Semantic-Mediation for Standards-based B2B Interoperability

Marko Vujasinovic[1], Nenad Ivezic[1], Boonserm Kulvatunyou[2], Edward Barkmeyer[1], Michele Missikoff[3], Francesco Taglino[3], Zoran Marjanovic[4], Igor Miletic[4]

[1] Manufacturing Engineering Laboratory
National Institute of Standards and Technology, Gaithersburg MD, USA
{marko.vujasinovic, nenad.ivezic, boonserm.kulvatunyou, edward.barkmeyer}@nist.gov

[2] Oracle Corporation
Belmont CA, USA
serm.kulvatunyou@oracle.com

[3] Laboratory for Enterprise Knowledge and Systems
Consiglio Nazionale delle Ricerche, Rome, Italy
{missikoff, taglino}@iasi.cnr.it

[4] Department of Information Systems, FON
University of Belgrade, Belgrade, Serbia
marjanovic.zoran@fon.rs, igor.miletic@brezasoftware.com

Abstract. The authors discuss a semantic-mediation architecture to advance traditional approaches for standards-based business-to-business (B2B) interoperability. The architecture is supported by the ATHENA Knowledge Representation and Semantics Mediation tool suite. Initial experimentations with the architecture and the toolset offer discussions of key architectural and functional aspects and suggest directions for future tools enhancements.

Introduction

Most existing business software applications are not interoperable as they use proprietary data models and message sets for B2B communication. As a response, industry consortiums and standards development organizations (SDOs), such as Automotive Industry Action Group (AIAG[7]), publish standard messages for interoperable B2B data exchanges to accomplish standards-based interoperability.

The current approach for standards-based interoperability unfolds at three levels;

- *Business process and data modeling.* An SDO defines business process models and conceptual models of data being exchanged, which, in turn, allows definition of standard message schemas (also called Business Object Documents or BODs) that specify standard messages.
- *BODs adoption and mapping*. An industry consortium adopts or extends the BODs for the industry-specific business process. Then, application providers interpret the BODs and define design-time mappings between proprietary application interfaces and the BODs.
- *Run-time execution.* Providers use the proprietary-to-BOD interface mappings to implement message content transformations. These transformations effectively implement standard-conformant message interfaces.

However, this traditional approach has four shortcomings that affect adoption of the standards. (s1) Informal specification of the business domain concepts using syntactic notations to convey data-exchange requirements and their business meaning leads to ambiguity and misinterpretation[1]. (s2) Informal and non machine-understandable annotation of the meaning of the standard or proprietary message elements leads to misinterpretations of the message semantics and application integration problems[2] - homonymy and synonymy issues, for example. (s3) Manual and hard-coded mappings between the proprietary and standard message elements lead to error-prone and labor-intensive implementations[3]. (s4) High inter-dependence between proprietary and standard message interfaces (at execution-platform, technology, terminology, adopted message-exchange standard, and message syntax levels) implies inflexible, tightly-coupled integrations.

To address these shortcomings in standards-based interoperable message exchanges, we explored a novel semantic-mediation architecture for standards-based interoperability. As a basis for exploration, we executed a small-scale industrial

message exchange scenario. In this paper, we discuss key aspects of the architecture, approaches taken, and recommended advances to handle realistically large-scale industrial situations.

Semantic-mediation architecture for standards-based interoperable applications

The proposed semantic-mediation architecture builds on the traditional approach and introduces new activities at each level (now indicated by new titles).

- *Business-domain ontology modeling.* An SDO captures the intended meaning of the data exchange, and creates a reference ontology (RO) on the basis of the business process model and the data-exchange requirements (shown as the 'Create' activity in Figure 1). The reference ontology specifies, formalizes, and explicates domain business concepts and concept relationships. The reference ontology is publicly available to the application providers.
- *Design-Time semantic-mediation specification.* Following adoption of the BODs, the SDO annotates the BOD semantics (step B_1 in Figure 1) by relating each BOD element to the corresponding concept in the reference ontology. Then, the SDO uses the BOD annotations to define publicly avaiable reconciliation rulesets (step C_1 in Figure 1) for the transformation of a BOD-conformant message to a reference ontology instance and a reference ontology instance to a BOD-conformant message. The application providers annotate the semantics of their proprietary-message interfaces (step B_2 in Figure 1). The providers use their respective interface annotations to define their proprietary reconciliation rulesets (step C_2 in Figure 1) for the transformation of a proprietary message to a reference ontology instance and a reference ontology instance to a proprietary message. Effectively, the proprietary and public rulesets define transformations between proprietary messages and BOD-conformant messages via the reference ontology.
- *Run-Time semantic-mediation execution*. When an application is sending, the semantic-mediator uses the appropriate reconciliation rulesets to translate the proprietary message to the reference ontology instances and those instances to the BOD-conformant message. When an application is receiving, the semantic-mediator translates from the BOD-conformant messages to the reference ontology instance and then to the proprietary message. Effectively, the semantic-mediator implements the standard-conformant message interface for the application.

