NISTIR 7101

The AMIS Approach to Systems Integration: An Overview

Don Libes
Edward J. Barkmeyer
Peter Denno
David Flater
Michelle Potts Steves
Evan Wallace
Allison Barnard Feeney

National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 7101

The AMIS Approach to Systems Integration: An Overview

Don Libes
Edward J. Barkmeyer
Peter Denno
David Flater
Michelle Potts Steves
Evan Wallace
Allison Barnard Feeney

Manufacturing Systems Integration Division
Manufacturing Engineering Laboratory

May 2004

U.S. Department of Commerce
Donald L. Evans, Secretary

Technology Administration
Phillip J. Bond, Under Secretary for Technology

National Institute of Standards and Technology
Arden L. Bement, Jr., Director

Abstract

This paper is an overview of the AMIS (Automated Methods for Integrating Systems) project approach to systems integration. The objective of the AMIS project is to reduce the cost and time for software integration by devising methods, algorithms, and tools by which activities of a systems engineer can be automated. The motivation for this work is to reduce the expense of integration efforts where traditional standards-based approaches are inappropriate or ineffective, e.g., where the time it takes to develop a standard is longer than the life of the integration problem. The anticipated benefits of this project include: improving interface/service specifications, improving knowledge capture for existing software systems and standards, reducing the time and cost of systems integration projects, identifying the unsolved problems, and providing knowledge for new toolkits.

The AMIS approach is based on the idea that the published interface specifications for a software system can be abstracted into an understanding of the roles in the business processes the system was built to support. Those roles can be formalized into specifications and models for interactions in which each element will be associated with the corresponding business action/entity/property notion. When an engineer devises new processes, software may be able to match existing role definitions for component systems and implement those roles along with any needed choreography and process-specific wrappers that transform the physical message sequences on the basis of equivalent business notions.

Keywords

AMIS; Systems Integration; Semantics Integration, Automated Systems

Introduction

Manufacturing systems become more complex every year. New business requirements require new functionality. Often, these new systems are built with last year's systems as components. These components are made to work together by employing systems integration methods [2]. In many cases, the systems of a manufacturer must communicate with the systems of a supplier or customer as well, in what is called an "e-commerce" or "supply-chain integration" activity.[1] In these cases, a new system is, in effect, being built on the spot from the communicating components of the two organizations.

A typical integration problem is stated as a requirement to produce an improved business result from a set of available systems. Each of the available systems exposes interfaces by which it can interact with people and other systems. Integration is accomplished by building a new system that includes a set of components that are used as is or modified to accomplish new functions jointly. The process of integration is the activity that produces a connector between selected interfaces that leads to the improved business result.

The current manual process of integration involves systems engineering techniques employed by human experts that reason from the required results and the behavior of the existing components to the required behaviors of the modified system. Human engineers then design and create the communications/interactions between the components that will produce that behavior. Then software design and development are used, again by human engineers, to produce those communications by building some kind of connector that uses the available interfaces.

The effort that goes into these point-to-point integration efforts is not readily reusable. Subsequent integration problems require another integration effort. In June of 2001, the Aberdeen Group reported that on average, 40% of enterprise information technology budgets are allocated to integration. [1] One solution is effective standards; however this is unrealistic. Industry requirements change faster than we can create standards. [9] And, every industry has domain-specific requirements that affect standards development and use. So standards usually result in small communities of interoperability – some systems that together, largely by design.

---

1. Any commercial product or service identified in this document is for the purpose of describing a software environment only. This identification does not imply any recommendation or endorsement by NIST, nor imply it is necessarily the best product or service available.

In the particular case of supply-chain integration, systems supporting two independent enterprises need to communicate in order to do business. A number of standards have been developed over the last 20 years to address this problem, and there are now several competing standards. Each of these standards involves many end-user options for how they will actually be used to do business and what information they will contain. In essence, any of these standards specifies a family of similar interfaces, no two of which are exactly alike. " Middleware" solutions, such as ebXML, CORBA, Webservices, and .Net, standardize the communications techniques but not the actual messages. [8][11][12][7] Applications still have to agree on messages, content, and terms, i.e., use a common standard. Or, someone has to translate them.

A good example is the recent ebXML effort. This standardization effort has produced agreement on an automated mechanism for determining whether both trading partners can find a common variant of one of these standards that both partners support. But agreements on what those variants include are being developed by pilot programs of specific major manufacturers and by industrial and regional consortia. The result is a large body of similar "standards" that conflict in many details, and the ebXML engine installed in any given trading partner site supports only a few. In effect, the integration problem has been changed but not solved by these new technologies and "standardization efforts."

