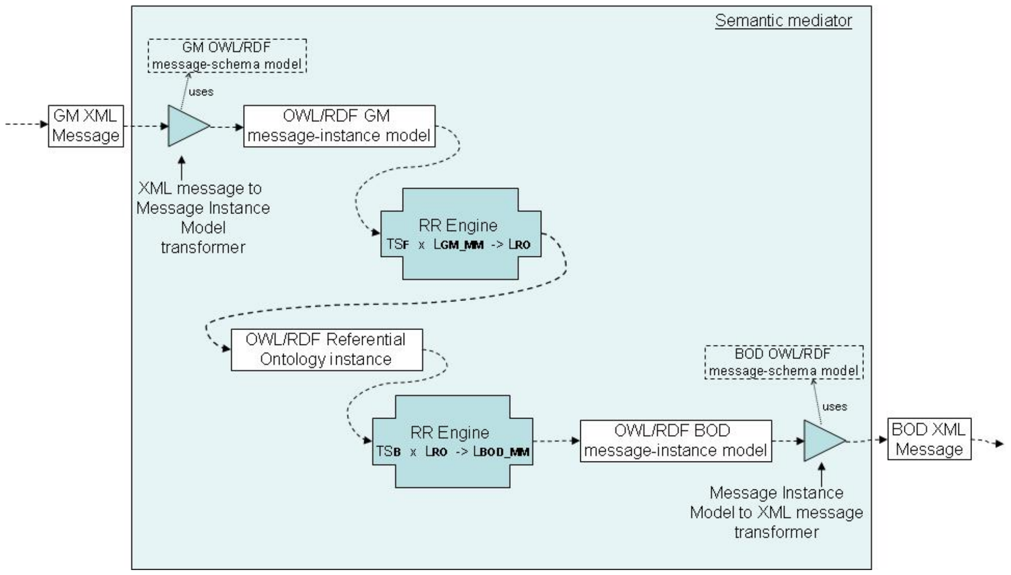
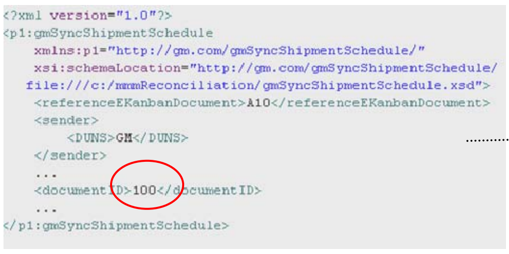

Semantic mediator
GM OWL/RDF message-schema model
uses
GM XML Message
OWL/RDF GM message-instance model
XML message to Message Instance Model transformer
RR Engine
TSF x LGM_MM -> LRO
OWL/RDF Referential Ontology instance
BOD OWL/RDF message-schema model
uses
RR Engine
TSB x LRO -> LBOD_MM
OWL/RDF BOD message-instance model
BOD XML Message
Message Instance Model to XML message transformer

Figure 10 GM to BOD message instance transformation and reconciliation flow

Listings 7, 8, 9, 10, and 11 show message-representation transformations and message-content transformation of the *`gmSyncShipmentSchedule.documentId`* element in the GM message instance to the *`SyncShipmentSchedule.DataArea.ShipmentSchedule.ShipmentScheduleHeader.DocumentID.ID`* element in the BOD instance. The message-instance models and RO instance are presented in RDF Turtle syntax.

Listing 7 The fragment of the GM XML message: semantic reconciliation input



```
<?xml version="1.0"?>
<p1:gmSyncShipmentSchedule
    xmlns:p1="http://gm.com/gmSyncShipmentSchedule/"
    xsi:schemaLocation="http://gm.com/gmSyncShipmentSchedule/
   file:///c:/mmmReconciliation/gmSyncShipmentSchedule.xsd">
    <referenceEKanbanDocument>A10</referenceEKanbanDocument>
    <sender>
        <DUNS>GM</DUNS>
    </sender>
    ...
    <documentID>100</documentID>
    ...
</p1:gmSyncShipmentSchedule>
```

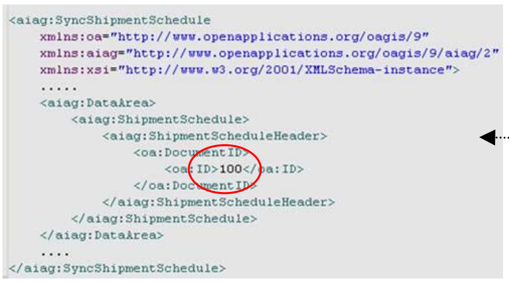
Listing 8 The fragment of the OWL/RDF GM message-instance model

```
@prefix p1:      <http:///MessageMetamodel.ecore#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#>
.
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-
syntax-ns#> .

…
:structuredElement1
      rdf:type p1:StructuredElement ;
      p1:items :simpleElement1 ;
      p1:model :elementModel1 .
:simpleElement1
      rdf:type p1:SimpleElement ;
      p1:actualValue :value1 ;
      p1:inElement :structuredElement1 ;
      p1:model :elementModel2 .
:value1
      rdf:type p1:SimpleValue ;
      p1:text "100"^^xsd:string .
```

Listing 9 The fragment of the RO instance

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-
syntax-ns#> .
@prefix ro:      <http://referenceOntology.eKanban#>
.

