Use of Semantic Mediation in Manufacturing Supply Chains

Peter Denno, Edward J. Barkmeyer, Fabian Neuhaus
National Institute of Standards and Technology

Abstract: This case discusses lessons learned about enabling interoperability using semantic methods in three automotive industry projects spanning 8 years. In each project, the essential form of the solution is to reconcile differences in viewpoint of entities exposed at component interfaces.

Keywords: interoperability, supply chains, semantic mediation

Introduction

This case study reports on our experience investigating issues of semantic interoperability through three related research projects. The common thread throughout our work is the central role that system engineering processes and their formalization and automation play in facilitating semantic interoperability. *Systems engineering* is any methodical approach to the synthesis of an entire system that (1) defines views of that system that help elicit and elaborate requirements, and (2) manages the relationship of requirements to performance measures, constraints, risk, components, and discipline-specific system views. The projects described in this case study automate tasks traditionally performed by human systems engineers. Our perspective is that today's service-oriented architectures, tomorrow's semantic web, and other efforts in interoperability seek to do the same. In these efforts, the task to be automated determines the knowledge that needs to be formally represented. Qualities of that representation and the soundness of the process that automates the task will determine the characteristics of the solutions produced. Among these characteristics, confidence in the assessments of risk, and performance of the solution produced, stand out as key issues in determining the solution's effectiveness in a given problem space. The search for a new paradigm in systems integration is a search for an approach that provides a good return on the cost of producing the formal representation that enables the automation.

The three projects discussed in this case study focus on providing semantic interoperability by reconciling differences of viewpoint that may be present in the system components whose joint work provides a system function. Reconciling differences in viewpoint that are exposed in component interfaces is a systems engineering task. Though the three projects differ with respect to the nature of the solutions that they deliver, they share important aspects of design.[1]

Background

System components *interoperate* when they act jointly for the purpose of achieving a shared goal. Their joint work is coordinated through their communication with each other. This is as true for wholly mechanical systems as it is for information systems and agencies with human components. With respect to information systems particularly, the communication is a message, which is the bearer of information that is necessary for the component to fulfill its role in the system.

The requirements conceived for a component are not necessarily a subset of the system requirements. The component may have been designed for use with another system, or without foreknowledge of the systems with which it might interact. What is necessary is only that when provided with an appropriate message, the component must respond with a behavior that serves the intended system requirement. Whereas system interoperability is the ability to do the joint work generally, *semantic interoperability* concerns what is meant by "intended" and "appropriate" here. (See *Illustration 1*.) Semantic interoperability concerns the relationship between the intended immediate goal of the sender and its understanding of how it might elicit the response it requires from the component. The message is tailored by the sender based on its immediate goal and its understanding of this relationship.

Semantic interoperability then, is not a form of interoperability, but rather a term that brings attention to a failure mode of interoperability – one in which intended consequences are not achieved due to a misinterpretation of a message/behavior relationship.

[1] References to proprietary products are included in this paper solely to identify the tools actually used in the industrial applications. This identification does not imply any recommendation or endorsement by NIST as to the actual suitability of the product for the purpose.

The trichotomy of syntax, semantics, and pragmatics can be introduced in different ways. One that is widely accepted identifies the semantics of a sentence with its truth-conditions, which can be formally specified with the help of a model theory. Hence, the meaning of a sentence (or an axiomatic theory) is identified with the set of models that satisfy the sentence (axiomatic theory). This is the notion of "semantics" which is fundamental to the approach presented by Michael Gruninger in another chapter of this book. He embraces the Ontological Stance and models software applications as if they were axiomatic theories; thus the notion of semantic interoperability between two software applications is analyzed in his view in terms of constraints on satisfying models.

In contrast, we do not link semantic interoperability to models and truth-conditions, but to behavior or, to be more specific, lack of intended behavior: the absence of semantic interoperability between system components has been defined above as the inability to achieve some joint action of the systems components which is the result of the inability of a component to respond with the intended behavior if provided by the appropriate message. Within linguistics, the study of how speaker intend to bring about reactions of their listeners is part of pragmatics, hence one could argue that we are not concerned with "semantic interoperability" but "pragmatic interoperability." However, we decided to follow the liberal use of the term "semantics" that seems to be prevalent in the literature

A purely semantic view of communication would relate structures (message structures) to structures (implemented in the component). The view depicted in *Illustration 1* relates message structures to behaviors, and therefore admits a broader set of concerns and is more directly relevant to the goal of system integration. Relating message to behaviors enables recognition of quality of service issues, for example, that the response is timely and accurate to a degree defined by a requirement. Indeed, quality of service requirements may be included in what is communicated to the message recipient. Secondly, message/behavior relationships are a special case of stimulus/response relationships, the building blocks of every engineering endeavor. Aspects of behaviors recognized as sufficing particular system requirements are harnessed through stimulus/response (or message/behavior) relationships to serve system goals. It is the recognition of the sufficiency, inconsequentiality, and antagonism of behaviors that motivates the engineer's choice of design.

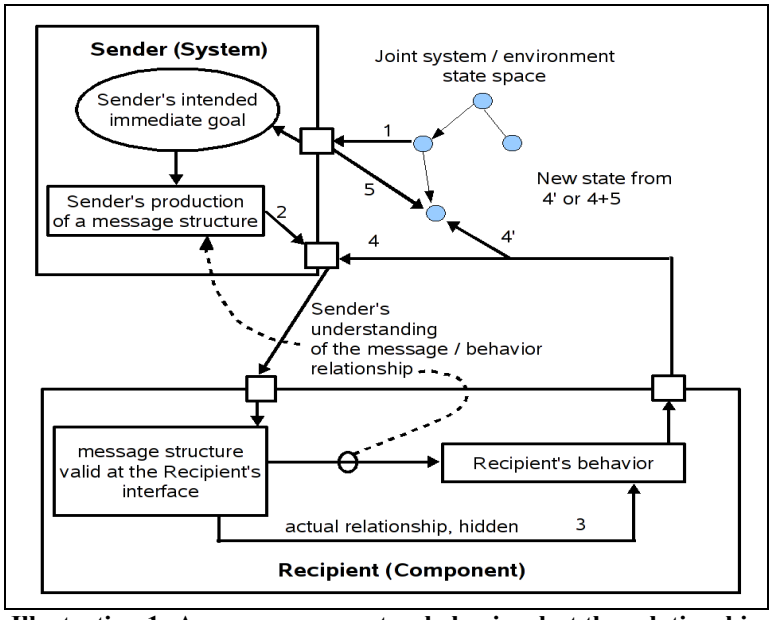

Illustration 1: A message prompts a behavior, but the relationship between the message and behavior is hidden from the sender.

Messages are intended to elicit behaviors. As important as is understanding what must be included in the message is understanding what need not. To someone unfamiliar with the workings of a system, the content of its communications may appear starkly insufficient for the purpose of coordinating joint work. For example, an information system supporting a manufacturing supply chain may issue an order fulfillment request message containing little more than an order reference number, industrial code terms identifying the parties involved, and a list of part numbers and their quantities. The message might omit, for example, details describing the transport of the goods. This is possible because messages are not the sole source of information enabling the system to function. Messages serve to trigger a system function and to identify the individuals that are to participate in an

occurrence of a process. But the pattern of activity that is the process, and the properties associated with the types of the individuals can be encoded into system components and elsewhere.