To address the aforementioned shortcomings (s1-s4), our proposed architecture introduces the five advances. (a1) *The formal specification of the business domain concepts* in the reference ontology provides a basis for unambiguous interpretation of the data-exchange artifacts. (a2) The ontology-based semantic annotation of message schemas provides *machine-understandable annotation expressions* that formally and precisely describe the meaning of message elements. (a3) The *precise specification of semantic reconciliation rules* are enabled by machine-understandable annotation expressions. (a4) A*utomated and consistent standard-interface implementation through the reconciliation rules execution* moves the engineering effort from implementation time to modeling and design time. (a5) *Semantic-mediation* reconciles terminological, structural, semantic, and representational differences between message specifications.

In summary, the key steps to achieve standards-based interoperability by the semantic-mediation are the reference ontology development, semantic annotation of message elements, and definition of reconciliation rules. The total effort is distributed between application providers and the SDO. The SDO creates the ontology, annotates BODs and defines reconciliation for BOD-conformant messages; this is done initially, and only once. To implement the standard-conformant message interface for the application, each application provider must annotate its proprietary-message interface, define reconciliation rules for its proprietary messages using the reference ontology, and deploy the semantic-mediator. This must be done once for each reference ontology and for each proprietary interface. Translations into and out of tool-specific forms may also be required. Obviously, the scale of effort will depend on the complexity of the proprietary messages and the ontology.

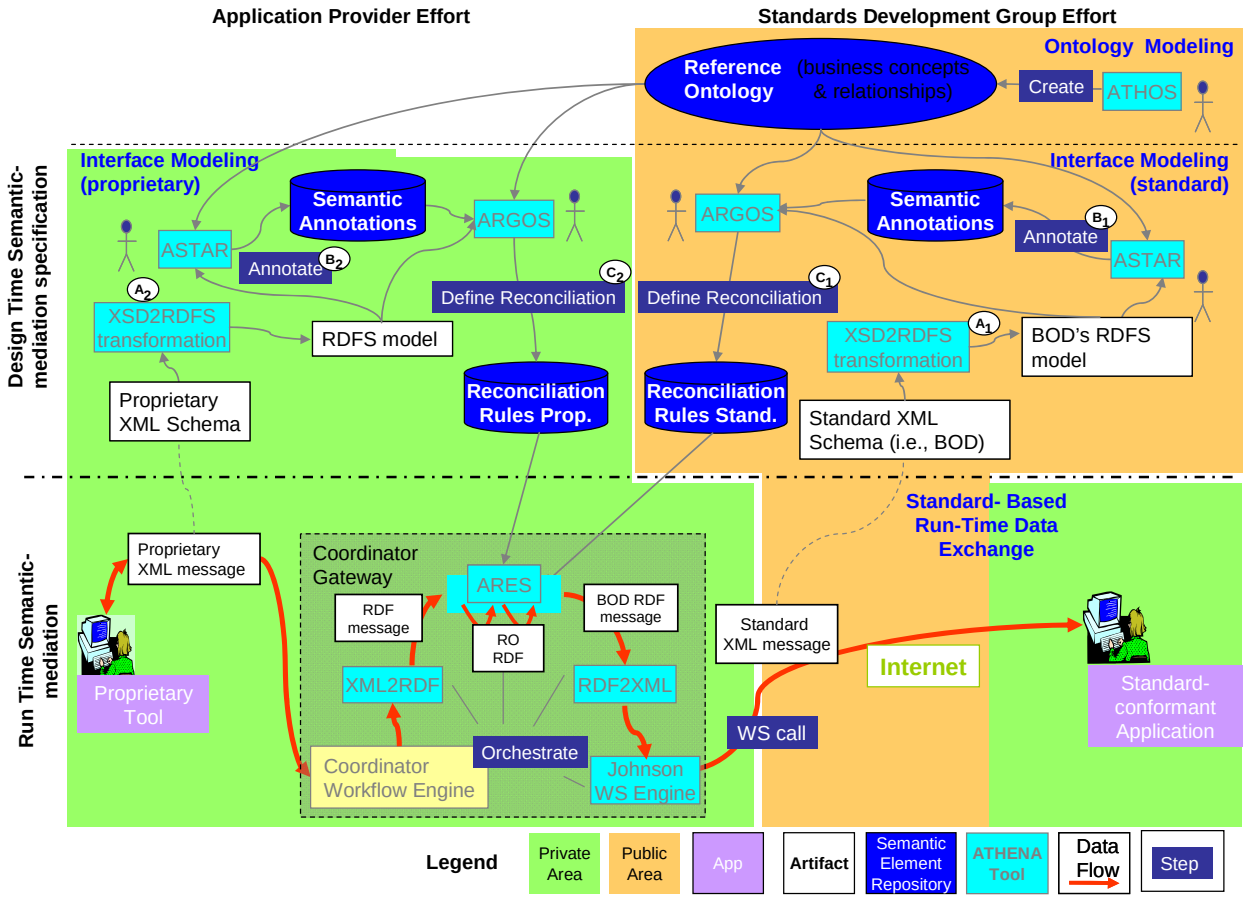

Figure 1 Semantic-Mediation for standards-based interoperability

Supporting tools

We implemented the proposed architecture using a suite of tools developed as part of the ATHENA project[4]. These tools facilitate semantics-based reconciliation of RDF (Resource Definition Framework) documents (Figure 1). At design-time, we used the ATHOS ontology development tool (http://leks-pub.iasi.cnr.it/Athos), the ASTAR semantic annotation tool (http://leks-pub.iasi.cnr.it/Astar), and the ARGOS reconciliation specification tool (https://services.txt.it/argos). At run-time, we employed the ARES reconciliation execution engine (http://euproj.gformula.com/athena/ares), and the Johnson Web Service (WS) execution engine to support WS interfaces.

