Semantic Mediation for Standard-Based B2B Interoperability

A semantic-mediation architecture advances traditional approaches for standard-based business-to-business interoperability. The architecture formally models a business domain in a reference ontology and annotates domain message schemas to define public and proprietary reconciliation rule sets. Enterprises can use the rule sets to implement standard-based message interfaces and to translate message content between their proprietary message forms. An implementation of the semantic-mediation architecture augments a general applications-integration toolset developed for the Athena European FP 6 project. The implementation demonstrates the architecture’s feasibility and suggests directions for future tool enhancements.

Marko Vujasinovic, Nenad Ivezic, Boonserm Kulvatunyou, and Edward Barkmeyer
US National Institute of Standards and Technology

Michele Missikoff and Francesco Taglino
Consiglio Nazionale delle Ricerche

Zoran Marjanovic and Igor Miletic
University of Belgrade
specifying business domain concepts informally (by using syntactic notations to convey data exchange requirements and their business meanings) leads to ambiguity and misinterpretation. Second, informally annotating the meaning of the standard or proprietary-message elements in nonmachine-processable form leads to misinterpreted message semantics and application integration problems. Third, manual and hard-coded mappings between proprietary and standard message elements lead to error-prone and labor-intensive implementations. Finally, high interdependence between proprietary and standard message interfaces at many levels — execution platform, technology, terminology, message standard, and message syntax — implies inflexible, tightly coupled integrations.

To address these shortcomings, we propose a novel semantic-mediation architecture for standard-based interoperability. To demonstrate its feasibility, we describe a small-scale industrial message-exchange pilot implementation.

**Semantic-Mediation Architecture**

Our proposed semantic-mediation architecture builds on the traditional approach but introduces new activities at each level, which we rename as follows:

- **Business domain ontology modeling.** An SDO captures the data exchange’s intended meaning and creates a reference ontology (the “create” activity in Figure 1) based on the business process model and data exchange requirements. The reference ontology specifies, formalizes, and explicates domain business concepts and their relationships; it’s publicly available to application providers.

- **Design-time semantic-mediation specification.** Following BOD adoption, the SDO annotates the BOD semantics (step B₁ in Figure 1) by relating each BOD element to the corresponding reference-ontology concept. The SDO then uses the BOD annotations to define publicly available reconciliation rule sets (step C₁ in Figure 1) for transforming a BOD-conformant message to a reference-ontology instance, and vice versa. Application providers annotate the semantics of their proprietary-message interfaces (typically XML schemas; step B₂ in Figure 1) by using their respective interface annotations to define proprietary reconciliation rule sets (step C₂ in Figure 1). Effectively, the proprietary and public rule sets define transformations between proprietary messages and BOD-conformant messages via the reference ontology.

- **Runtime semantic-mediation execution.** When an application sends a message, the semantic mediator executes the appropriate reconciliation rule sets, first to translate the proprietary message to the reference-ontology instances and then to translate the reference-ontology instances to the BOD-conformant message. When an application receives a message, 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.

Our proposed architecture addresses the traditional approach’s shortcomings through several advances. First, the reference ontology formally specifies business domain concepts, providing a basis for unambiguous interpretation of data exchange artifacts. Second, message schemas are annotated with the reference ontology, providing machine-processable expressions that formally describe message elements’ meaning; such annotation also enables precise specification of semantic reconciliation rules. The automated and consistent implementation of a standard-based interface — enabled by executing the reconciliation rules — moves the engineering effort from implementation to modeling and design time. Finally, the architecture reconciles terminological, structural, semantic, and representational differences between message specifications.

Ultimately, our semantic-mediation architecture achieves standard-based interoperability in three steps: reference-ontology development, semantic annotation of message elements, and definition of reconciliation rules. The total effort is distributed between application providers and the SDO. Initially, the SDO creates the ontology, annotates the BODs, and defines reconciliation for BOD-conformant messages; it must perform these tasks only once. To implement an application’s standard-conformant message interface, each provider annotates its proprietary-message interface using the reference ontology, defines reconciliation for its proprietary messages, and
deployed the semantic mediator. The providers must do this once for each proprietary interface to a specific reference ontology. Obviously, the scale of effort will depend on the complexity of the proprietary messages and the ontology. (The SDO and providers transform their respective standard and proprietary-message schemas into tool-specific forms when annotation or reconciliation tools require it—steps A₁ and A₂ in Figure 1.)