To the reader familiar with Service Oriented Architecture, (SOA) a casual reading of the above paragraph, its mention of stark message content and static process definitions, may appear antagonistic to SOA's goals – that it entails opaque and idiosyncratic interfaces that are costly to establish and to evolve as requirements change. This is not the case. SOA and methods that seek to mechanize semantic interoperability operate by exposing information about supported services and protocols to an infrastructure that can use it to dynamically build system solutions. In the problem space that SOA addresses, this information is sufficient. For other classes of problems, other systems integration knowledge may be more pertinent. For example, there may be a need to consider the "risk profile" for the composition of services (a plan) performing some high-level task. Knowledge of the risk of the plan, assessed in its social context, would determine the system solution's viability. In other classes of problems, ontology matching and meditation, or traceability and accountability to high-level goals might be critical. What SOA and these other examples share is that they select among the totality of systems engineering concerns those that are to be mechanized in their infrastructure. Characteristics such as stark message content and static process definitions are entailments of some selection, and might be completely appropriate for the problem at hand. In those cases, concerns not mechanized in the run-time infrastructure can be addressed outside it, using semi-automated tools, to perform systems integration. Business process modeling, when used by a human system integrator, is an example of this.

To automate tasks of systems integration, it is necessary to know what these tasks are. Unfortunately, what we might call the "systems integration body of knowledge" is relatively unexplored terrain. It encompasses some of the software engineering body of knowledge (SWEBOK) [1] and some of the systems engineering body of knowledge (SEBOK) [2] and some elements not part of either of these.

Integration and Mediation

An integration problem can be stated as a requirement to provide an improved business result from a set of existing systems. The existing systems may be viewed as components of a new system that will provide new functions. (We use the term *component systems* to refer to these throughout the case study.) Each component system contributes through interfaces it exposes to the new system. These interfaces are communication endpoints. Integration is the activity of enabling communication among the components so that they may coordinate their behaviors to perform *joint actions* that achieve the improved business result.

To enable the desired communication between the endpoints where differences in message syntax or interpretation exist, the engineer can either (1) generate the appropriate translators of the relevant messages between the two tools using the interfaces they already support, or (2) adapt existing interfaces or provide new interfaces to support the required communications directly. The former is analogous to the mediator design pattern of object-oriented programming [3] and the latter to its adapter design pattern. However, our focus is on the general system organization for communication, and not object-oriented programming.

In practice, the choice of a mediator design versus an adapter design may be based on pragmatic logistical concerns of deployment in the environment in which the system will operate. If the components that must interoperate are numerous and loosely federated, as they are in, for example, supply chain logistics, an adapter design using agreed-upon messages may be preferred. On the other hand, if the components that must interoperate are few, there is little opportunity for modification of the interface, nor a desire to collaborate, a mediator design may be more appropriate.

The first project described in this case study uses a mediator design, the second an adapter design based on a mediator architecture, and the third an adapter design. Nonetheless, the process by which the integration is achieved, and the information on which the mediation relies, remains substantially the same across the designs. In the following section we discuss the process from the viewpoint of a mediator design without loss of generality. The difference is primarily in where the reconciliation occurs.

Enabling Communication

In *Illustration 2*, Tool A and Tool B represent existing systems that can contribute to a new business process by some joint action. Each exposes interfaces for communication, in the form of sharable files, application program interfaces (APIs) or middleware interfaces, such as webservices. A human engineer applies systems engineering techniques to obtain the improved business result. The systems engineer reasons from the desired business result to determine the behavior the new system must yield – the joint action the two systems are to perform. From knowledge of the required system behavior and the interfaces they expose, the system engineer determines the required communications between Tool A and Tool B.

Ideally, Tool A and Tool B were designed to support the target joint action and both were implemented to support that action using a common interface and identical interpretations of the interface elements. When that is the case, the systems engineer need only create the connection between them. In many cases, however, the two systems implement incompatible interfaces or interpret a common standard in differing ways, so that simply connecting them will not produce the joint action.

In order to enable the communication in the latter case, the engineer must identify the appropriate translation of the relevant messages between the two tools, using the interfaces they support. This is accomplished by first linking the concepts and functions that are pertinent to the new joint action to their representations in the exposed interfaces. The systems engineer then uses these links to specify the required interactions between the tools, and the translations of operations and messages that are needed to produce those interactions. From these specifications, software developers can generate integrating code to produce the necessary translations and obtain the new system behavior. The function of semantic mediation technologies is to assist the engineer in generating the integrating code, following the model s/he is implicitly using – matching interface elements based on common meaning.

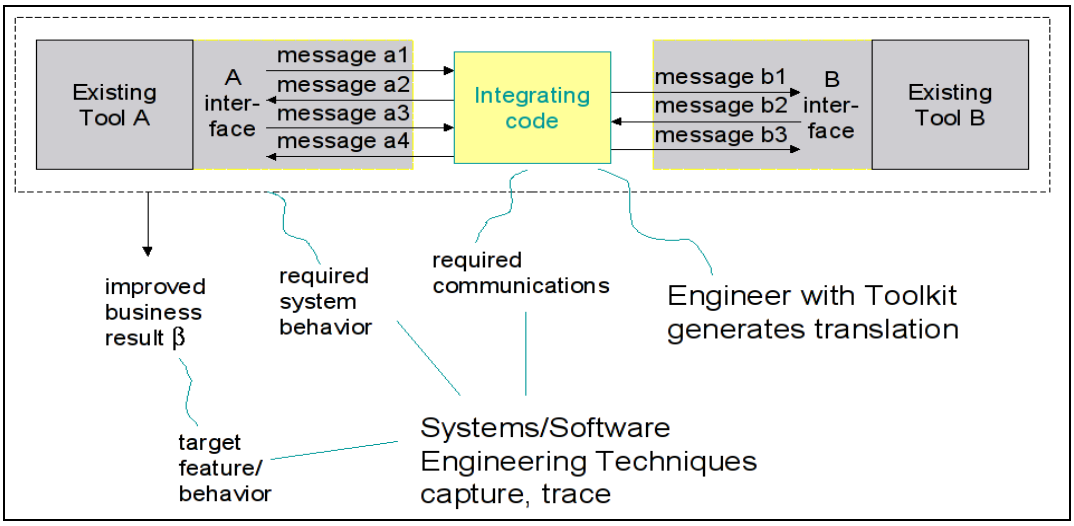

Illustration 2: A high-level view of integration

The universe of discourse relevant to each component system can be represented in a *local ontology*. Likewise, the universe of discourse relevant to the required new business process can be represented in a *reference ontology*. Hameed [4] makes the assumption that each system has its own local ontology, represented by its documentation, and the mediation problem is then a problem of matching independently developed ontologies. He identified three general architectures for ontology-based mediation systems:

(1) pair-wise reconciliation of local ontologies (the any-to-any model), in which ontology matching is done directly between the ontologies of two communicating partners

(2) local ontology to reference ontology (the any-to-one model), in which one reference ontology serves as an "interlingua" to which any participants local ontology may be matched

(3) hybrid, in which each domain has its own reference ontology, each local ontology is mapped to the reference ontology in its domain, and the reference ontologies are matched pairwise as needed to support inter-domain interactions

In our experience, the assumption that each system has its own local ontology is correct, but the available form of that ontology is largely text descriptions and software documentation. This requires then that any matching problem begin with converting an informal document to a formal form. To the extent that this process is manual, when it is done by the systems engineers to enable integration, the conversion itself obviates much of the subsequent mapping process.