….
ro:e1
      rdf:type ro:SyncShipmentSchedule ;
      ro:BOD ro:e2 .
ro:e2
      rdf:type ro:ShipmentSchedule ;
      ro:ids ro:e3 .
ro:e3
      rdf:type ro:DocumentId ;
      ro:identifier "100"^^xsd:string .
….
```

Listing 10 The fragment of the BOD XML message: semantic reconciliation output



```
<aiag:SyncShipmentSchedule
    xmlns:oa="http://www.openapplications.org/oagis/9"
    xmlns:aiag="http://www.openapplications.org/oagis/9/aiag/2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    .....
    <aiag:DataArea>
        <aiag:ShipmentSchedule>
            <aiag:ShipmentScheduleHeader>
                <oa:DocumentID>
                    <oa:ID>100</oa:ID>
                </oa:DocumentID>
            </aiag:ShipmentScheduleHeader>
        </aiag:ShipmentSchedule>
    </aiag:DataArea>
    ....
</aiag:SyncShipmentSchedule>
```

Listing 11 The fragment of the OWL/RDF BOD message-instance model

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-
syntax-ns#>.
@prefix p1:      <http:///MessageMetamodel.ecore#> .
@prefix bmsd:
<http://localhost/BODMessageSchemaMOdel#>
…
p1:e1
      rdf:type p1:StructuredElement ;
      p1:items p1:e2 ;
      p1:model bmsd:elementModel10 .

p1:e2
      rdf:type  p1:StructuredElement ;
      p1:inElement  p1:e1 ;
      p1:items p1:e3 ;
      p1:model bmsd:elementModel20.

p1:e3
      rdf:type p1:StructuredElement ;
      p1:inElement  p1:e2 ;
      p1:items  p1:e4 ;
      p1:model  bmsd:elementModel30  .

p1:e4
      rdf:type p1:StructuredElement ;
      p1:inElement p1:e3 ;
      p1:items p1:w5 ;
      p1:model bmsd:elementModel40.

p1:e10
      rdf:type p1:StructuredElement ;
      p1:inElement p1:e4 ;
      p1:items p1:e20 ;
      p1:model  bmsd:elementModel50 .


p1:e20
      rdf:type p1:SimpleElement ;
      p1:inElement   p1:e10 ;
      p1:model bmsd:elementModel60;
      p1:actualValue   p1:wsv1 .

p1:wsv1
      rdf:type p1:SimpleValue ;
      p1:text    "100" .
```

6. Discussion

Execution of the experimental scenario showed that the proposed architecture successfully supported semantic reconciliation from a source message instance to a schema-conforming target message instance. The Message Metamodel form provided required data-representation information for the message reconciliation.

At design-time, for a given message schema, the corresponding message-schema model provided (1) a definition of each message concept; and (2) a model of a message structure. The definition and model are used for the semantic reconciliation rules definition.

At run-time, for a given message instance, the corresponding message-instance model (1) assigned definition of a concept to each message concept; (2) carried the content of the original message instance; (3) provided access to the message instance content; and (4) through the definition of each message concept, provided information about the syntactic concepts of messages such as naming, structure, occurrences, value representation, and encoding language of the message (i.e., the `language` attribute of `Schema` concept in Figure 2).

The experiment showed that the Message Metamodel form captures necessary information to reconcile a message instance of application A to a message schema-conforming message instance of application B. Although our experiment was based on XML-to-XML message exchange, the

Message Metamodel would work as well for an EDI-to-XML scenario (e.g. EDIFACT Order to UBL Order message).

6.1. *Minimum set vs. maximum set of available message information to the semantic reconciliation and mappings correspondence discovery*

The necessary set of information about a message that the Message Metamodel form explicitly provides to the semantic reconciliation is *the minimum set of explicitly available information* to the semantic-reconciliation. In contrast to that, the *maximum set of explicitly available information* to the semantic reconciliation exists if a local conceptual model of the message is also explicitly available.