ATHOS relies on the OPAL[5] ontology modeling language to construct ontologies through predefined business categories and inherent constraints. OPAL is built on top of the OWL (Web Ontology Language), which gives a formal basis to the ATHOS-developed ontologies.

ASTAR sets up semantic correspondences, semantics annotation expressions, between the RDFS (Resource Description Framework Schema) model concepts and the reference ontology concepts. ASTAR provides a graphical annotation environment, visualisation of RDFS models, and semi-automatic support to define annotation expressions.

ARGOS provides a graphical environment for reconciliation rules specification. ARGOS visualizes RDFS models and reference ontology and assists in creating reconciliation rulesets that transform RDF documents to and from reference ontology instances (*forward-backward* rulesets). ARGOS is driven by annotation expressions from ASTAR.

ARES performs the actual RDF-to-RDF document reconciliation by executing declared forward and backward rulesets on RDF documents.

These tools provided many of the capabilities needed to support our B2B integration requirements. However, our B2B software applications also used XML (Extensible Markup Language) messages,

besides RDF messages. We needed to build some additional tools. The first tool, **XSD2RDFS**[6], transformed specific message-element definitions in XML Schema to the corresponding message-elements' concepts in the RDFS model (i.e. to *conceptual message-schema*). The second and third tools - **XML2RDF**[6] and **RDF2XML**[6] - transformed XML messages to and from the corresponding RDF documents, respectively. To integrate all of the run-time tools, we developed a **Coordinator Gateway** to transform a proprietary message into a standard-conformant message.

eKanban Experimental pilot

To assess representational capabilities of our architecture, we executed an experimental pilot based on the AIAG eKanban Inventory Visibility and Interoperability (IV&I) business scenario[7]. The AIAG defined a set of standard BOD messages for the electronic Kanban (eKanban) business process that regulates the flow of goods from suppliers to match actual usage by customers.

We employed two independently developed applications capable of sending and receiving only their proprietary *AuthorizeKanban* message, and one standard-conformant application capable of receiving the standard BOD *AuthorizeKanban* message:

- Apolon open source application with an RDFS-based proprietary message interface.
- General Motors' experimental application (GM) with an XML Schema-based proprietary message interface.
- Ford Test Harness (FTH) with an XML Schema-based BOD-conformant message interface.

We adopted the following scenario: Apolon (running in Serbia) exchanges a message with the FTH (running in Maryland, USA), and the GM application (running in Michigan, USA) exchanges a message with the FTH and Apolon.

First, we used the ATHOS tool to develop the **eKanban Reference Ontology**[8], which formally captured the business conceptual model for the eKanban process.

Then, we performed the **design-time** steps. First, we completed XSD2RDFS transformation of the BOD and GM *AuthorizeKanban* XML schemas to the corresponding RDFS *conceptual message-schemas* (steps A_1, A_2 in Figure 1). Next, we completed annotation of the BOD, GM, and Apolon *AuthorizeKanban* RDFS *conceptual message-schemas* by using ASTAR (steps B_1, B_2 in Figure 1). Then, reconciliation specification (steps C_1, C_2 in Figure 1) was completed using ARGOS to create (a) *forward* rules to transform data from the GM, Apolon, and BOD *AuthorizeKanban* RDF documents to the reference ontology instances, and (b) *backward* rules to transform data from the reference ontology instances to Apolon and BOD *AuthorizeKanban* RDF documents.

At **run-time**, Coordinator Gateway orchestrated a sequence of transformations and reconciliations, as shown in Table 1. Each application had its own appropriately configured Coordinator. The semantic-mediation was successful; applications sent and received messages in their proprietary formats. More importantly, these messages were conformant to the adopted exchange standard.

Table 1 Message flow and executed transformation inside the Coordinator: blue font indicates the sender and the messages sent; red font indicates the receiver and the messages received; black font indicates the intermediate transformations and data formats.