Further, Hameed and a number of other academic efforts [5], [6] concentrate on the ontology mapping problem, ignoring the fact that aligning the terms used to describe the functions of system interfaces is only one step in achieving communication between them. It is as if two physicists who agree on the essential theory could communicate even though one speaks French and the other Chinese. Actual communication must be achieved at

the expression level, which for software means at the interface. An equally critical issue in semantic mediation is the relationship between the interface elements and the ontologies.

Finally, the hybrid architecture highlights the idea that a reference ontology can be purpose-built for a domain of known interactions, rather than covering a large domain of all knowledge relevant to a business area. The purpose-built reference ontology can still be used in a larger environment by federation with ontologies for other domains in the larger scope. We have found this to be characteristic of real manufacturing software integration problems, and to be very valuable in establishing the scope in which conceptual agreement must be achieved.

Three Research Projects in Semantic Mediation

Project 1: AMIS

NIST began work on semantic mediation technologies with the Automated Methods for Integrating Systems (AMIS) project in 2001 [7]. The project developed a semantic mediation architecture based on the idea that the enabling communication of a new business requirement must reference a shared viewpoint of the joint action. The enabling communication is analyzed to a set of *required flows* described in terms of an ontology of the business entities and properties of the joint action. *Available flows* are defined in terms of the messages the component systems send and receive. The mediation problem is to map the message structure data elements to the business entities and properties, and to provide the required flow as a transformation of one or more available flows. AMIS defines an approach to automating the generation of the software that performs this transformation.

The approach involves three major activities (See *Illustration 3*):

- Formulation of the Joint Action Model - capturing the business and engineering concerns relevant to the joint action in a form suitable for machine reasoning.
- Semantic Mapping – capturing the relationships between the elements of the exposed interfaces and the concepts in the Joint Action Model.
- Connector Transformation – creating the physical mappings between the interfaces, by generating the integrating software component from the semantic relationships and knowledge of the engineering characteristics of the interfaces. The Message Converter generated by this activity is an ad hoc software component that implements runtime interface conversions between the selected systems.

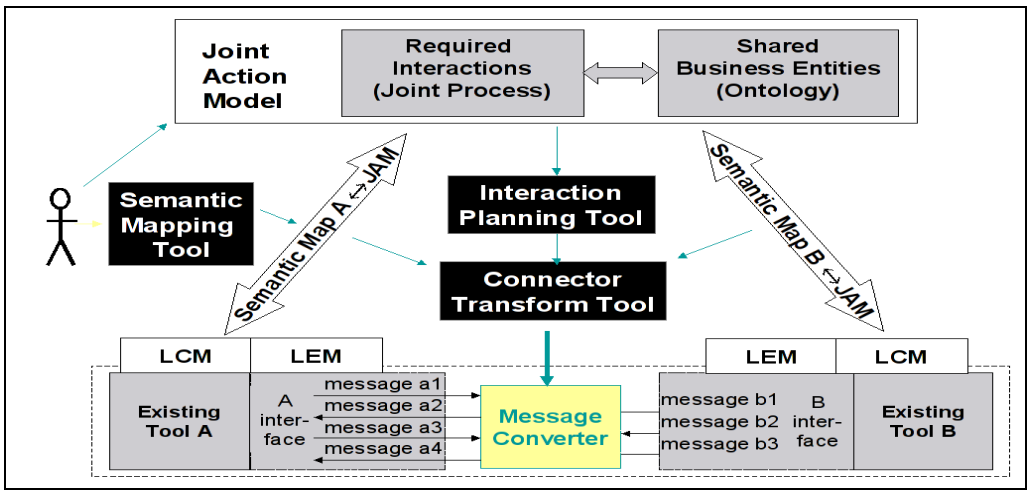

Illustration 3: The AMIS Approach to Integration

The AMIS viewpoint identifies the following conceptual components: *local engineering models* (LEMs) of component systems, that describe the interfaces that support interactions with other components; *local conceptual models* (LCMs) of these components that each describe the entities and relationships relevant to the new interaction from the viewpoint of that component; and, a *joint action model* (JAM) that describes a shared viewpoint on the entities, relationship and actions of the required interaction.

The JAM provides the semantic foundation for the integration activity. It is a model of the concepts, rules, relationships, functions, and actions of the business domain. The conceptualization provided by the JAM sees the interacting agents as playing specific roles in the business process, such as buyer and seller. The model represents

- business actions: the functions and behaviors that implement the roles of each agent, or relate to those functions
- business entities: the objects that are discussed in the communications and used or modified by the joint action
- transactions: notifications, requests for information, requests for functions or services, and responses

Some other semantic mediation schemes see the semantic center as a reference ontology [5] for the business domain generally. The AMIS view of the JAM is that it need only include as much of the domain as is relevant to the intended joint action. Important differences with other semantic mediation schemes are that the JAM includes a model of the business process in which the joint action occurs, and that it defines the roles of the agents in that process.

A local conceptual model (LCM) is a local ontology that describes the entities, properties, and actions relevant to the new business requirement from the viewpoint of a component system. It is possible that the component systems have differing views of the domain, since it is assumed that they were developed in isolation from each other and without foreknowledge of the new business requirement. Artifacts from the development of the component systems, including requirements specifications, class diagrams, activity diagrams, and database schema can provide information that may be restated in the LCM. In AMIS, the specification of the LCM is a manual process.

A local engineering model (LEM) is a model that describes the interfaces supporting interactions with other components. These interactions could be performed as file transfers, database accesses, operation invocations, or message passing. For each unit of communication there is a mechanism, and each agent plays a specific role with respect to that mechanism; e.g., sender or recipient of a message. There is a further level of detail associated with engineering models that defines the detailed protocols for the communications and the binary representations for each data item exposed in the interface.

Generation of the Message Converter and the interaction plan requires knowledge of two forms of mapping. The first is a mapping that relates engineering elements of the interfaces (in a LEM) to corresponding element in the LCM. From the point of view of the development of that component system, these relationships are traces from implementation to intent. The second mapping is from elements in the LCM to corresponding elements of the JAM. This is a mapping from conceptual model to conceptual model (ontology to ontology). A LEM-to-LCM mapping and a LCM-to-JAM mapping is specified for each component system. In AMIS, the specification of these mappings is a manual process. The generation of the Message Converter and Interaction Planner is provided by a software tool that composes the JAM-to-LCM mappings with the LCM-to-JAM mappings. The tool also makes use of a knowledge base for data representation transformations and a knowledge base for the idiosyncrasies of the engineering mechanisms - XML or SOAP, for example - and various protocol implementation libraries.

The Metals Request for Quotation Test Case

Two experiments were performed using the AMIS approach and some versions of supporting technology. The first involved an exchange of “Request for Quote” and Quotation messages between an automotive manufacturer and a sheet metal supplier. The interfaces to the automotive system used standard messages defined by the Open Applications Group (OAG) [8]. The metals supplier used standard messages conforming to the Chemical Industry Data eXchange (CIDX) standards [9]. Both interfaces used compatible (proprietary) message-passing technologies. To simplify the problem, we used the XML version of CIDX, so that the generated connector had only to use the same message-passing library and convert XML message structures to XML message structures. This reduced the required background knowledge to understanding XML.