**Supporting Tools**

We implemented our proposed architecture using the Athena (Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Applications) tool suite that was developed as part of an FP 6 European Integrated Project. The Athena tools support semantics-based reconciliation of Resource Description Framework (RDF) documents (see Figure 1).

At design time, our architecture uses three Athena tools:

- **Athos ontology-development tool** (http://leks-pub.iasi.cnr.it/Athos). Athos relies on the Object-Model-Actor Language (Opal)⁵ to construct ontologies through predefined business categories and inherent constraints. Opal is built on top of the Web Ontology Language (OWL), which gives a formal basis to Athos-developed ontologies.
- **Astar semantic-annotation tool** (http://leks-pub.iasi.cnr.it/Astar). Astar lets us set up semantic correspondences (that is, semantic annotation expressions) between
RDF Schema (RDFS) model concepts and reference-ontology concepts. It provides a graphical annotation environment, RDFS model visualization, and semiautomatic support for defining annotation expressions.

- **Argos reconciliation-specification tool** ([http://leks-pub.iasi.cnr.it/Sinergia_Athos](http://leks-pub.iasi.cnr.it/Sinergia_Athos)). Argos provides a graphical environment for specifying reconciliation rules. It visualizes RDFS models and the reference ontology and assists in creating forward and backward reconciliation rule sets that transform RDF documents, respectively, to and from a reference-ontology instance. Astar annotation expressions drive the reconciliation rules creation in Argos.

At runtime, our proposed architecture supports Web services (WS) messaging through the Ares reconciliation execution engine and the Johnson WS execution engine. Ares performs the actual RDF-to-RDF document reconciliation by executing declared forward and backward rule sets on RDF documents.

The Athena tool requirements for supporting general applications integration were similar to our B2B integration requirements. However, specific characteristics of the B2B integration revealed a need for additional support. First, to enable semantic mediation of XML messages, we built

- XSD2RDFS, a design-time tool that transforms message-element XML schema definitions (XSD) to the corresponding conceptual RDFS model (that is, to a conceptual message model), and
- XML2RDF and RDF2XML, runtime tools that transform XML messages to and from the corresponding RDF documents.

Second, to integrate the runtime tools, we developed the **coordinator gateway**, which orchestrates all runtime support for transforming a proprietary message into a standard-conformant message.

**eKanban Experimental Pilot Implementation**

To assess our architecture’s representational capabilities, we executed an experimental pilot project. We based the pilot implementation on the electronic Kanban (eKanban) inventory visibility and interoperability (IV&I) business process that AIAG developed to regulate the flow of goods from suppliers to match actual customer usage. AIAG defined a standard BOD message set for the eKanban process, which we used in our project.

We employed two independently developed applications that could send and receive only their proprietary AuthorizeKanban messages and one standard-conformant application that could receive the standard BOD AuthorizeKanban message:

- The Apolon open source IV&I application with an RDFS-based proprietary-message interface.
- The 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), and the GM application (running in Michigan) exchanges a message with the FTH and Apolon.

First, we used the Athos tool to develop the eKanban Reference Ontology, which formally captured the business conceptual model for the eKanban process. Next, we performed the design-time steps. We completed the XSD2RDFS transformation of the BOD and GM AuthorizeKanban XML schemas to the corresponding RDFS conceptual message models (steps A1 and B1 in Figure 1), as required by Athena tooling. Then we used Astar to annotate the BOD, GM, and Apolon AuthorizeKanban RDFS conceptual message models (steps A2 and B2 in Figure 1). We used Argos to complete the reconciliation specification (steps C1 and C2 in Figure 1), creating forward rules to transform data from the GM, Apolon, and BOD AuthorizeKanban RDF documents into the reference-ontology instances, and backward rules into transform data from reference-ontology instances to Apolon and BOD AuthorizeKanban RDF documents.

At runtime, the coordinator gateway orchestrated a sequence of transformations and reconciliations, as shown in Figure 2. Each application had its own appropriately configured coordinator gateway. The semantic mediation was successful — that is, applications sent
Interoperability

and received messages in their proprietary formats and, moreover, conformed to the adopted exchange standard.