Scenario	Message	Input (relative to RO) message transformation	Message	Reconciliation	Reconciled message in ontology (RO) form	Reconciliation	Message	Output (relative to RO) message transformation	Message
GM sends to Apolon	XML GM →	XML2RDF →	RDF GM →	GM FORWARD →	RDF RO →	BOD BACKWARD →	RDF BOD →	RDF2XML →	standard XML BOD →
(continued from above) ↓	standard XML BOD →	XML2RDF →	RDF BOD →	BOD FORWARD →	RDF RO →	APOLON BACKWARD →	RDF APOLON →	**not needed** →	RDF APOLON →
Apolon sends to FTH	RDF APOLON →	**not needed** →	RDF APOLON →	APOLON FORWARD →	RDF RO →	BOD BACKWARD →	RDF BOD →	RDF2XML →	standard XML BOD →
GM sends to FTH	XML GM →	XML2RDF →	RDF GM →	GM FORWARD →	RDF RO →	BOD BACKWARD →	RDF BOD →	RDF2XML →	standard XML BOD →

Key aspects and findings

Central to successful use of the proposed architecture in realistically complex B2B integration cases are message-representation transformation, message semantics annotation, and message reconciliation specification.

A. Message representation transformation

Message representation transformation is an automated process that transforms a message schema or message instance into a message representation form required by the semantic-mediation tools. That form is typically aligned with the ontology representation language. A key challenge here is to develop a general and flexible transformation that abstracts from the unnecessary message syntax details while maintaining the essential schema or instance information.

Approach. The XSD2RDFS, XML2RDF, and RDF2XML message representation transformations were developed because the ASTAR and ARGOS tools dealt with RDFS models and messages in the form of the RDF documents. To transform an XML message schema to an RDFS *conceptual message-schema*, XSD2RDFS builds an 'internal tree' structure that reflects the XML message structure. That tree nodes represent XML element and attribute definitions and each node encapsulates name, datatype, and namespace for the corresponding element or attribute. Then, the tool transforms that tree into RDFS *conceptual message-schema* through predefined transformation rules and the 'extended names' naming convention[6]. Listing 1 shows an XSD2RDFS example.

Even though the functionality of XSD2RDFS was driven by specific requirements from ASTAR and ARGOS, the tool is very general – it can transform any given XML schema into corresponding RDFS *conceptual message-schema*.

Listing 1 Portion of the XSD2RDFS transformation for the GM schema. xsd:elements are transformed into corresponding RDFS classes (e.g., gmSyncShipmentSchedule to gmSyncShipmentSchedule_sender). Each simple xsd:element is transformed into a corresponding RDFS data-property (e.g., gmSyncShipmentSchedule_sender_DUNS_sValue for the DUNS element). Each XML parent-child relation is transformed into an RDFS object-property (e.g., gmSyncShipmentSchedule_sender_DUNS_PROP RDFS property for the sender-DUNS parent-child XML relation)

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

 <xsd:element name="gmSyncShipmentSchedule">
  <xsd:complexType>
   <xsd:sequence>
    …
    <xsd:element name="sender"
         type="tns:partner"/>
    ….
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

<xsd:complexType name="partner">
  <xsd:sequence>
   <xsd:element name="DUNS" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
 …..
</xsd:schema>
```

→ XSD2RDFS →

```
<rdf:RDF xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns="http://www.nist-athena-ivi.com/rdfs#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
<rdfs:Class rdf:about="http://www.nist-athena-ivi.com/rdfs#gmSyncShipmentSchedule">
</rdfs:Class>
...
<rdfs:Class rdf:about="
http://gm.com/gmSyncShipmentSchedule/rdfs#gmSyncShipmentSchedule_sender">
</rdfs:Class>
<rdfs:Class rdf:about=" http://gm.com/gmSyncShipmentSchedule
/rdfs#gmSyncShipmentSchedule_sender_DUNS">
</rdfs:Class>
 ....
<rdf:Property rdf:about=" http://gm.com/gmSyncShipmentSchedule
/rdfs#gmSyncShipmentSchedule_sender_DUNS_sValue">
  <rdfs:domain rdf:resource= http://gm.com/gmSyncShipmentSchedule
/rdfs#gmSyncShipmentSchedule_sender_DUNS"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
...
<rdf:Property rdf:about=" http://gm.com/gmSyncShipmentSchedule
/rdfs#gmSyncShipmentSchedule_sender_PROP">
  <rdfs:domain rdf:resource=" http://gm.com/gmSyncShipmentSchedule
/rdfs#gmSyncShipmentSchedule"/>
  <rdfs:range rdf:resource="http:// http://gm.com/gmSyncShipmentSchedule
/rdfs#gmSyncShipmentSchedule_sender"/>
</rdf:Property>
<rdf:Property rdf:about=" http://gm.com/gmSyncShipmentSchedule
/rdfs#gmSyncShipmentSchedule_sender_DUNS_PROP">
  <rdfs:domain rdf:resource=" http://gm.com/gmSyncShipmentSchedule
/rdfs#gmSyncShipmentSchedule_sender"/>
  <rdfs:range rdf:resource=" http://gm.com/gmSyncShipmentSchedule
/rdfs#gmSyncShipmentSchedule_sender_DUNS"/>
</rdf:Property>
...
</rdf:RDF>
```

For the runtime message-representation transformations, we defined the XML2RDF and RDF2XML transformation-naming convention[6] that transformation algorithms used to create RDFS-conformant RDF documents for the reconciliation input, and XML schema-conformant XML documents for the reconciliation output. Listing 2 shows the example of XML2RDF and RDF2XML

transformations outcome. Neither the XML schema nor the RDFS *conceptual message-schema* was required during run-time transformation, which made the approach very flexible.

Listing 2 Portion of the XML2RDF and RDF2XML transformation for the GM's message