Each of the standards contained fairly rich documentation enabling the specification of Local Conceptual Models. These models, however, were expressed only in text and diagrammatic form. In the CIDX case, the relationships between the conceptual models and the XML engineering models were only traceable based on consistent naming of entities. In the OAG case, most of the information relevant to integration was provided in the documentation of the individual XML elements; only the outer process models were separate. So the local mappings were easy to formulate.

Since the local engineering models were represented in XML Schema, it was possible to use a tool to extract a representation of the engineering models suitable for use by a human engineer in performing the mappings.

In this exercise, the formal mapping from each LEM to the corresponding LCM was never performed. Instead, the project constructed tooling to allow a human expert to specify the mapping from the simplified engineering model of the XML elements to the corresponding JAM concepts, using the textual documentation of the local conceptual model as a guide. The "composed mappings" were thus generated outright. The Connection Generator Tool [10] was fitted with some specialized XML knowledge and applied to the composed mappings to generate the runtime integrating components. After some additional work to fit the generated component to the proprietary message-passing library, the generated component was plugged into the runtime environment. The messages tested were transformed successfully.

The Automotive Dealership Test Case

The second test case investigated whether information about operations provided by the interfaces of components can be used to automatically compose a plan of joint action to satisfy a new business requirement. This work investigated issues in the development of an interaction plan using an Interaction Planning Tool. The example used was one in which a server of bulk used car buying information can be queried

(1) for a list of cars of a given make, model and year – this operation returns vehicle identification numbers (VINs) of all such cars

(2) for the dealer location of a car given its VIN

The challenge problem placed against these capabilities is that the client component needed the VINs joined with the corresponding dealer locations for every car provided in the response to its query. (The client may be interested in minimizing transportation costs, for example). As a consequence, the request the client would like to make did not correspond to a single (atomic) message in the bulk server's interface. Rather, the request required the composition of results from multiple messages that would need to be planned (atomic actions identified and the order of their execution determined).

In the exercise, relationships were defined between objects in the local conceptual models of the server and client. OASIS Business Process Specification Schemas (BPSS) of the client and server capabilities and object definitions were analyzed by software that could translate the operation specifications into operators of a planning domain to be used by the Interaction planning Tool. The Interaction Planning Tool was implemented as a usage of a domain-independent planning system, SHOP2 [12]. The BPSS referenced XML Data Type Definitions (DTDs) to describe the form of the request and response structures.

The plan-generation software was able to generate a planning domain that the planner then used to produce a successful plan. However, the exercise revealed difficulties arising from the lack of useful information toward this task. One might reasonably expect to find this useful information in the business process specification, but it was not provided in the BPSS. Though BPSS provided definitions of the interface operators by way of the request and response object types, it did not contain provisions to qualify those types in ways that might be important to the user of the interface. For example, an interface operation that responds with junked cars is indistinguishable from one that responds with ordinary cars. Secondly, the BPSS did not indicate what properties may be used as identity conditions on the objects it returns in response. It was necessary to manually specify this in the planning domain.

Project 2: ATHENA

Via participation in an Automotive Industry Action Group (AIAG) project called "Inventory Visibility and Interoperability," NIST worked with semantic mediation tools independently developed in the EU Framework 6 ATHENA project.[13] The test case developed automated mappings between the proprietary interfaces of three commercial "inventory visibility" tools and a proposed AIAG/OAG standard. This proved to be feasible, but not cost-effective. It had interesting intellectual property aspects, and it clarified the requirements for representation capabilities in the ontology, in the message models, and in the transformation rules.

The ATHENA semantic mediation architecture is shown in *Illustration 4*. It involves the following components

- A reference ontology captures the shared concepts of the business interactions that are to be enabled, including the business objects and their properties, the processes and activities, actors/agents. and messages. The ontology was created in the OPAL language, a knowledge representation language specialized for business processes, using the ATHOS tools. [20], [21].

- A message model for each type of message to be used at runtime. The message model is a simple RDF [14] model of each actual message structure and its data elements. The RDF schema supports only the following concepts: *element has name*, *element contains element*, *element has data type* (from a standard list), *element has named value*.
- The semantic annotation set for each such message model. A human-interactive semantic annotation process, using the ASTAR tool, captures the relationships between the message structure elements (as captured in the RDF model) and the business concepts in the reference ontology. The related elements can be described as equivalent, or as functionally related. In the latter case, the function can be a standard one, such as extracting part of a text value, concatenating text strings, converting common date/time formats, or designated as user-defined.
- Two reconciliation rulesets for each such message model. Both rulesets are constructed by semi-automated process, using the ARGOS tool, the reference ontology, the message model, and the semantic annotation set. The *forward ruleset* is a set of executable rules that transform runtime message structure elements into individual RDF instances of classes and properties in the reference ontology. The *backward ruleset* is a set of executable rules that transform RDF instances of concepts in the reference ontology into runtime message structure elements.
- The ARES runtime mediation engine. A running instance of the ARES engine is configured to convert messages in the form output by the sending system (A) to the form expected by the receiving system (B), using the forward rulesets for the messages sent by the sending system followed by the backward rulesets for the messages expected by the receiving system. The rules themselves are written in the JENA rules language and the open source Jena rules engine is the knowledge engineering engine for ARES.

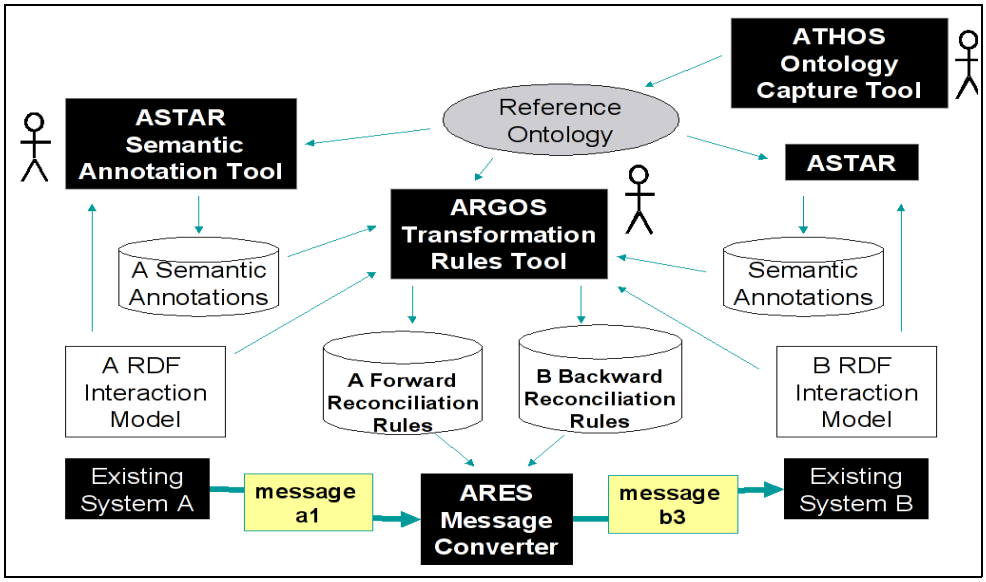

Illustration 4: The ATHENA semantic mediation architecture

Note that the development of the reference ontology for the interactions, the construction of the message models, the development of the semantic annotations, and the development of the reconciliation rulesets are all design-time software engineering activities in which the primary intelligence is the human engineer, somewhat supported by the ATHOS, ASTAR and ARGOS tools. They are applied to each component separately and require knowledge of only the reference ontology and that component system. However, once developed, the rulesets can be used with the rulesets for any other system to achieve effective communication.