The semi-automatic mapping algorithms explore similarities between model of a message and reference ontology by comparing the respective naming and structural characteristics. The minimum set of explicitly available information that Message metamodel form provides may be used in support of semi-automatic mapping correspondences discovery, as the structural organization and naming rules of the message are contained in the minimum set of explicitly available information. The Message Metamodel form does not provide the maximum set of explicitly available information, per se; the Message Metamodel form is not the local conceptual model of message elements. However, when the local conceptual model of a message is also explicitly available, it provides an additional knowledge that may be used in support of semi-automatic mapping correspondences discovery (the additional techniques from ontology matching domain may be applied). Nevertheless, our experiment showed that the presence of the local conceptual model is not mandatory for actual reconciliation – only the minimum set of explicitly available information is necessary and mandatory for successful reconciliation.

Other approaches for semantic reconciliation of messages assume that a local conceptual model of message elements should be the model of a message used for reconciliation rules definition. However, the local conceptual model of message elements is free of all message-format and data-representation rules, which are needed for the semantic reconciliation of a message - for constructing a message schema-conforming message for the intended recipient.

Therefore, this paper argues that the Message Metamodel (or some similar model) should be used as the model of a message for semantic reconciliation. Besides the fact that the Message Metamodel form captures the minimum set of explicitly available information for semantic reconciliation, it also may serve as a model from which a *semantics extraction* software can re-engineer the local conceptual model of a message, if the domain knowledge (such are schema naming and design rules) is accessible and if the local conceptual model is needed by other technologies included in the reconciliation process.

6.2. *Message Metamodel and message semantics annotation*

The Message Metamodel is also envisioned as a model of message for the semantics annotation of business message. The semantics annotation of business messages is needed as existing standards for message schemas define only the syntactic structure of messages without any explicit, formal, and machine-processable representation of the message elements' meaning. The semantics annotation of message elements clarifies a message element semantics by associating explicit and machine-processable semantic annotation expressions to the message element. A semantic annotation expression represents the business meaning of a message element in terms of the adopted reference ontology concepts and their relationships.

The semantics annotation is an indirect approach to the reconciliation definition as the semantics annotation expressions are a knowledge base for automated reconciliation rules generation. Besides that, the semantics annotation expressions shall also be a knowledge base for semantic querying over the business messages, for message schema components discovery and components reusability, regardless of message-schema and message-representation standards. Thus, the semantics annotation activity needs to be transparent with respect to the different message-schema and message-representation standards by providing the unified approach to annotate message elements defined in message schemas. To achieve the transparency and unification of the semantics annotation, the Message Metamodel may be used.

In contrast to a local conceptual model, the Message Metamodel, as a model of message schema and message instance, captures enough information about messages to enable effective mapping between syntactic elements and content concepts, captures original message elements definition as given in message schemas, and captures definition of core and derived components from message schemas. This entire information collection is needed for semantics annotation of message schemas.

7. Related work and assessment

Semantic reconciliation between messages defined by different message schemas may take on several alternative forms, depending on whether the reference ontology is present or not, and on a reconciliation approach applied. Hameed et.al., in [19], and Vetere & Lenzerini in [21] categorize the different architectural models for the semantic reconciliation between local ontologies, which are local conceptual models of message schemas, into: (1) any-to-any model, which does not employ a reference ontology and which reconciles local ontologies pair-wise as needed; (2) any-to-one model, which does employ a single reference ontology that serves as an “interlingua” to which any local ontology may be translated and vice-versa; and (3) hybrid model, which employs multiple reference ontologies in different clusters, providing for reconciliation between the local ontologies and a reference ontology in each cluster, and among the reference ontologies of different clusters.

In these models, the reconciliation may be achieved by the merging or mapping of the ontologies. Merging unifies two or more ontologies with overlapping parts into a joint ontology that includes all information from the sources. Mapping builds executable mapping rules that specify transformation from source ontology to the target ontology.

There are several demonstrations of the any-to-any model in the literature. For an example, Anicic [20] demonstrated an any-to-any model where local OWL ontologies are merged and source OWL individuals classified and transformed into target OWL individuals by description-logic reasoners. Artemis [22] demonstrated any-to-any model based on crosswise mappings among local OWL ontologies. In Artemis, the OWLmt tool (http://sourceforge.net/projects/owlmt) was used for ontology mappings. Oh and Yee [12] described a semantic reconciliation of XML-based messages in a web-services-based manufacturing applications environment by using the Jena rule-based mapping between corresponding local RDFS ontologies.

In the any-to-any models, when many local ontologies are involved, the number and complexity of joint pair-wise ontologies is increased if merging is applied, or the number of crosswise mappings is increased if mapping is applied. In contrast to the any-to-any model, the any-to-one mapping models significantly reduce number of mappings, and there is no need for the ontology merging. The approach proposed in this paper is an instance of the any-to-one model in which the mappings are defined as a set of transformation (reconciliation) rules.