```
<?xml version="1.0" encoding="UTF-8"?>
<gmSyncShipmentSchedule
        xmlns="http://gm.com/gmSyncShipmentSchedule/">
 ...
 <sender>
 <DUNS>General Motors</DUNS>
 </sender>
 ..
 </gmSyncShipmentSchedule>
```

XML2RDF →

← RDF2XML

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://www.defaultnsrdf.rdf#">
 <gmSyncShipmentSchedule>
  ...
  <gmSyncShipmentSchedule_sender_PROP>
   <gmSyncShipmentSchedule_sender rdf:ID="X2">
    <gmSyncShipmentSchedule_sender_DUNS_PROP>
     <gmSyncShipmentSchedule_sender_DUNS rdf:ID="X3">
      <gmSyncShipmentSchedule_sender_DUNS_sValue
            rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
       General Motors
            </gmSyncShipmentSchedule_sender_DUNS_sValue>
     </gmSyncShipmentSchedule_sender_DUNS>
    </gmSyncShipmentSchedule_sender_DUNS_PROP>
   </gmSyncShipmentSchedule_sender>
  </gmSyncShipmentSchedule_sender_PROP>
  ....
 </gmSyncShipmentSchedule>
</rdf:RDF>
```

Findings. In executing the pilot, however, we discovered that the message representation transformation needs to preserve both message-structure and data-representation rules (originally specified in the message schema) in order to transform RDF documents to XML schema-conformant XML messages. Particularly, to produce an XML schema conforming message from reconciliation output, at least the following definition must be maintained: a) message structure, b) elements order, c) concepts (elements/attributes) names including namespaces, d) concept granularity (element vs. attribute), and e) formatting rules.

A *conceptual message-schema* doesn't capture this information. However, we were able to embed the structural and granularity characteristics in naming conventions - concepts' *extended-name* convention maintained message-structure definition, and the _ATTR suffix maintained elements vs. attributes distinctions. Additionally, we created reconciliation rules that generated additional RDF statements for the target RDF document to carry purely data-representation rules through the entire semantic-mediation - additional RDF statements carried the definition of namespaces and elements order.

Creating such additional reconciliation rules required undesired effort on behalf of a rule expert and additional knowledge about message-representation and formatting rules, beyond understanding the message structure and semantics. Neither our approach nor other similar transformation approaches[6], transform message schemas to a message-representation form that sufficiently capture information (a-e) and that is aligned with an ontology-representation language (OWL or RDFS).

Future direction: *Abstract message model.* To eliminate this additional work and to provide mediation for other message-formats, such as EDI (Electronic Data Interchange), we propose an *abstract message model*. This a*bstract message model* will not capture certain syntax-specific constructs of message schemas or messages. It will, however, faithfully capture (1) message schema concepts such as definition of elements/attributes, definition of complex/simple types, and information (a-e) from above; and, (2) message concepts such as elements, attributes, content, and values. The message-representation transformations shall then instantiate and populate the *abstract message model* with information from an actual message schemas or message instances. That further implies that the forward reconciliation shall be from, and the backward reconciliation shall be to, the *abstract message model* instance - GM and BOD *abstract message model* instances, for example. Then, the *abstract message model* instance, as an output from a reconciliation tool, will contain enough information so that the specific message-representation transformation needs to produce the schema-conformant message from that *abstract message model* instance.

B. Message semantics annotation

Message semantics annotation clarifies message semantics by associating each message element with a machine-understandable expression that represents its business meaning in terms of reference ontology concepts. The significant challenge is to have an annotation method that reduces human effort, detects semantic correspondences, provides sufficient expressivity, and allows multi-purpose usability of semantic annotation expressions.

Approach. The ASTAR annotation method is organized in two phases: diagnostic and remedial. The diagnostic phase aims to identify terminological, structural, and semantic mismatches between a *conceptual message-schema* and the reference ontology. In the remedial phase *conceptual message-schema* concepts are associated with expressions that represent their semantics in terms of the reference ontology. The remedial phase has four steps: terminological semantic annotation (TSA), path semantic annotation (PSA), simple semantic annotation (SSA), and full semantic annotation (FSA).

In the TSA step, the ASTAR tool contrasts the terms of the *conceptual message-schema* concepts with the terminology of reference ontology concepts and automatically detects lexical-terminological similarity among them. Then, we establish correspondences between the terms, which resolve terminological mismatches and further assisted in structural path matches identification in the PSA step.

In the PSA step, we consider the structures of the *conceptual message-schemas* and reference ontology, and associate one or more *conceptual message-schema* paths to one or more matching reference ontology paths, which resolve structural mismatches.