We give details for the integration steps from a provider’s perspective, along with example annotation expressions, reconciliation rules, and implementation details elsewhere. Here we concentrate on key architectural and functional challenges in the pilot and recommendations for future tool enhancements.

Key Implementation Findings

Three functions are central to successful use of our proposed architecture in realistically complex B2B integration cases: message-representation transformation, message-semantics annotation, and message-reconciliation specification.

Message-Representation Transformation

The semantic-mediation tools require transformation of a message schema or message instance — typically to a form that’s aligned with the ontology representation language. A key challenge in the message-representation transformation is to develop a general and flexible solution that abstracts from the unnecessary message syntax details while maintaining the essential schema or instance information.

Approach. We designed the XSD2RDFS, XML2RDF, and RDF2XML message-representation transformations to meet the Astar and Argos tool requirements — specifically, requirements for a message-schema abstraction in the form of an RDFS conceptual message model and for messages in the form of RDF documents.

To transform an XML message schema to an RDFS conceptual message model, XSD2RDFS builds an internal tree that reflects the XML message structure. Tree nodes represent XML element and attribute definitions, and each node encapsulates the name, data type, and namespace for the corresponding element or attribute. The tool then transforms the tree into an RDFS conceptual message model through predefined transformation rules and the “extended names” naming convention. Figure 3 shows sample XSD2RDFS transformation. It transforms xsd:elements into corresponding RDFS classes — for example, gmSyncShipmentSchedule is transformed to gmSyncShipmentSchedule_sender. Also, each simple xsd:element is transformed into a corresponding RDFS data property — for example, the DUNS element is transformed into gmSyncShipmentSchedule_sender_DUNS_sValue. Each XML parent-child relation is transformed into an RDFS object...

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message (relative to RO)</th>
<th>Input</th>
<th>Message</th>
<th>Reconciliation</th>
<th>Reconciled message in ontology form</th>
<th>Reconciliation</th>
<th>Message</th>
<th>Output</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM sends to Apolon</td>
<td>XML GM</td>
<td>XML2RDF GM</td>
<td>RDF GM</td>
<td>Forward RO</td>
<td>RO Backward RDF BOD</td>
<td>RDF RDF2XML Standard XML BOD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apolon sends to FTH</td>
<td>RDF not needed</td>
<td>BOD Forward RO</td>
<td>RDF BOD</td>
<td>RDF RDF2XML</td>
<td>Standard RDF Apolon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM sends to FTH</td>
<td>XML GM</td>
<td>XML2RDF GM</td>
<td>RDF GM</td>
<td>Forward RO</td>
<td>RO BOD_</td>
<td>RDF RDF2XML Standard XML BOD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2. Message flow and executed transformation inside the coordinator gateway. 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.
property — the sender-DUNS parent-child XML relation becomes the gmSyncShipmentSchedule(sender)_sender_DUNS_PROP RDFS property.

Even though specific Astar and Argos requirements drive XSD2RDFS’s functionality, the tool is general. It can transform any given XML schema into a corresponding RDFS conceptual message model.

For the runtime message-representation transformations, we defined the XML2RDF and RDF2XML transformation-naming convention.\(^7\) The transformation algorithms use this convention to create RDFS-conformant RDF documents for the reconciliation input and XML schema-conformant XML documents from the reconciliation output. This approach makes the RDF2XML and XML2RDF transformations flexible enough to eliminate the need for runtime presence of the corresponding XML schema and RDFS conceptual message model. Figure 4 shows sample XML2RDF and RDF2XML transformations.

Findings. Our pilot implementation uncovered the need to preserve message-structure and message-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-conformant message from reconciliation output, you must maintain definitions for at least the following information:

- message structure,
- message concept granularity (element versus attribute),
- message concept names including namespaces,
- message concepts’ order, and
- formatting rules.

A conceptual message model doesn’t capture this information. However, we were able to maintain it, first, by embedding the structural and granularity characteristics in naming conventions. For example, the extended-name convention for concepts maintained the message-structure definition and the elements versus attributes distinction with the _ATTR suffix. Second, we created reconciliation rules to gen-
Interoperability

Figure 4. Portion of the XML2RDF and RDF2XML transformation output for the General Motors message. XML2RDF transforms its output (GM’s XML message on the left) to the RDF document that contains GM’s message data on the right. RDF2XML makes the opposite transformation.

erate additional target-RDF-document statements to carry the XML-message-representation rules through the entire semantic mediation. For example, additional RDF statements carried the namespace and element order definitions.