We have found several related demonstrations of the any-to-one model in the literature. The Harmonise project [23] demonstrated semantic-mediation in a tourist B2B network to allow participants to keep the proprietary XML-based message interfaces and still be able to exchange messages. Harmonise defined a reference Harmonisation RDFS Ontology and used the Mafra tool [24] for establishing the mappings between RDFS ontologies. In contrast to the forward-chaining rules used in this work, Mafra provides a proprietary Semantic Bridge Meta-Ontology (SBO) for establishing the mappings by bridging axioms. Mafra transforms source RDF documents to the target RDF documents by evaluating the defined bridge axioms and executing pre-defined functions assigned to each bridge axiom.

Ye and Yang [33] introduced a general Supply Chain Ontology (SCO) that captured concepts and relationships common to the supply chain management, and also used a forward chaining rule-based approach to map between the SCO and local OWL ontologies. The mappings in their work are represented in Semantic Web Rule Language (SWRL) [30].

The ATHENA project [10] introduced toolset for semantic reconciliation of RDF documents: the Astar tool for annotation of the concepts of local RDFS ontologies with their meaning as captured by concepts and its relationship in reference ontology; the Argos tool for RDF-to-RDF document reconciliation definition; and the Ares tool for RDF-to-RDF reconciliation execution. Argos performs either semi-automatically, supported by annotation expressions from the Astar tool, or manually, with the engineer directly instantiating and specifying the rules.

The work in this paper differs from approaches in [10, 12, 20, 22, 23, 33] as we propose the use of Message Metamodel as an ontological model of messages for the reconciliation definition and reconciliation executions, and demonstrate its capability for reconstructing message schema-conforming message instances from the output of the semantic reconciliation. The approaches in [10, 12, 20, 22, 23, 33] assume a local conceptual model in form of local ontology as the model of message, which is insufficient for reconstructing message schema-conforming message instance from the output of the semantic reconciliation.

There are already proposed generic models of information structures in the field of model management [28, 29] that can represent artifacts such as XML schema (or a relational database schema and UML conceptual schema). However, these approaches provide models of schemas only, and do not consider the requirement for providing a neutral-intermediate model of message instances on which reconciliation rules actually operate in a run-time, and whose elements are for that purpose associated with its definition captured in the message-schema models.

Bowers [27] approach relates to ours as he introduced a generic approach for representing and transforming model-based information with the focus on supporting interoperability. Bowers' superimposed-information metamodel is a model of a wide variety of models or schemas including XML Schema, EDI, UML, RDFS, etc. Bowers' superimposed-information metamodel is a single generic representation schema based on RDF that represent the superimposed models, schemas and instances. The metamodel is, however, one MOF layer above the Message metamodel , and therefore, not appropriate as a model of message schema and message instances for the purpose of semantic reconciliation between business messages.

Also, there are several proposed XML metamodels, e.g., XML Schema Definition [47] or Document Object Model [48]. However they are XML-specific. Besides being XML-specific, they may be too expensive in terms of processing as they usually capture all XML syntax-specific details.

Other less expensive XML metamodels are also proposed, such as JavaScript Object Notation (JSON) [49]; however they are insufficient to capture details needed for semantic reconciliation.

Ontological metamodel for EDI-based messages has been reported in [35]. In [35], authors propose ontologization for X12 EDI specs in two phases: the syntax and semantics ontologization. To ontologize syntax, authors in the [35] define and encode a vocabulary for specifying the formats of the EDI concepts: Transaction Sets, Data Segment groups, Data Segments, Composite Data Elements, simple Data Elements, Data Element codes, and Code Sets. Through the use of that vocabulary, the syntax of each of the components is defined. The semantics ontologization involves specifying the semantics of each Transaction Set, Data Segment group, Data Segment, Composite Data Element, simple Data Element, and code in each code set of a chosen X12 subset. The vocabulary defined in [35] is EDI-specific and no effort was made to ontologize the semantics of the X12 terms.

Ontological metamodel for XML-based messages has been reported in [37]. In [37], Yarimagan introduces a Component Ontology for UBL XML message schemas that allows capturing UBL message element definitions and its structural relationships. In Yarimagan's approach, core and custom message components defined in UBL schemas are transformed in corresponding Component Ontologies, and the reconciliation between messages of different customized UBL schemas is accomplished by using description-logic reasoning, similar to [20]. Further, in [36], authors propose the transformation of XML schemas to Web Service Modeling Language (SWML) [39] based local ontologies. However, the [36] proposes the transformation of XML message concepts to the local conceptual model, so it inappropriately captures the needed message schema and message details.