In the SSA step, all the path matches are further composed into *path expressions* by using *abstract operators*, which denote a data-transformation template needed at run-time. In this step, other semantics mismatches, which are mostly about the encoding and representation choices for information units (such as named values, time intervals) can also be noted by *abstract operators.*

Table 2 Portion of the GM's RDFS conceptual message-schema annotation

- TSA -		**Mismatch**
GM's RDFS concept	**Reference ontology concept**	
gmSyncShipmentSchedule	ShipmentSchedule, SyncShipmentSchedule_Message	
gmSyncShipmentSchedule_sender	SenderParty	
gmSyncShipmentSchedule_sender_DUNS	PartyId	
gmSyncShipmentSchedule_sender_PROP	relTo_SyncShipmentSchedule_Message_SenderParty	naming
gmSyncShipmentSchedule_sender_DUNS_PROP	relTo_ SyncShipmentSchedule_Message_SenderParty_PartyId	
gmSyncShipmentSchedule_sender_DUNS_sValue	has_identifier	
- PSA -		
GM's RDFS path	**Reference ontology path**	
gmSyncShipmentSchedule. gmSyncShipmentSchedule_sender_PROP. gmSyncShipmentSchedule_sender. gmSyncShipmentSchedule_sender_DUNS_PROP. gmSyncShipmentSchedule_sender_DUNS. gmSyncShipmentSchedule_sender_DUNS_ sValue:STRING	ShipmentSchedule. relTo_ShipmentSchedule_message_SyncShipmentSchedule. SyncShipmentSchedule_Message. relTo_SyncShipmentSchedule_Message_SenderParty. SenderParty. relTo_SyncShipmentSchedule_Message_SenderParty_PartyId.PartyId. has_identified:STRING	structuring
- SSA -		
gmSyncShipmentSchedule. gmSyncShipmentSchedule_sender_PROP. gmSyncShipmentSchedule_sender. gmSyncShipmentSchedule_sender_DUNS_PROP. gmSyncShipmentSchedule_sender_DUNS. gmSyncShipmentSchedule_sender_DUNS_sValue :STRING	= (abstract operator)	ShipmentSchedule. relTo_ShipmentSchedule_message_SyncShipmentSchedule. SyncShipmentSchedule_Message. relTo_SyncShipmentSchedule_Message_SenderParty. SenderParty. relTo_SyncShipmentSchedule_Message_SenderParty_PartyId.PartyId. has_identified:STRING
- FSA - **OWL DL semantic annotation expressions**		
STRING∩ (∃ inverseOf_gmSyncShipmentSchedule_sender_ DUNS.(gmSyncShipmentSchedule_sender_DUNS∩ (∃ inverseOf_ gmSyncShipmentSchedule_sender_DUNS_PROP. (gmSyncShipmentSchedule_sender∩ (∃ inverseOf_gmSyncShipmentSchedule_sender_ PROP(gmSyncShipmentSchedule))))))	<owl: equivalentClass>	STRING∩ (∃ inverseOf_has_Id_identified.(PartyId∩(∃ inverseOf_ relTo_ SyncShipmentSchedule_Message_SenderParty_PartyId.(SenderParty∩(∃ inverseOf_relTo_SyncShipmentSchedule_Message_SenderParty.(Syn cShipmentSchedule_Message∩(∃ inverseOf_relTo_ShipmentSchedule message_SyncShipmentSchedule(ShipmentSchedule)))))))

In the FSA step, *path expressions* are translated into OWL semantic annotation expressions. OWL allows encoding of the actual relationships between the semantic concepts such as equivalence, subsumption, or overlap. Table 2 shows an example of the mismatches and the four-step annotation for a portion of the GM *conceptual message-schema*.

Findings. Path matching required significant human effort since ASTAR provided combinations of all paths through the reference ontology. This led to overwhelming complexity in the annotation activity. A semantic-annotation tool should allow the user to steer the path development rather than present all possible paths. Nevertheless, by using ASTAR, we successfully annotated all the *conceptual message-schemas*.

ASTAR uses OWL as an internal representation language for semantic annotation expressions. However, OWL has shortcomings such as interpretation-framework dependence, complexity, limited expressivity, and non-executability[9]. The message semantic annotation, however, needs an interchangeable, executable, expressive, but simple representation format.

Significantly, ASTAR doesn't provide for semantic annotation of the actual (XML/EDI) message schema components and message elements, but annotation of *conceptual message-schemas*. Hence, the usability of annotations in the reconciliation that generates message-schema conformant messages is directly affected. The important information about the message element definitions, message structure, and data-representation cannot be engineered from *conceptual message-schemas*. Also, multi-purpose usability of such annotations for other purposes - such as semantic querying over XML/EDI messages and schema components discovery - is impossible.

Future direction: A message semantics annotation method enabled by the *abstract message model*. We propose a method that annotates *abstract message model* instances (conformant with an actual message schema) rather than *conceptual message-schemas*. We believe that this method will improve the reconciliation run-time capabilities and multi-purpose usability. Additionally, it will support three different types of annotation re-use (1) reuse for message components, – annotation of the base elements type definitions (e.g., the base type PartyType with its elements) could provide annotation re-use for the specific XML elements, which are declared for specific use context, whose type is that already annotated base type (e.g., SenderParty and ReceiverParty elements which base type is Party); (2) reuse across different, but overlapping message types, and (3) re-use of an entire set of annotations for a set of messages.