Because of the pace of changing requirements, growing systems complexity, cost of point-to-point integration efforts, and gulf left between two applications attempting to communicate using the same middleware standard, the AMIS project is developing an approach for automating parts of the integration process for manufacturing software systems. The activities that show the most promise for automation are:

- Conceptual design: reasoning from available and required behaviors and the available interfaces to a set of communication actions that will produce those new behaviors, and
- Engineering design: generating or including software modules that perform those communication actions using the available interfaces.

The systems engineering functions that determine the required behaviors from the target business results and other requirements will continue to require human expertise.

The Integration Problem

An integration problem is typically stated as a requirement to produce an improved business result from a set of systems and components. At times, some of the required component functionality must be obtained or created anew and then installed before it can be integrated. To illustrate the integration concept, we portray a simplified integration problem between two existing tools, although in reality several such integrations may be required to obtain the over-arching improved business objective.

Each of the systems that contribute information or functionality to the improved business result exposes interfaces. These interfaces are the communication endpoints by which the systems can be considered to be components that will interact with each other and thereby form a new integrated system. The process of integration is the activity that joins the components appropriately, with modification if necessary, so that that the components together may accomplish new functions. Functions accomplished by such interactions are called *joint actions.*

As shown in figure 1, tool A and tool B each expose interfaces for communication, in the form of sharable files or Application Program Interfaces (APIs) or middleware interfaces, like Java or 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 target feature or behavior the new system must yield – the joint action the two systems are to perform. From the required system behavior and the interfaces they expose, the system engineer determines the required communications between tool A and tool B. To enable the communications, the engineer must generate the appropriate translators of the relevant messages between the two tools using the interfaces they support and forms that are meaningful. 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.

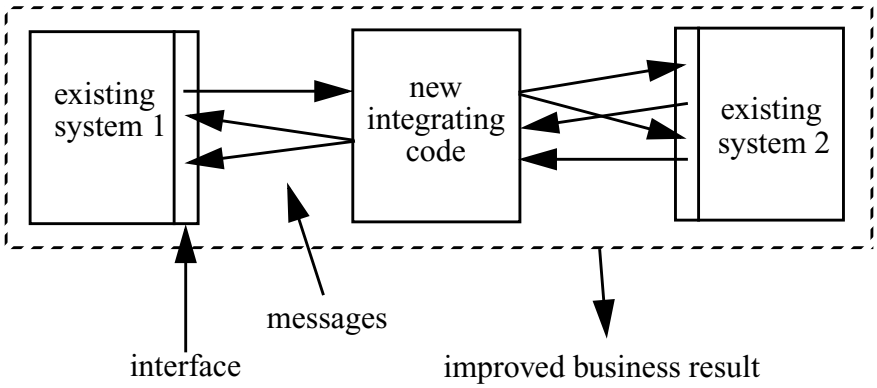

Figure 1. In the typical integration problem, the new business result depends on new features and behavior, which requires new communications and integrating code. The integrating code may turn a single message into multiple messages in order to accomodate tools with different interfaces.

The *research question* is: can we automate building translators that work directly from the interface specifications, as human engineers do?

The AMIS Approach

There are three main areas of work in the AMIS project. Each area must be solved to support automated integration.

- Joint Action Model Formulation - capturing the business and engineering interaction concerns in a form suitable for machine reasoning.
- Semantic Mapping - building tools to implement semantic links between models.
- Connector Transformation - building tools to create semantic maps between models.

Joint Action Model Formulation

As stated earlier, integration is a systems engineering activity that identifies how two or more software components should act jointly to perform a role in an improved business process. In doing so, the systems engineering activity also must assign roles to the actors in the revised business process and implement the necessary communications between the systems. The resulting integration is a new system that implements the new business process.

Systems engineers perform an integration task using a combination of top-down and bottom-up approaches to match business process objectives with component functionality. Our approach is to formalize and capture the information the systems engineer uses in a way that allows us to use software-based reasoning tools to automate parts of the integration task, and assist the systems engineers in performing others. For this purpose, we use a variety of specialized models.

The JAM (Joint Action Model) is a requirements model for the intended joint action. [5] The JAM is abstracted from the relevant concepts in the envisioned business process. The JAM specifies the required actions of interactions between the component systems, their conceptual interactions with roles for each, and a shared model of the business entities pertinent to those interactions. The target business process model and the Joint Action Model are specified by humans. The conceptual formulation of a JAM is shown in figure 2. For our approach, the JAM captures the business interaction concerns in a form suitable for machine reasoning. That is, it goes beyond UML-like models of entities and actions to include some formal definitions of the business terms used and the relationships among them.