In contrast to [35, 36, 37], the Message Metamodel form syntactic ontologizes relevant concepts of both EDI-based and XML–based message schema or message instance, irrespective of message-specification standard.

The Semantic Annotation for WSDL and XML Schema (SAWSDL) standard [40, 43] also relates to this work. SAWSDL defines a set of XML Schema extension attributes that 'add' semantics to XML schemas. Actually, SAWSDL provides "liftingSchemaMapping" and "loweringSchemaMapping" extension attributes that associate XML schema types or element definitions with a mapping to an ontology. These attributes refer to executable mapping rules that translate XML documents to and from business ontology.

SAWSDL is agnostic to representation language that business ontology is captured in, and it does not prescribe use of any particular mapping representation language. The METEOR-S framework [44] provides for SAWSDL-based reconciliation between XML messages by using Extensible Stylesheet Language Transformation (XSLT) [51] to implement "liftingSchemaMapping" and "loweringSchemaMapping" rules. For the lifting, METEOR-S uses XML Query Language (XQuery) [52] to extract the message content from XML message instance and XSLT to generate a reference ontology population of instances matching the content of the message instance. When lowering, METEOR-S uses RDF Query Language (SPARQL) [53] to extract the content from reference ontology instance, and XSLT to generate a message instance matching the content reference ontology instance.

The approach presented in this paper can also provide support for SAWSDL-based reconciliation between XML messages. The Message Metamodel captures all the XML Schema concepts relevant to SAWSDL (e.g., data types or element definition) and forward and backward reconciliation rules are actually "lifting" and "lowering" rules. The "liftingSchemaMapping" and “loweringSchemaMapping” extension attributes, which refer to external resource where reconciliation rules are stored, may be added to the original schemas by using the knowledge captured in message-schema model and the reconciliation rule base. However, in this case, an actual run-time “lifting” and “lowering” must be supported by the run-time semantic reconciliation architecture proposed in this paper.

The SAWSDL-like reconciliation for non-XML message representation (e.g. EDI) can also be supported by applying the approach in this paper. The proposed Message Metamodel-enhanced semantic reconciliation is agnostic to message-representation language and the “liftingSchemaMapping” and “loweringSchemaMapping” may be applied to non-XML message representations. However, the extensional attributes are not yet defined for other message specification standards but only for XML Schema.

The work in this paper mostly follows the Automated Methods for Integrating Systems (AMIS) project [26] that studied automated methods for supply-chain integration, and it is directly motivated by the results of the ATHENA B5.10 validation project [14]. AMIS proposed the any-to-one model for semantics-based integration, and introduced a model of a message as a Local Conceptual Model (LCM) *together* with a Local Engineered Interface Model (LEM). LCM identifies business entities, properties, and relationships referenced in services, while LEM identifies functions or services provided by a tool, services it expects to use, specification models for those interfaces, and identification of the interface technologies (such as Webservices or ebMS). The work in this paper focuses only on the software applications with message-based interfaces.

In AMIS, the LEM concepts are mapped to the local conceptual models concepts , and the local conceptual models concepts are mapped to the reference ontology concepts for semantic reconciliation. In fact, we developed the LEM for the business messages – the Message Metamodel , and captured it as an OWL ontology. However, in contrast to AMIS idea, in this work the “engineered interface” is directly used to define mappings to the reference ontology for semantic reconciliation. That is, OWL/RDF message-schema models are used to define logic rules-based transformations to the reference ontology to demonstrate operational reconciliation between two message-based applications. The use of the message-schema models for the reconciliation rules definition was possible as a message-schema model captures the message structure, which is actually the essential intent of the Local Conceptual Model when it is considered as the model of messages for the semantic reconciliation.

Semantic web services (SWS) also have become a key technology for enabling the workflow integration of supply chains [41]. The SWS frameworks, such is Web Service Execution Environment [42], may provide automated web service discovery, execution, composition and interoperation in the supply chains environments. This paper, however, is only concerned with the semantic reconciliation of business messages in the traditional web-service environments.

Table 1 summarizes this comparison of related semantic reconciliation architectures and approaches.