Creating 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 (surveyed elsewhere) can transform message schemas to a form that sufficiently captures the five minimum schema definitions listed earlier while aligning with an ontology representation language, such as OWL or RDFS. (Also, see the sidebar for more information on related work in semantic mediation.)

Future direction: abstract message model. To rectify this behavior and, furthermore, to mediate messages other than those in XML syntax, such as messages in Electronic Data Interchange (EDI) syntax, we believe that a semantic mediation needs an abstract message model that, when instantiated, represents a syntax-neutral model of both the schema and the message of some concrete syntax.

An abstract message model doesn’t capture certain syntax-specific constructs of message schemas or messages. However, it faithfully captures message-schema concepts, such as element and attribute definitions, complex versus simple type definitions, and the mandatory message schema information we listed under “Findings.” It also captures message concepts such as elements, attributes, content, and values. These capabilities will allow the message-representation transformation to instantiate and populate the abstract message model with information from an actual message schema or message. This approach further implies that the forward reconciliation must come from — and the backward reconciliation go to — the abstract message model instance (for example, GM and BOD abstract message model instances). The abstract message model instance, as an output from a reconciliation engine, will then contain enough information for the specific message-representation transformation to produce the schema-conformant message from it.

Message-Semantics Annotation
To clarify message semantics, the message-semantics annotation activity associates each message element definition with a machine-processable 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 multipurpose usability of semantic annotation expressions.

Approach. The Astar annotation method is organized in two phases:
- The diagnostic phase identifies mismatches between a conceptual message model and
Related Work in Semantic Mediation

Researchers have explored and demonstrated several different architectural models for semantic mediation.¹

Nenad Anicic and his colleagues demonstrated an any-to-any mapping model, in which local OWL ontologies (that is, OWL conceptual message models) are merged and source ontology individuals classified and transformed into target ontology individuals by automated reasoners.² The Artemis message-exchange framework uses the OWLmt ontology mapping tool (http://sourceforge.net/projects/owlmt) to demonstrate crosswise mappings among local OWL ontologies.³

The any-to-any model employs no reference ontology as the mediation point. For this reason, when the semantic mediation involves many local ontologies, this model increases the number of crosswise mappings or the merged ontology’s size and complexity. 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 mappings.

The European Harmonise project demonstrated an any-to-one mapping model, developed a reference Tourism RDFs Ontology, and used RDF as an interchange format.⁴ Harmonise used the Mafra toolkit (http://sourceforge.net/projects/mafra -toolkit) for mappings between RDFs ontologies. Yu Ye and his colleagues at Shanghai Jiao Tong University developed a similar semantic-mediation architecture for supply-chain applications.⁵ Their architecture introduced a general supply-chain ontology (SCO) and used the Semantic Web Rule Language for mappings between the SCO and local ontologies.⁶

Our work leans on the Automated Methods for Integrated Systems (AMIS) project, which also discussed an any-to-one mapping model for semantic mediation.⁷ We applied the AMIS model to address the lack of formal and machine-processable message semantics in schemas and to correct time-consuming and error-prone practice of hard-coding mappings between proprietary and standard interfaces.

Besides the Athena toolset, other tools that application integrators could use in our proposed architecture include Protege (http://protege.stanford.edu) for ontology development and Mafra, OWLmt, or Snuggle (http://snuggle.projects. semwebcentral.org) for specifying mappings. Tools for message semantics annotation are missing. SAWSDL (Semantic Annotations for WSDL and XML Schema) is an emerging standard that defines a set of extensional attributes by which semantic annotations can be added to Web service description and XML schemas.⁸ However, no tools can yet generate mapping rules from SAWSDL annotations. So far, only Athena provides a design and runtime toolset to enable the proposed semantic mediation of XML messages.

Semantic Web services have also become a key technology for semantic integration of supply chains.⁹ Our work, however, is concerned with the semantic integration of supply-chain applications in traditional Web service environments.