There are strong similarities between the ATHENA model and the AMIS model – a single reference ontology (Joint Action Model), engineering models of the individual messages, and semantic mappings/annotations relating the message structure elements to the reference ontology. (While AMIS describes a two-step mapping process and a composition of the mappings, the actual test cases implemented direct mappings from the engineering models to the JAM; the ATHENA architecture posits that single-step approach.) They both depart from the Hameed architectures in the same way – they are about mapping message models to ontologies, not matching ontologies.

The Inventory Visibility test case

The purpose of the ATHENA/AIAG project [17] was to evaluate the use of the ATHENA semantic mediation tools in an actual industrial application. The AIAG scenario deals with the automotive supply-chain situation in which a joint "electronic Kanban" business process regulates the flow of parts from the supplier to match actual consumption by the customer's automotive assembly plant. As conceived by their vendors, "Inventory Visibility" (IV) applications provide web-based interfaces to the parts suppliers and communicate with the specific ERP systems of the automotive manufacturer using proprietary messages. The problem arises when a supplier connected to one of these IV applications is also contracted to supply parts to a different manufacturer, who has a different IV application with a differing, proprietary interface. (See *Illustration 5*). The AIAG solution was to develop a standard set of messages for communications between the IV applications. If each proprietary message can be converted to/from the corresponding standard message using semantic mediation tools, the IV applications themselves do not need to be modified. This was the test scenario.

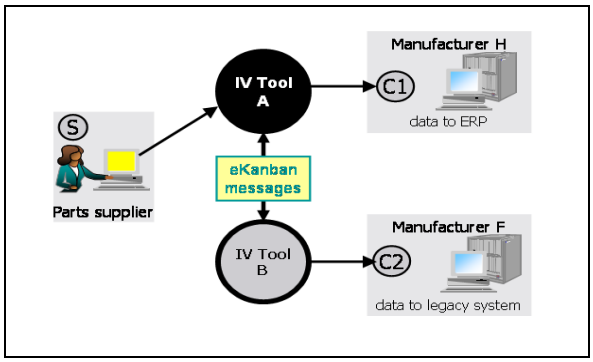

Illustration 5: Inventory Visibility and Interoperability Scenario

This scenario had certain advantages. First, the joint electronic Kanban business process was standardized by AIAG, along with the messages that supported it. This meant that a reference ontology common to all such interactions could be developed from the AIAG standards. Second, because the standards were in the public domain, placing the reference ontology, the semantic annotations, and reconciliation rules in the public domain and making them available to all of the participating IV application vendors would allow effective collaboration. Further, the validity of the approach could be decided by whether each of the vendors could use the tooling to generate runtime reconciliation rules that mapped messages in their proprietary form to and from the standard form. Importantly, they would have exclusive access to the artifacts they created: the message models, the semantic annotations and the reconciliation rules. To the outside world, their tooling, inclusive of their uses of the ATHENA semantic mediation tools, would accept and produce the standard messages.

The e-Kanban business process involved 4 standard messages, loosely referred to as "authorize shipment", "shipment notification", "delivery notification", "consume Kanban notification". (The actual messages were derived from standard messages of the Open Applications Group suite.) The reference ontology supporting the entire e-Kanban business process [8] comprised approximately 50 classes and 150 properties. There were 3 participating IV applications, plus interfaces to the systems of two major automotive manufacturers.

One problem became apparent immediately. The ATHENA toolkit did not include tools for converting XML Schema message models to the simple RDF message model used by the ATHENA tools. So additional tooling was created to do that. It was also necessary to convert the actual XML message structures to a simple RDF form of the message content, in order for the forward mappings to work, and to convert the simple RDF form back to XML for the backward mappings to be useful. [22] Further, the ARES tool had to be embedded in a runtime module that implemented standard communication protocols. In the AIAG test case, the required protocols were "reliable webservices with addressing and security." Using additional ATHENA webservice tooling, this gave rise to the actual runtime architecture depicted in *Illustration 6*.

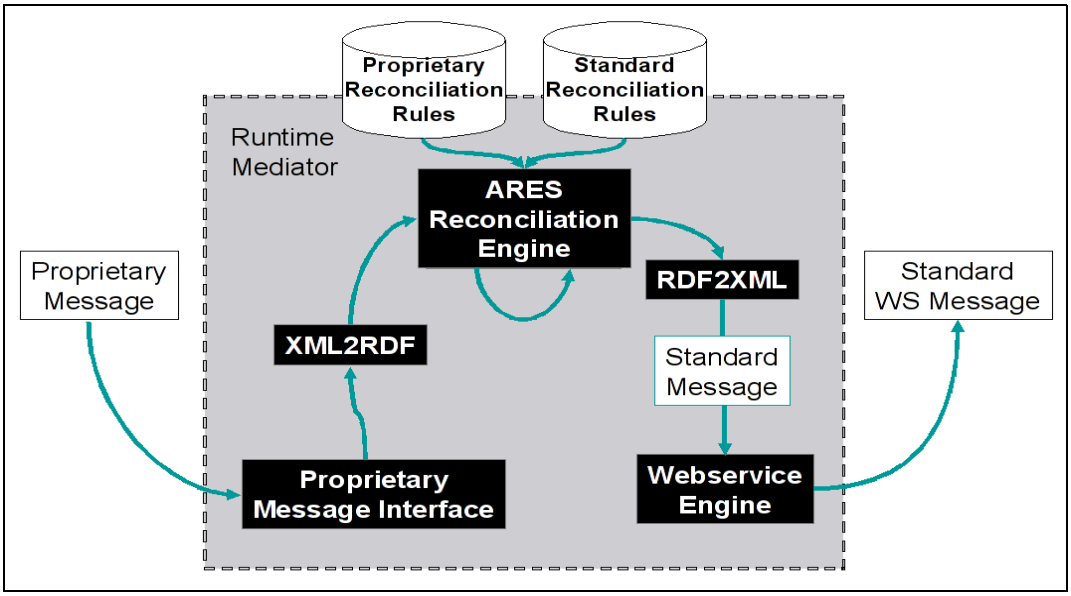

Illustration 6: ATHENA / AIAG Runtime Adapter

The more serious problem with the ATHENA tooling was that the over-simplified RDF form of the message models was not rich enough to guide conversion to a valid XML message structure. The sequencing of elements, multiple occurrences of an element, and the concept of namespaces were not supported. To get this information captured and propagated through the ATHENA tools, and in particular, the ARES runtime engine, this information had to be encoded in the names. That, in turn, made the semantic annotation process much more difficult for the IV application engineers, because the names of their proprietary message structure elements had been significantly modified when they appeared in the tools and were difficult for them to recognize.

The lesson from this experience is that the semantic mediation toolkit is useless if it does not include tools for supporting common message syntaxes and protocols used in industry. Today, support for XML message structures described by XML schemas and conveyed by SOAP [18] or ebMS protocols [19] is the minimum requirement. The academic favorite, RDF, is simply not used in industrial communications.