Table 1 The comparison of the approaches

	Architectural Model Demonstrated (ATO Any-to-one) (ATA any-to-any)	**Toolset supports other architectural models** (Y-yes, N-no)	**Model of a message** (LCM –local conceptual model)	**Supported message-representation language** (XML,/EDI, or A when agnostic)	**Suitable for the creation of schema-conformant messages from semantic reconciliation output** (Y-yes, N-no)	**Reconciliation specification**	**Reconciliation execution**	**Ontological language**	**Message-schema components annotation** (S-supports) (M-may support) (N- cannot support)	**Suitable for SAWSDL framework** (Y-yes, N-no, E is can be extended) (B-based)	**Supports non-standard based message schemas reconciliation** (Y-yes, N-no)
Harmonize [23] (Mafra)	ATO	Y	LCM	A	N	bridge axioms defined by Semantic Bridge Ontology	bridge axioms interpretation	RDFS	N	N	Y
Artemis [22] (OWLmt)	ATA	Y	LCM	A	N	bridge axioms defined by Mapping Schema Definition Ontology	OWL-QL/ bridge axioms interpretation	OWL, RDFS	N	N	Y
ATHENA A3 (Astar/ Argos)	ATO	N	LCM	A	Y (used [25] as a workaround)	annotations between LCM and RO	Jena rule-based reasoning	OPAL-based[50] OWL, RDFS	N	N	Y
METEOR-S [44]	ATO	N	-	XML	Y	SAWSDL	XQuery/ XSLT	OWL, RDFS	S	Y	Y
Yarimagan [37]	ATA	N	Component ontology	A	N	no specification	description-logic reasoning	OWL DL	M	E	N
Anicic [20]	ATA	N	LCM	A	N	No specification(merging)	description-logic reasoning	OWL DL	N	N	N
Ye and Yang [33]	ATO	Y	LCM	A	N	no specification (executable mapping)	SWRL rule-based reasoning	OWL	N	N	Y
Oh and Yee [12]	ATA	Y	LCM	A	Y (used [25])	no specification (executable mapping)	Jena rule-based reasoning	RDFS	N	N	Y
AMIS [26]	ATO	N	LCM	A	not discussed	composition of software patterns [46]	generated code (C)	not discussed	N	not discussed	Y
Our approach	ATO	Y	Message Metamodel	A	Y	no specification (executable mapping) Annotation future work	Jena rule-based reasoning	OWL, RDFS	M	E	Y

8. Conclusion

The Message Metamodel provides a unique contribution toward the design-time and run-time activities of the proposed semantic reconciliation architecture.

The experimental results showed that the proposed semantic reconciliation architecture, enhanced with the Message Metamodel as the model of business message, contributes to the semantic reconciliation by ensuring that (1) the reconciliation tasks are isolated from and independent of differently specified and represented messages, and (2) the proposed solution ultimately produces message schema-conforming message instance from reconciliation output, in the syntax of target message instance.

This is in opposition to approaches based on a local conceptual model of business messages as a highly abstract model that proved insufficient to capture the required information about messages and to enable effective mapping between syntactic elements and content concepts, and thus, insufficient for the semantic reconciliation and semantic annotation of business messages.

Also, the proposed model of a message does not require additional reconciliation rules that generate axioms for the reconciled message to carry purely data-representation and formatting rules through the entire semantic reconciliation, which was reported as an unwanted behavior when a local conceptual model is used as the model of message.

As the Message Metamodel form captures the original structure of a message, a reconciliation rules definition tool may visualize the original message structure to the reconciliation engineer when rules are being specified. The original message structure is likely to be more familiar to the reconciliation engineer then a structure of the message captured in a local conceptual model (when a local conceptual model is used as a model of message).

However, writing reconciliation rules, which are detailed and formal, is definitely not an easy task, even for a highly skilled rule engineer. The future work will be focused on the development of a tool for the reconciliation rules definition. The plan is to take the approach where reconciliation rules are being derived from semantic annotation expressions, as the semantic annotation expressions that describe the meaning of message elements may be reused for different purposes in heterogeneous B2B environments, not only for reconciliation rules generation. Most of the forward reconciliation rules can be derived directly from the annotation expressions and the reverse forward rules can provide most of the backward rules.

In summary, we believe that the Message Metamodel and the semantic reconciliation and message semantics annotation based on the Message Metamodel, as the model of message schemas and message instances, may provide significant support for industry integration requirements. This is, a common ontological representation of the present well-established message-representation standards (XML, EDI, ASN.1); a transparent reconciliation and annotation approach for the well-established message-representation standards; an improved runtime reconciliation capabilities; and an approach to multi-purpose reuse of annotation expressions (e.g., reconciliation, semantic querying, schema component discovery and reusability).