References


The remedial phase has four steps: terminological semantic annotation (TSA), path semantic annotation (PSA), simple semantic annotation (SSA), and full semantic annotation (FSA):

- **TSA step.** Astar contrasts the terms of the conceptual message-model concepts with the reference-ontology terminology and automatically detects lexical-terminological similarities among them. The user then manually establishes correspondences between these terms, which resolves mismatches and assists the identification of structural path matches in the PSA.
• **PSA step.** To resolve structural mismatches, the user considers the conceptual message-model and reference-ontology structures and associates one or more conceptual message-model structural paths to one or more matching reference-ontology structural paths.

• **SSA step.** The user composes all the path matches into path expressions by using abstract operators, which denote a data transformation template needed at runtime. The abstract operators can also note other semantic mismatches in this step, such as the different data encoding and data representation choices.

• **FSA step.** Path expressions are translated into OWL DL (description logics) semantic annotation expressions. OWL allows encoding of the actual relationships between the semantic concepts — for example, equivalence, subsumption, and overlap.

Table 1 shows an example of the mismatches and the four-step annotation for a portion of the GM conceptual message model.

**Findings.** Path matching required significant human effort because Astar a priori generated and showed users all path combinations through the reference ontology graph. This led to overwhelming complexity in the annotation activity. A semantic-annotation tool should let users steer the path development rather than presenting a long list of all possible paths. Nevertheless, we used Astar successfully to annotate all the conceptual message models.

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 nonexecutability. Annotating message semantics needs an interchangeable, executable, expressive, but simple expression representation format.

Significantly, Astar doesn’t provide for semantics annotation of the actual (XML/EDI) message-schema components and message elements — only annotation of conceptual message-model concepts and their properties. This directly affects the annotation usability in the reconciliation that generates message-schema-conformant messages. A message-representation transformation tool can’t engineer important information about message element definitions, message structure, and data representation from conceptual message models. It also rules out the possible reuse of such annotations for other purposes, such as semantic querying over XML/EDI messages and discovery and reuse of schema components.

**Future direction:** semantics annotation through abstract message models. We need a novel annotation method that works with abstract message model instances (conformant with actual message schemas) rather than conceptual message models. Annotations based on the abstract message model instances should improve our architecture’s reconciliation runtime capabilities and multipurpose usability of the annotation expressions. Furthermore, the annotation method must support reuse of annotation expressions on at least three levels.

The first is message component reuse. For example, annotation of context-specific aggregated message components, such as *SenderId* and *ReceiverParty* (in Open Application Group BODs; www.oagi.org), requires the capability to reuse annotation expressions of the message components’ base-element type definitions (such as the *PartyType* base type) or of their base context-free aggregated elements’ definition (such as *Party*) along with the containing elements.

The second level is reuse across different but overlapping message types. The third is reuse of an entire annotation set for a message set.

**Message-Reconciliation Specification**

To specify message reconciliation, we must define the forward- or backward-executable message content transformation rules. Forward rules describe how to transform the content of one or more message elements into instances of reference-ontology concepts that match the content. Backward rules describe how to transform the content of one or more instances of the reference-ontology concepts to message elements that match the content.

Several transformation patterns might exist, such as one-to-one, many-to-one, one-to-many, or more complex patterns including conversion functions. The significant challenge is to automate rule generation.

**Approach.** Argos provides semiautomatic speci-
fication of reconciliation rules based on semantic annotation expressions. A user manually selects and instantiates an appropriate predefined rule template for one or more conceptual message-model paths leading to the content to be transformed or populated. Argos rule templates include one-to-one, one-to-many, and many-to-one maps, as well as sum, set value, and conversion functions. The tool then creates a declarative runtime rule by substituting the conceptual message-model path (or paths) and the matching reference-ontology path (or paths) into the template, based on the annotation expressions.

Argos uses Jena as an executable rule representation language (http://jena.sourceforge.net). Figure 5 shows a Jena rule that specifies a one-to-one mapping between GM’s Sender.DUNS and the reference ontology’s senderParty.PartyId identifier concepts.