C. Message reconciliation specification

Messages reconciliation defines the forward and backward executable message-content-transformation rules. Forward rules describe how to obtain content of a concept appearing in a reference ontology instance by transforming the content from one or more message elements. Backward rules describe how to obtain content of a message element by transforming the content from one or more concepts appearing in the reference ontology instance. There are several simple transformation patterns including one-to-one, many-to-one, one-to-many; there are also more complex patterns including conversion functions. The significant challenge is to achieve automated rules generation.

Approach. ARGOS provided for semi-automatic specification of reconciliation rules based on semantic annotation expressions. The specification was performed by selecting an appropriate transformation pattern for one or more *conceptual message-schema* paths leading to the content to be transformed. The tool then created declarative run-time rules by substituting the *conceptual message-schema* path (or paths) and matching reference ontology path (or paths) into the template, on the basis of annotation expressions. ARGOS uses Jena as an executable rule-representation language (http://jena.sourceforge.net*).* Listing 3 shows Jena rule that specifies one-to-one mapping between the GM '*Sender.DUNS*' and reference ontology '*senderParty.PartyId.identifier*' concepts.

Listing 3 The one-to-one map Jena rule example

```
[rule:(?x rdf:type a:gmSyncShipmentSchedule)
      (?x a:gmSyncShipmentSchedule_sender_PROP ?y)
      (?y rdf:type a:gmSyncShipmentSchedule_sender)
      (?y a:gmSyncShipmentSchedule_sender_DUNS_PROP ?z)
      (?z rdf:type a:gmSyncShipmentSchedule_sender_DUNS)
      (?z a:gmSyncShipmentSchedule_sender_DUNS_PROP ?t)
      (?t rdf:type a:gmSyncShipmentSchedule_sender_DUNS)
      (?t a:gmSyncShipmentSchedule_sender_DUNS_sValue ?value)
  ->
     (?x rdf:type refOnt:ShipmentSchedule)
     (?x refOnt:relTo_ShipmentSchedule_message_SyncShipmentSchedule ?y)
     (?y rdf:type refOnt:SyncShipmentSchedule_Message)
     (?y refOnt:relTo_SyncShipmentSchedule_Message_SenderParty ?z)
     (?z rdf:type refOnt:SenderParty)
     (?z refOnt:relTo_SyncShipmentSchedule_Message_SenderParty_PartyId ?t)
     (?t ref:type refOnt:PartyId)      (?t refOnt:has_identified ?value)
]
```

Findings. Although ARGOS is a semi-automatic tool, all rules were instantiated manually. This, however, could be automated if annotation expressions and *abstract operators* were fully as

mapping directives. In fact, the instantiations of the one-to-one mappings could have been done automatically. Since 85% of pilot rules were one-to-one mappings, this would have reduced the reconciliation time considerably. Nevertheless, using ARGOS, we successfully created all required reconciliation rules.

Future direction: Automated reconciliation method. We propose a novel reconciliation rules generator that can fully use the semantic annotation expressions to derive most reconciliation rules automatically. Hence, the semantic annotation tool must capture non-trivial semantic correspondences including value-to-value map tables, conversion, and default values. Application experts should be involved only in the most difficult cases such as complex conversion functions.

Related work

Several different architectural models for the semantic-mediation have been discussed[10] and demonstrated.

Anicic[11] demonstrated an any-to-any model where local OWL ontologies are merged and source ontology individuals classified and transformed into target ontology individuals by automated reasoners. Artemis[12] demonstrated crosswise mappings among local OWL ontologies, by using an ontology mapping tool (http:// sourceforge.net/projects/owlmt).

However, the any-to-any model, which employs no reference ontology as the mediation point, can increase the number of crosswise mappings and the size and complexity of the merged ontology, when many local ontologies are involved. On the other hand, an architectural model that employs a reference ontology as the central mapping point - the any-to-one mapping model - reduces the number of such mappings.

Harmonise[13] demonstrated any-to-one mapping model, developed reference Tourism RDFS Ontology, and used RDF as an interchange format. Mafra tool (http://sourceforge.net/projects/mafra-toolkit) was used for mappings between RDFS models. Similar semantic-mediation architecture for supply-chain applications introduced general Supply-Chain Ontology (SCO) and used Semantic Web Rule Language (SWRL) for mappings between the SCO and local ontologies[14].

Our work leans on the Automated Methods for Integrating Systems (AMIS) project[15], which also discussed any-to-one mapping model for the semantic-mediation. We applied AMIS model to address the lack of formal and machine-understandable message semantics and inappropriate hard-coded implementation of mappings between proprietary and standard interfaces.

Besides the ATHENA tools, other tools could be used in the proposed architecture: Protege (http://protege.stanford.edu) for ontology development; Mafra, OWLmt or Snoggle (http://snoggle.projects.semwebcentral.org/) for mappings specification. Tools for message semantics annotation are missing. The SAWSDL (Semantic Annotations for Web Service Description Language and XML Schema)[16] is an emerging standard that defines set of extensional attributes by which semantic annotations can be added to web-service description and XML schemas; however, there are no tools capable to generate mapping rules from SAWSDL annotations yet. So far only ATHENA provided a toolset for both design and run-time activities to enable the proposed semantic-mediation of XML messages.