Ultimately, we were able to demonstrate exchange of one of the standard messages between two IV applications and between them and the manufacturers systems. It would not have been difficult to complete the set, but it was apparent to all involved that the name-tracking work-arounds had made the experience unpleasant for the IV vendors. With the "first-draft" ATHENA tooling, the development process using semantic mediation was not cost-effective, and they withdrew their resources after the first demonstration.

In addition, the ATHENA project conducted an end-user test of semantic interoperability involving one IV application and the automotive manufacturers ERP system. In this exercise, a supplier's shipment notification was created in the IV application and sent, in proprietary form, to the mediator for that application, which converted the message to the AIAG standard form. The standard form was sent to the manufacturer's mediator, which converted it to the proprietary form for the manufacturer's ERP system. A human "material manager" for the manufacturer inspected the state of that Kanban loop using the human interface to the ERP system and verified that the information provided by the supplier had been correctly recorded.

Three discoveries of this work are highlighted here because they were unexpected and also because they influenced the direction taken in the subsequent MOSS project (the third project, discussed below). First, the AIAG community found the development of the formal ontology to be of significant value in clarifying and documenting the intended e-Kanban business scenario. Second, they considered that the semantic annotations, viewed as a trace from the business concepts to the XML implementations, would be of significant value to the software engineers - if and when the noise introduced by the oversimplified RDF message model was eliminated. That is, they saw significant value in the capture of the business model as an ontology, and in the trace from the ontology to the XML implementation, independent of run-time semantic mediation.

The third discovery is that the ATHENA team was surprised that the test scenario involved conversion from proprietary message A to standard message, and from standard message to proprietary message B. The scenario they envisaged was direct conversion from proprietary message A to proprietary message B within the ARES

tool. With the proprietary mappings actually developed, that pair of rulesets could have been loaded into ARES and the rulesets for the standard discarded. In that configuration, the value of the AIAG standard is the standard e-Kanban reference ontology, not the standard XML schemas for the exchange messages.

The ideas that development of the business ontology clarifies understanding of subject business scenarios, that trace relationships from messages to the business domain ontology served to guide tool implementations of the messages, and, that the ontology might serve as an interlingua in message communication were each embraced in the development plan of the MOSS project, the third of the projects discussed in this case study.

Project 3: MOSS

NIST is currently involved in an automotive industry project, the AIAG's Materials Off-Shore Sourcing, (MOSS) which seeks to reduce the cost and improve the performance of intercontinental automotive manufacturing supply chains. The scope of the project is ocean-going automotive parts from order placement to the point the goods are delivered to the customer. Begun in May, 2005, the project is on-going at the time of this writing. There is not sufficient experience at this time to make practical evaluations, but the technical issues and planned work will be discussed.

The MOSS project was motivated by a survey of automotive industry stakeholders that showed that errors in the information coordinating the movement of goods was causing 15% of shipments to be delayed en route. Unanticipated delays can result in substantial variation in transit times. To ensure the availability of components despite this variation, manufacturers must carry additional, costly buffer stock. The survey also found that customers had limited visibility into the progress of a shipment as it moved through the supply chain.

The survey revealed some of the sources of the problems experienced. Oftentimes, processes involved paper documents, fax and email. These forms of communication proved error-prone and costly due to the need to manually copy information from one media to another. Electronic Data Interchange (EDI) technology is pervasive in the automotive industry, but messaging between the systems supporting the processes was hampered by inconsistent interpretation of the content. In summary, the study revealed that a lack of semantic interoperability (but also error-prone manual processes) resulted in costly inefficiencies.

The problem space of the MOSS project is distinguished from the others described in this case study in that (1) its processes are supported by a large installation of legacy information technology; (2) the systems must support substantial variation in business processes and interact with many partners; (3) there are many roles to be played in the processes (customers, suppliers, freight forwarders, ocean carriers, pre-carriage and on-carriage carriers, customs brokers, import and export customs authorities, warehouse operators, and third-party logistics provider, among others); and, (4) MOSS is defining industry standard message definitions as well as participating in the NIST research.

Typically, the systems in the supply chain logistics community support EDI interfaces. XML messaging in this community is almost non-existent. The community is experienced in adapting its systems to support new business requirements by modifying the EDI messages and the systems that use them. These facts, the diversity of the ownership of the systems, and their loose federation, all worked against solutions that would deploy mediators.

MOSS needs to enable semantic interoperability through the standard EDI messages it is developing. However, EDI has little semantic foundation, and its modeling methodology has serious shortcomings. To address these problems, the project is defining mappings from its EDI message structure elements to structures in a Supply Chain Logistics Conceptual Model, a reference model for MOSS. The conceptual model shall be mirrored by an ontology on which reasoning can be performed. Validation requirements are defined in terms of the business concepts in the ontology. Through this approach, we will use the ontology to investigate a new modeling methodology and to support the validation of implementations of the MOSS messages.

A commercial technology provider is collaborating on the project to address an additional problem with the current-state process: the high cost of establishing the communication channel among participants intending to collaborate in trade relationships. That software is implemented upon a viewpoint closely resembling the MOSS reference model.

EDI standards such as EDIFACT [23] and ANSI X12 [24] define widely-used message types covering processes in the scope of the MOSS project. A key task addressed in deploying EDI is that of tailoring the standard message type definition (the “directory message type”) to the needs of the business process. EDI's directory message type definitions contain provisions for an immense spectrum of situations, only a small subset of which

are likely to be relevant and supported in an actual usage of the message type. In EDI technology, a document known as a *Message Implementation Guide,* (MIG) is used to identify a subset of the directory message structure elements that will be used in actual messages. A MIG is an engineering model of a file-based interface. The partners collaborating in the design of an EDI message supporting a new business requirement use the MIG to identify what information will be included in the exchange. In order to designate the business purpose of an element conveyed in a message, an EDI message designer may be required to associate with the element a term from a code list. A *code list* is a dictionary of terms and their associated natural language definitions that enumerates the values of some domain property. For example, EDIFACT provides a code list enumerating business process roles. This code list enumerates about 200 roles, and provides short English language descriptions of the intended semantics. The code term "BY" is defined "Buyer. Party to whom merchandise and/ or service is sold." [23]

In practice, EDI's code lists are a major contributing factor to its failure to provide a semantic foundation for messaging. Before the details of this problem are described, however, it is important to recognize that many of the code lists used in business serve important roles beyond their use in messages. For example, INCOTERMS [17] is a code list describing 13 patterns for the sharing of risk and transport responsibilities between a buyer and a seller. Stakeholders use these terms to concisely describe the logistics relationships they intend. Acceptance to operate under a given INCOTERM entails legal obligations consistent with its definition. Business processes are structured around some code lists; typically, process stakeholders are conversant in these.

Certain industrial and government organizations maintain registries of individual entities. In EDIFACT these registries are considered code lists, but they are in fact quite different. The registry identifies individuals of the type that is its concern. For example, a tax registry may identify a taxable organization in the context of a government's process for assessing a tax. Processes for which the registry's "Tax ID" identifier is a relevant property may treat the property as closed. Further, it may be impossible in the context of these processes for an entity not represented in the registry to play roles that one might casually associate with it. In essence, it would be of the type, but for its absence from the registry.