**Table 1. Portion of General Motor’s RDFS conceptual message-model annotation.**

<p>| Terminological semantic annotation (TSA) |  |  |</p>
<table>
<thead>
<tr>
<th>GM’s RDFS concept</th>
<th>Mismatch</th>
<th>Reference-ontology concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmSyncShipmentSchedule</td>
<td></td>
<td>ShipmentSchedule, SyncShipmentSchedule_Message</td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_sender</td>
<td></td>
<td>SenderParty</td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_sender_DUNS</td>
<td></td>
<td>PartyId</td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_senderPROP</td>
<td></td>
<td>relTo_SyncShipmentSchedule_Message_SenderParty</td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_sender_DUNSPROP</td>
<td></td>
<td>relTo_SyncShipmentSchedule_Message_SenderParty_PartyId</td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_sender_DUNS_sValue</td>
<td></td>
<td>has_identifier</td>
</tr>
</tbody>
</table>

<p>| Path semantic annotation (PSA) |  |  |</p>
<table>
<thead>
<tr>
<th>GM’s RDFS path</th>
<th>Mismatch</th>
<th>Reference-ontology path</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmSyncShipmentSchedule. gmSyncShipmentSchedule_sender_PROP</td>
<td></td>
<td>structuring</td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_sender. gmSyncShipmentSchedule_sender_DUNSPROP. gmSyncShipmentSchedule_sender_DUNS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_sender_DUNS_sValue:STRING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_message_SyncShipmentSchedule</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_Message_SenderParty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_Message_SenderParty_PartyId</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule.Message_SenderParty.PartyId.has_identified:STRING</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| Simple semantic annotation (SSA) |  |  |</p>
<table>
<thead>
<tr>
<th>GM’s RDFS path</th>
<th>Abstract operator</th>
<th>Reference-ontology path</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmSyncShipmentSchedule. gmSyncShipmentSchedule_sender_PROP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_sender. gmSyncShipmentSchedule_sender_DUNSPROP. gmSyncShipmentSchedule_sender_DUNS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_sender_DUNS_sValue:STRING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_message_SyncShipmentSchedule</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_Message_SenderParty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule_Message_SenderParty_PartyId</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmSyncShipmentSchedule.Message_SenderParty_PartyId.has_identified:STRING</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| Full semantic annotation (FSA) - Web Ontology Language Description Logics (OWL DL) semantic annotation expressions |  |  |</p>
<table>
<thead>
<tr>
<th>GM’s RDFS path in DL</th>
<th>OWL axiom</th>
<th>Reference-ontology path in DL</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING ∨ (inverseOf_gmSyncShipmentSchedule_sender_DUNS(gmSyncShipmentSchedule_sender_DUNS ∨ (inverseOf_gmSyncShipmentSchedule_sender_PROP(gmSyncShipmentSchedule_sender_DUNSPROP) ∨ (inverseOf_gmSyncShipmentSchedule_senderPROP(gmSyncShipmentScheduleSender PROP(gmSyncShipmentSchedule))))))</td>
<td>owl:equivalentClass</td>
<td>STRING ∨ (inverseOf_has_Id_identified(PartyId ∨ (inverseOf_reTo_SyncShipmentSchedule_Message_SenderParty_PartyId(SenderParty ∨ (inverseOf_reTo_SyncShipmentSchedule_Message_SenderParty(SyncShipmentSchedule_Message) ∨ (inverseOf_reTo_ShipmentSchedule_message_SyncShipmentSchedule(ShipmentSchedule)))))))</td>
</tr>
</tbody>
</table>
### Findings

Although Argos is a semiautomatic tool, we instantiated all rules manually. Argos could have automated this process if it had fully used annotation expressions and abstract operators as mapping directives. In fact, the instantiations of the one-to-one map template could be completely automated, which would have reduced the time needed for reconciliation specification because 85 percent of the pilot rules were one-to-one maps.

Despite unnecessary manual effort, we used Argos successfully to create all the required reconciliation rules.

### Future approach: automated reconciliation

We need a reconciliation rules generator that fully uses the semantic annotation expressions to derive most reconciliation rules automatically. This means the semantic annotation tool must capture nontrivial semantic correspondences, such as value-to-value map tables, conversions, and default values. Special cases, such as complex conversion functions, might still require an application expert, but they will come up only a fraction of the time.

Our approach moves syntactic, informal specification of business intent to formal, semantic-based specification. Several implementation tasks associated with standards compliance therefore move to a model-based approach, which makes actual implementation much more straightforward.