***Disclaimer**: Certain commercial and open source software products are identified in this paper. These products were used only for demonstration purposes. This use does not imply approval or endorsement by NIST, nor does it imply these products are necessarily the best available for the purpose.*

References

1. Standards for Technology in Automotive Retail (STAR), online at www.starstandard.org/, accessed August 2008
2. Automotive Industry Action Group (AIAG), online at www.aiag.org/, accessed August 2008
3. OASIS Universal Business Language (UBL), online at www.oasis-open.org/committees/ubl/, accessed August 2008
4. United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT), online at www.unece.org/trade/untdid/, accessed August 2008
5. Kantor, M, James H., B., (1996). Electronic Data Interchange (EDI). National Institute of Standards and Technology, online at www.itl.nist.gov/fipspubs/fip161-2.htm/, accessed August 2008
6. Extensible Markup Language (XML), online at www.w3.org/XML/, accessed August 2008
7. Abstract Syntax Notation One (ASN.1), online at http://asn1.elibel.tm.fr/, accessed August 2008
8. Web ontology Language (OWL), online at www.w3.org/TR/owl-features/, accessed August 2008
9. RDF Vocabulary Description Language (RDF Schema), online at www.w3.org/TR/rdf-schema/, accessed August 2008.
10. ATHENA Knowledge Support and Semantic Mediation Solutions - Deliverables D.A3.2-D.A3.5, January 2006, online at www.modelbased.net/aif/, accessed August 2008
11. Vujasinovic, M., Ivezic, N., Kulvatunyou, B., Barkmeyer, E., Missikoff, M., Taglini, F., Marjanovic, Z., Miletic, I., (2009). Semantic-Mediation Architecture for Interoperable Supply-Chain Applications, to appear in Int'l Journal of Computer Integrated Manufacturing, accepted October 2008
12. Oh, S-C., Yee, S-T., (2008), Manufacturing interoperability using a semantic mediation, Int'l Journal of Advanced Manufacturing Technology, Vol. 39, pp. 199-210
13. Ray, S., Jones, A., (2006), Manufacturing interoperability, Journal of Intelligent Manufacturing, Vol. 17, Number 6, pp. 681–688
14. ATHENA B5.10 - Inventory Visibility Sub-Project: IV&I End-to-End Interoperability Demonstration including Conformance Testing Demonstration, online at http://xml.aiag.org/athena/resources/WD.B5.7.6--InteropAndConformanceTestDemo.pdf/, accessed August 2008
15. Turtle - Terse RDF Triple Language, online at www.dajobe.org/2004/01/turtle/, accessed August 2008
16. Jena rule language, online at http://jena.sourceforge.net/, accessed August 2008
17. Barkmeyer, E., Kulvatunyou, B., (2007). An Ontology for the e-Kanban Business Process, NIST Internal Report 7404, National Institute of Standards and Technology, online at www.mel.nist.gov/msidlibrary/doc/NISTIR_7404.pdf/, accessed August 2008
18. Vujasinovic, M., Barkmeyer, E., (2009). Message Metamodel, NIST Internal Report, National Institute of Standards and Technology
19. Hameed, A., Preece, A.D., Sleeman. D.H., (2004). Ontology Reconciliation, Handbook on ontologies, Springer, pp. 231– 250
20. Anicic, N., Marjanovic, Z., Ivezic, N., Jones, A., (2007). Semantic Enterprise Application Integration Standards, Int'l Journal of Manufacturing and Technology, Vol. 10(2-3), pp. 205-226
21. Vetere, G., Lenzerini, M., (2005). Models for semantic interoperability in service-oriented architectures, IBM Systems Journal, Vol. 44(4), pp. 887-903