The semantic web-services have also become a key technology for semantic integration of supply chains[17]. Our work is, however, concerned with the semantic integration of supply-chain applications in traditional web-service environments.

Conclusion

We demonstrated a novel semantic-mediation architecture supporting interoperable standards-conformant message exchanges between heterogeneous applications employing proprietary message schemas.

The proposed approach moved syntactic, informal specifications of business intent to formal, semantic-based specifications. Consequently, several implementation tasks associated with standards compliance were moved to a model-based approach. As a result, implementation became more straightforward compared to the traditional one.

Although the experimental scenario was small-scale and involved three participants, it was based on a real standards-based message exchange - the BOD developed by the AIAG standards development organization. The actual BOD presented annotation and reconciliation requirements that would be found in large-scale scenarios, too. The pilot showed that most of the mappings between elements of, either actual BOD or proprietary-message interfaces, and the reference ontology concepts were simple one-to-one mappings (86% of BOD, 92% of GM, and 96% of Apolon rules, respectively). We believe that this will be true for most other real-world scenarios. Nevertheless, in any mapping case, the reconciliation specification shall not require a large effort on behalf of the annotation or rules engineer if the supporting

toolset is optimized to realistically handle industrial B2B messaging solutions (e.g. XML schemas or EDI).

While the ATHENA toolset supported our XML semantic-mediation scenario, we showed that current design-time and run-time semantic-integration tools, which use Semantic Web technologies, were not adequate. The chief suboptimal results that we observed include (1) inadequate representation method that supports low-level handling of information for both the message schemas and messages; (2) limited reuse of annotation artifacts; and (3) insufficiently automated reconciliation rules creation.

These findings are the basis for our on-going work in developing *abstract message model, abstract message model based semantic annotation,* and *automated reconciliation support* that could largely eliminate problems in handling realistically complex integration artifacts.

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

References

[1] P. Snack, "Standards-Based Interoperability: The Road Ahead", AIAG Actionline, July/August 2007, pp. 21-29

[2] J.Heflin, J. Hendler, "Semantic Interoperability on the Web", Proc. of Extreme Markup Languages, 2000, pp. 111-120

[3] S. Decker, et al., "The Semantic Web: The roles of XML and RDF", IEEE Internet Computing, 15(3), 2000, pp. 63-74

[4] ATHENA Web Site, Knowledge Support and Semantic Mediation Solutions - Deliverables D.A3.2-D.A3.5, January 2006, online at http://www.modelbased.net/aif/

[5] F. D'Antonio, et al., "Formalizing the OPAL eBusiness ontology design patterns with OWL", Proc. Int'l Conf. IESA, Springer-Verlag, 2007, pp. 345-357

[6] I. Miletic, et al., "Enabling Semantic-Mediation for Business Applications: XML-RDF, RDF-XML, and XSD-RDFS Transformation," Proc. Int'l Conf. IESA, Springer, 2007, pp. 483-494

[7] Automotive Industry Action Group Web Site, online at http://www.aiag.org/

[8] E. Barkmeyer, B. Kulvatunyou, "An Ontology for the e-Kanban Business Process", NISTIR 7404, June 2007, online at http://www.mel.nist.gov/msidlibrary/doc/NISTIR_7404.pdf

[9] J. Euzenat and P. Shvaiko, "Ontology Matching", Springer 2007. page 223

[10] G. Vetere, M. Lenzerini, "Models for semantic interoperability in service-oriented architectures", IBM Systems Journal 44(4), 2005, pp. 887-903

[11] N. Anicic, et al.,"Semantic Enterprise Application Integration Standards", Int'l Journal of Manufacturing and Technology, 10(2-3), 2007, pp. 205-226

[12] V. Bicer, et al., "Artemis Message Exchange Framework: Semantic Interoperability of Exchanged Messages in the Healthcare Domain", SIGMOD Record 34(3), 2005, pp. 71-76

[13] M. Dell'Erbaa, et al., "HARMONISE: A Solution for Data Interoperability", Proc. of IFIP I3E Conf., 2002, pp. 114-127

[14] Y. Ye, et al.,"An ontology-based architecture for implementing semantic integration of supply-chain management", Int'l Journal of Computer Integrated Manufacturing, 21(1), 2007, pp. 1-18

[15] D. Libes, et al., "The AMIS Approach to Systems Integration", NISTIR 7101, May 2004, online at www.mel.nist.gov/msidlibrary/doc/nistir7101.pdf

[16] J. Kopecký, et al., "SAWSDL: Semantic Annotations for WSDL and XML Schema", IEEE Internet Computing, 11(6), 2007, pp. 60-67

[17] S.A. McIlraith, et al.,"Semantic Web services", IEEE Intelligent Systems, 16(2), 2001, pp. 46-53