In practice, the principal problem with EDI code lists is that it is oftentimes not possible to determine from a code term's definition what business entity it designates. Thus when multiple parties independently create MIGs, they may choose differing code terms to describe identical entities, or they may use a single code term in a situation where a party with whom they communicate requires that a distinction be made, (perhaps using multiple code terms).

A second problem with the EDI modeling methodology is that there is nothing in it that provides knowledge of the correspondence (or "life-cycle") of its units of information across the messages in which they might appear – the methodology does not enable one to specify in what message and business process a unit of information is introduced, and in what messages and business processes it is reused. Lacking this, it is difficult to form a cohesive view of the business process at the level of message detail. Consequently, interpretation errors are made in implementing the messages.

The first step in the MOSS solution to these problems is to identify, through an analysis of the business process, the enabling business-relevant information. This analysis identified about 400 units of business information ("MOSS Properties") used across the 20 messages that were considered essential to managing the process ("MOSS Messages"). The 400 MOSS Properties are represented by about 1400 EDI information elements. The growth is due to the fact that the EDI encoding of a unit of business information may need to make reference to, for example, one code list to associate meaning to the value, and another to indicate the syntactic encoding of the value. For example, an instance of the MOSS Property "Requested Delivery Date" may be encoded as "DTM+2:20090803120000:204". In this structure, the "+" and ":" separate components of the structure. "DTM" indicates that this is an EDIFACT "Date Time" message structure element. The first position of this structure, containing the value "2," is a code term from an EDIFACT code list. "2" designates "Date on which buyer requests goods to be delivered." The value "20090803120000" is the timepoint noon, August 3, 2009 (Though it is ambiguous; the time zone has not been specified). "204" is a term from an EDIFACT code list describing various encodings for timepoints.

The 1400 EDI properties were mapped to EDI structures in the 20 MOSS Messages. Though this suggests that 28,000 units of mapping mappings may need to be specified, in practice only about one third of the business units are relevant to the process supported by the message. The mapping was performed manually, but automated tools were used to validate (1) the mapping consistency across messages, (2) the consistency of EDI encoding, and, (3) the correctness relative to the EDI directory message type. Mapping consistently across messages ensures that we do not introduce multiple message structure elements to represent a single business-level unit of information. Mapping to a consistent EDI encoding reduces the cost and complexity of

implementing the messages. Mapping consistent to the EDI directory type ensures that implementers can use their usual EDI tooling in providing an implementation.

The end products of the process just described are MIGs, similar to what would be produced by the conventional EDI modeling methodology except that our process additionally provides (1) knowledge of the correspondence of the units of information across the messages in which they appear (life-cycle information), and (2) consistency across messages of the EDI encoding of the information. MIGs are similar in purpose to the Local Engineering Models of AMIS, but in MOSS they are more accurately "Reference Engineering Models" since they describe an AIAG MOSS standard interface. They are documented by web-based tools that are used by project participants implementing the MOSS standard. [27]

As in AMIS and ATHENA, the means by which we provide a semantic foundation for messaging is to relate message structure elements to a domain ontology. MOSS maps units of business information from the MIG engineering model (EM) message structures described above to the MOSS Supply Chain Logistics Conceptual Model (CM). In MOSS, the first role of these mappings is to provide engineers implementing the standard with knowledge of how the message structure elements should relate to structures in their implementations.

The MOSS Supply Chain Logistics Conceptual Model, currently under development, [26] is similar to the AMIS Joint Action Model in that it describes business entities and transactions. However, it does not contain action descriptions, and it is scoped to the entire business process, not a single joint action. In MOSS, action descriptions are represented only informally, through activity diagrams that each reference multiple actions of typical scenarios. These scenario include, for example, Asia Pacific full container load shipments to the US, Europe less-than-container-load shipments consolidated and sent through Canada to the US, and so on.

Since MOSS is specifying a standard, "local" viewpoints (those of existing systems) are not represented. The mapping is done in one step, from the engineering models to the conceptual model (EM-to-CM).

Semantic Interoperability Testing in MOSS

Information systems conforming to the MOSS standard must be able to send and receive MOSS messages. Compliance with the standard requires that the data elements within the message structures be interpreted as defined, and that the interchange between the systems follows the business process as specified. Had the focus of MOSS, like AMIS and ATHENA been the automatic generation of mediators and adapters, testing could have been limited to assessing the soundness of the message transformations those mediators and adapters produced. MOSS does not specify a process by which mediators and adapters may be generated. MOSS tooling does not link to knowledge of local engineering models – it has no direct knowledge of how implementations view elements of the Trade Logistics Conceptual Model.

One strategy MOSS is pursuing to assess the correctness of implementations is to assess whether they can correctly populate a knowledge base mirroring the Supply Chain Logistics Conceptual Model. The EM-to-CM mappings described above are executable in the sense that a "mapping engine" can create from a message instance a population of entities and relationships conforming to the conceptual model. The mapping engine uses the Queries, Views and Transformations (QVT) relational language. [28] The mapping performed by the QVT engine is not just a purely syntactical translation. During the mapping process, the information contained within the messages is enriched with additional information from a MOSS formal business process model and a reference ontology. The aggregate model consisting of the reference ontology and the formal business process model can be viewed as a representation of the MOSS standard in a formal language. A population of contingent facts are derived from messages produced by the implementation, representing its view of a challenge problem posed to it. These are placed in a message assertion base .(See *Illustration 7*, below.)

By enriching the information from messages presented to the validation tool with other information about the trade lane, defaults, closure properties, and other considerations, the reasoner can perform some validation tasks on the collection of messages presented to it. Messages describing a span of interactions in the supply chain.

The reference ontology contains a classification of all the entities that are relevant within a manufacturing supply chain, the relations among these entities, and an axiomatization of these relations. The formal business process model specifies the intended process flow for supply chain processes including the information flow; for example, how milestones within a supply chain process are linked to MOSS messages.

The logical representation of the messages in the message assertion base allows an inference engine to test for semantic interoperability. We distinguish five different kinds of test

1. **Content validation** contains tests that evaluates individual messages given the axioms of the reference ontology.
2. **Information flow validation** consists of test that determine whether the messages are sent and received in the appropriate order, and whether messages are missing.
3. **Process flow validation** consists of test that determine whether the events within the shipment process meet the requirements of the business process model. Since many of the messages are linked to milestones of the shipment process, information flow validation and process flow validation are closely linked.
4. **Consistency validation** utilizes the power of the reasoning engine to determine whether the available information is consistent within and across the messages. Inconsistencies might be explicit – for example, in the case where a value was copied wrongly from one message to another – but the inconsistencies might also be implicit and only discovered by the reasoner with by reasoning with the axioms within the reference ontology.
5. **Assertion validation** utilizes additional knowledge to evaluate the information. E.g., one can use external knowledge bases to check DUNS numbers or postal codes.