22. Bicer, V., Laleci, B.G., Dogac, A., Kabak, Y., (2005). Artemis Message Exchange Framework: Semantic Interoperability of Exchanged Messages in the Healthcare Domain, SIGMOD Record Vol. 34(3), pp. 71-76
23. Fodor, O., Werthner, H., (2004). Harmonise: A Step Toward an Interoperable E-Tourism Marketplace, Int'l Journal of Electronic Commerce, Vol. 9(2), pp. 11-39
24. Maedche, A., Motik, B., Silva N., Volz, R., (2002). MAFRA - A MApping FRAmework for Distributed Ontologies, Proc. of the EKAW 2002, LNCS 2473, Springer, pp. 235–250
25. Miletic, I., Vujasinovic, M., Ivezic, N., Marjanovic, Z., (2007). Enabling Semantic Mediation for Business Applications: XML-RDF, RDF-XML, and XSD-RDFS Transformation, Proc. of the Int'l Conf. IESA, Springer, pp. 483-494
26. D. Libes, Barkmeyer, E., Denno, P., Flater, D., Steves, M.P., Wallace, E., Feeney, A.B., (2004). The AMIS Approach to Systems Integration, NIST Internal Report 7101, National Institute of Standards and Technology, online at www.mel.nist.gov/msidlibrary/doc/nistir7101.pdf/, accessed August 2008
27. Bowers S., Delcamre L., (2000). Representing and transforming model based information, Proc. of the 4th European conference on research and advanced technology for digital library (ECDL-2000), Lisbon, Portugal, pp. 5–18
28. Kensche, D., Quix. C., Chatti, M.A., Jarke, M., (2005). GeRoMe: A generic role based metamodel for model management, Proc. of the 4th Int'l Conf. on Ontologies, Databases, and Applications of Semantics (ODBASE), LNCS Vol. 3761, pp. 1206–1224
29. Melnik, S., Rahm, E., Bernstein, P.A., (2003). Rondo: A programming platform for model management, Proc. of the 22nd Int'l Conf. on Management of Data (SIGMOD), pp. 193–204
30. Horrocks, I., Patel-Schneider, Boley, H., Tabet, S., Grosf, B., Dean, M., (2004). SWRL: a semantic web rule language combining OWL and RuleML, WWW Consortium Member Submission, online at www.w3.org/Submission/SWRL/, accessed August 2008
31. ISO/IEC 19502 Meta Object Facility (MOF), online at www.omg.org/spec/MOF/, accessed August 2008
32. Bohring, H., Auer, S., (2005). Mapping XML to OWL Ontologies. Marktplatz Internet: Von e-Learning bis e- Payment. Leipziger Informatik-Tage (LIT2005), Leipzig, Germany, pp.147-156
33. Ye, Y., Yang, D., Jiang, Z., Tong., L., (2007). An ontology-based architecture for implementing semantic integration of supply chain management, Int'l Journal of Computer Integrated Manufacturing, Vol. 21(1), pp. 1-18
34. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R., (1999). What are ontologies, and why do we need them?, IEEE Intelligent Systems, Vol. 14(1), pp. 20-26
35. Foxvog, D., Bussler, C., (2005). Ontologizing EDI: first steps and initial experience, Proc. of the Int'l Workshop on Data Engineering Issues in E-Commerce, pp: 49 - 58
36. Karthick, S., (2006). Ontologizing XML Using Mediation Patterns, First International Workshop on Ontologizing Industrial Standards OIS 2006
37. Yarimagan, Y., (2008). Semantic Enrichment for the Automated Customization and Interoperability of UBL Schemas, PhD Thesis, Dept. of Computer Engineering, METU, March 2008
38. American National Standards Institute Accredited Standards Committee X12 (ANSI ASC X12), online at www.x12.org/, accessed August 2008
39. The Web Service Modeling Language WSML, online at www.wsmo.org/TR/d16/d16.1/v0.2/, accessed August 2008
40. Kopecky, J., Vitvar, T., Bournez, C., Farrell, J., (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema, IEEE Internet Computing, Vol 11(6), pp. 60-67
41. McIlraith, S.A., Son, T.C. et.al., (2001). Semantic Web services, IEEE Intelligent Systems, Vol. 16(2), pp. 46-53
42. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C., (2005). WSMX - a semantic service-oriented architecture, Proc. of the 2005 Int'l Conf. on Web Services, Vol. 1, pp. 321-328

43. Verma, K., Sheth, A., (2007). Semantically Annotating a Web Service, IEEE Internet Computing, Vol. 11(2), pp. 83-85
44. Patil, A., Oundhakar, S., Sheth, A., Verma, K., (2004), Meteor-s web service annotation framework, Proc. of the 13 Int'l Conf. on WWW, pp. 553-562
45. ISO 10303-11:2004 Industrial automation systems and integration -- Product data representation and exchange -- Part 11: Description methods: The EXPRESS language reference manual, online at www.iso.org/, accessed August 2008
46. Flater, D., (2004). Automated Composition of Conversion Software, NIST Internal Report 7099, National Institute of Standards and Technology, online at www.mel.nist.gov/msidlibrary/doc/nistir7099.pdf/, accessed August 2008
47. XML Schema Definition API, online at www.eclipse.org/xsd/, accessed August 2008
48. Document Object Model (DOM), online at www.w3.org/DOM/, accessed August 2008
49. JavaScript Object Notation (JSON), online at www.json.org/, accessed August 2008
50. D'Antonio, F., Missikoff, M., Taglino, F., (2007). Formalizing the OPAL eBusiness ontology design patterns with OWL, Proc. of the Int'l Conf. IESA, Springer, pp. 345-356
51. XSL Transformations (XSLT) Version 1.0, online at http://www.w3.org/TR/xslt, accessed August 2008
52. XQuery 1.0: An XML Query Language, online at http://www.w3.org/TR/xquery/, accessed August 2008
53. SPARQL Query Language for RDF, online at http://www.w3.org/TR/rdf-sparql-query/, accessed August 2008