The experimental scenario we report here was small-scale, involving three participants, but it involved a real standard-based message exchange. Thus, the AuthorizeKanban BOD annotation and reconciliation requirements we present would appear in large-scale scenarios, too. Our pilot showed that most mappings between elements of either actual BOD or proprietary-message interfaces and reference-ontology concepts were one-to-one (86, 92, and 96 percent, respectively, of BOD, GM, and Apollo rules). One-to-one maps are simple and easy to establish, and would represent most business participants’ mappings to the reference ontology in a real industrial case. Nevertheless, in any mapping case, the reconciliation specification shouldn’t require a large annotation or rule engineering effort if the supporting toolset is optimized to realistically handle industrial B2B messaging solutions, such as XML schemas or EDI.

The Athena toolset supported our pilot scenario, but the current Semantic Web technologies for semantic data management need new tools that are engineered for realistic integration requirements and can take advantage of these technologies. Our pilot study findings are the basis for ongoing work in developing an abstract message model, semantic annotation based on that model, automated reconciliation support, and new supporting tooling. These tools could largely eliminate problems in handling realistically complex integration artifacts.

### Disclaimer

This article identifies certain commercial and open source software products. We used these products only for demonstration purposes. This use doesn't imply approval or endorsement by the US National Institute of Standards and Technology, nor does it imply that these products are necessarily the best available for the purpose.

### References


**Marko Vujasinovic** is a PhD candidate in the University of Belgrade’s Department of Information Systems, Faculty of Organizational Sciences (FOS). He’s also a guest researcher at the US National Institute of Standards and Technology’s Manufacturing Systems Integration Division. His research interests include business-to-business systems integration, semantic interoperability, and Semantic Web technologies. Vujasinovic has an MSc in information systems from FOS. Contact him at marko.vujasinovic@nist.gov.

**Nenad Ivezic** is a research staff member of the Enterprise Systems Group at the US National Institute of Standards and Technology’s Manufacturing Engineering Laboratory. His research interests include IS development methodologies, databases, semantic enterprise application interoperability, and information systems development. Ivezic has a PhD in computer-aided engineering from Carnegie Mellon University. Contact him at nenad.ivezic@nist.gov.

**Boonsorn Kulvatunyou** is a standards and product architect in Oracle’s Adaptive Business Solutions division. His research interests are enterprise data models for interoperable service-oriented architecture. Kulvatunyou has a PhD in industrial engineering from Pennsylvania State University. Contact him at kbserm@psualum.com.

**Edward Barkmeyer** is a senior computer scientist at the US National Institute of Standards and Technology’s Manufacturing Engineering Laboratory. His research interests include the analysis and architecture of information interchange among manufacturing software systems for engineering, planning, control, and supply-chain operations. Barkmeyer has an MS in applied mathematics from the University of Maryland. Contact him at edbark@nist.gov.

**Michele Missikoff** is a professor at the University of Rome La Sapienza and research director of the Italian National Research Council, where he manages the Lab for Enterprise Knowledge and Systems. He’s also the scientific director of the European Interop-VLab. His research interests are in knowledge bases, knowledge representation, and semantic technologies with a focus on ontologies for interoperability in e-government and e-business. Missikoff has a Laurea in physics from University of Rome La Sapienza. Contact him at missikoff@iasi.cnr.it.

**Francesco Taglino** is a researcher at the Italian National Research Council’s Lab for Enterprise Knowledge and Systems. His research interests include semantic interoperability and ontology management systems. Taglino graduated information science from the University of Rome La Sapienza. Contact him at francesco.taglino@iasi.cnr.it.

**Zoran Marjanovic** is a full professor in the University of Belgrade’s Department of Information Systems, Faculty of Organizational Sciences (FOS), and a president of Breza Software Engineering. His research interests are IS development methodologies, databases, semantic enterprise application interoperability, and information systems development. Marjanovic has a PhD in information systems from the University of Belgrade. He’s a member of the ACM and the IEEE. Contact him at zoran.marjanovic@fon.rs.

**Igor Miletic** is a postgraduate student in the University of Belgrade’s Department of Information Systems, Faculty of Organizational Sciences (FOS). His research interests are the development and integration of business applications systems, Semantic Web technologies, and applying Semantic Web concepts to business applications. Contact him at igor.miletic@brezasoftware.com.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.