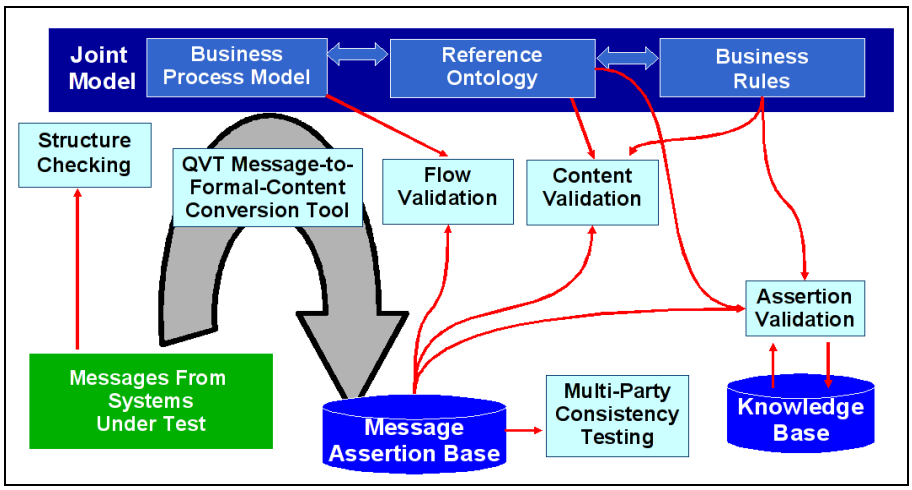

Illustration 7: Testing in the MOSS project

Conclusion

This case study reports on our experience investigating issues of semantic interoperability through three related research projects. We investigated architectures and procedures for the reconciliation of differences of viewpoint that are exposed in component interfaces. The essential form of our solution is to relate message structure elements to elements of ontologies.

The task to be automated determines the knowledge that needs to be formally represented. Much of the recent research in semantic interoperability concerns ontology matching ("reconciliation in the abstract"). Accordingly, these efforts make reference to ontologies that do not include representation of the supporting information technology and its interfaces. Another community, working in near isolation from the former, has produced service oriented architectures that provide scant reference to the domain ontology, and an inscrutable shorthand of the systems engineer's reasoning. The result is that practical automated semantic interoperability seems a long way off.

Achieving the results we sought required that we enrich message structures and interfaces with links to the domain ontology. Since these domain ontologies usually do not exist, they have to be developed as part of the integration process. One major obstacle is the poor quality or even lack of proper documentation, which required that we perform time-intensive "archeology" to recover engineering representations of interfaces capable of serving our goals. On the upside, our experience suggests that an ontology that serves to automate mediation has value beyond this narrow role; it may serve human system integrators in the manual process of system integration, and it may improve the quality of exchange specifications. Another lesson learned was that

implementing so-called "semantic technologies" was only a small fraction of the actual work, because there was substantial technical minutiae to address. Overcoming these obstacles does not so much require a reconceptualization of how systems and interface standards are made, as it does a commitment to capture systems engineering rationale as it is established, and in a form that facilitates its use in future systems integration. In systems development, the technical and business-domain specific needs of future systems integration projects often cannot be foreseen, but the need for a methodical systems engineering practice, performed in advance, can be.

References

[1] IEEE Computer Society, (2004) *Guide to the Software Engineering Body of Knowledge*, http://www.swebok.org/, retrieved February 10, 2009.

[2] International Council on Systems Engineering (INCOSE), (2009) *Guide to the Systems Engineering Body of Knowledge*, http://www.incose.org/practice/guidetosebodyofknow.aspx, retrieved February 10, 2009.

[3] Gamma, E. et al. (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley.

[4] Hameed, A. et al., (2004). *Ontology Reconciliation*, Handbook on ontologies, Springer-Verlag, 231– 250.

[5] Dell'Erbaa, M. et.al., (2002). *HARMONISE: A Solution for Data Interoperability*, Proc. of IFIP I3E 2002 Conf., 114-127.

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

[7] Libes, D., et al., (2004) *The AMIS Approach to Systems Integration*, NISTIR 7101 http://www.mel.nist.gov/msidlibrary/doc/nistir7101.pdf, retrieved August 2008.

[8] Open Applications Group, (2003) http://www.openapplications.org.

[9] CIDX, (2003) *CIDX Overview*, available from: http://www.cidx.org/AboutCIDX

[10] Flater, D., *Automated composition of conversion software*, NISTIR 7099, (2004) http://www.mel.nist.gov/msidlibrary/doc/nistir7099.pdf, retrieved August 2008

[11] BPSS

[12] D. Nau, H. Muñoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. *Total-Order Planning with Partially Ordered Subtasks*. In IJCAI-2001. Seattle, August, 2001.

[13] Athena A3 *Knowledge Support and Semantic Mediation Solutions* (2004), European integrated project, online at http://www.athena-ip.org. retrieved August 2008

[14] World Wide Web Consortium, *RDF Vocabulary Description Language 1.0: RDF Schema*, online at http://www.w3.org/TR/rdf-schema/, retrieved August 2008.

[15] Jena rule language, (2007) online at http://jena.sourceforge.net/, retrieved August 2008

[16] Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Application (ATHENA) (2007) B5.10 - Inventory Visibility Sub-Project : IV&I End-to-End Interoperability Demonstration including Conformance Testing Demonstration, online at xml.aiag.org/athena/resources/WD.B5.7.6--InteropAndConformanceTestDemo.pdf , retrieved August 2008

[17] Barkmeyer, E., Kulvatunyou, B., (2007). *An Ontology for the e-Kanban Business Process*, NIST Internal Report 7404, Available from: http://www.mel.nist.gov/msidlibrary/doc/NISTIR_7404.pdf, retrieved August 2008

[18] World Wide Web Consortium, (2000). *Simple Object Access Protocol, (SOAP) 1.1*, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ retrieved February 9, 2009.

[19] Organization for the Advancement of Structured Information Standards (OASIS), (2002) *Message Service Specification, Version 2.0. http://www.ebxml.org/specs/ebMS2.pdf, retrieved February 9, 2009.*

[20] Missikoff, M., (2000) OPAL A Knowledge Based Approach for the Analysis of Complex Business Systems. Rome: LEKS, IASI-CNR.

[21] Missikoff, M., Taglino, F., (2007) ATHENA Document D.A3.1 - Part 3, "*The ATHOS User Manual*", available from http://www.athena-ip.org

[22] Miletic, I., Vujasinovic, M., Ivezic, N., and Marjanovic, Z., (2007) "*Enabling Semantic Mediation for Business Applications: XML-RDF, RDF-XML, and XSD-RDFS Transformation,"* In Proceedings of International Conference on Interoperability of Enterprise Software and Applications, Springer-Verlag, pp. 483-494

[23] UN/CEFACT, (2009) United Nations Directories for Electronic Data Interchange for Administration, Commerce and Trade, http://www.unece.org/trade/untdid/welcome.htm, retrieve February 9, 2009.

[24] Accredited Standards Committee, (2009) ANSI X12, http://www.x12.org/ retrieve February 9, 2009.

[25] INCOTERMS, *Rules at the Core of World Trade,* http://www.iccwbo.org/incoterms/id3045/index.html retrieved 2008-11-10.

[26] AIAG MOSS Project, (2008) MOSS Conceptual Model (A UML model) http://syseng.nist.gov/poc/repo/trunk/conceptual-model/moss.mdxml, retrieved February 9, 2009.

[27] AIAG MOSS Project, (2009) MOSS Message Mapping, http://syseng.nist.gov/moss/moss-views/msg-view?msg=DELJIT&select=1185&open=1202*1219#1185, retrieved February 9, 2009.

[28] Object Management Group, (2009) Meta Object Facility (MOF) 2.0 Query/View/Transformation, V1.0, http://www.omg.org/spec/QVT/1.0/, retrieved February 9, 2009.