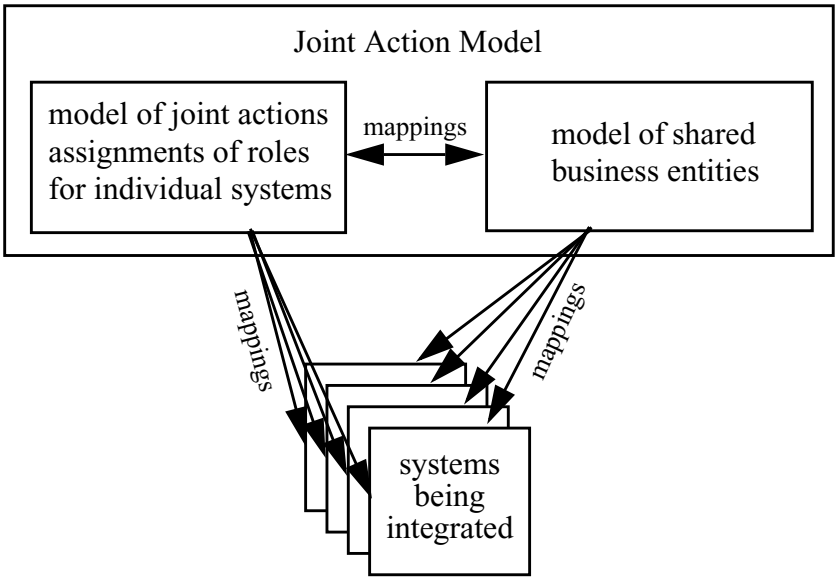

Figure 2. The JAM models the actions and the business entities that together produce the improved business result.

In modeling joint action and communication, there are two distinct levels of abstraction that are significant: the conceptual model and the engineering model. These views are important because solutions to integration tasks are conceived in business/conceptual terms and implemented in engineering terms.

The *conceptual model* describes concepts, rules, and relationships of the business. The conceptual model of the joint action sees the interacting agents as playing specific roles in the business process, such as buyer and seller. The model characterizes the interaction as flows of information and service requests. At this level, the model is defined in the following terms:

- 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, and
- transactions: notifications, requests for information, requests for functions or services, and responses.

An *engineering model* of an existing system includes models of the interfaces supporting possible interactions with other systems or agents. The engineering model of the joint action sees interacting agents as software components communicating by one or more mechanisms, such as file transfer, database access, operation invocation, or queued messaging. For each unit of communication there is a mechanism, and each agent plays a specific role with respect to that mechanism, *e.g.,* file writer or distributed object server. And for each unit of communication, there is a message – the set of information transferred by that communication unit. There is a further level of detail associated with engineering models, which defines the detailed protocols for the communications and the binary representations for each data item. Engineering models also contain terms for message types, operations, information units, and so on.

Although conceptual views and engineering views serve different roles, they are not used in isolation. Inter-model relationships between model elements from the engineering and conceptual views establish the relationships between an activity or entity expressed in business terms and an engineering means implementing that activity or representing the entity. When integration is performed manually, the knowledge that relates those views often exists only in the mind of the systems engineer. Our method formally defines these inter-model relationships that relate model elements from various models. We refer to an engineering model linked to concepts in the conceptual model as the *Local Interaction Model (LIM*) of an existing system, as shown in figure 3. We call the links *links-across-views*. The result is multiple LIMs, one for each system modeled

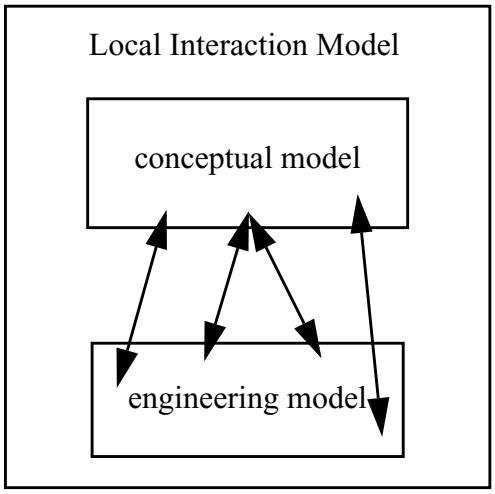

Figure 3. Local Interaction Model links concepts in the conceptual model and engineering model of each system.

Once developed, the Local Interaction Models correctly describe the expected business interactions of the component, for a number of purposes, not necessarily restricted to the concerns in the JAM.

Semantic Mapping

After the LIMs and the JAM have been produced, the next step of the process is to map the relevant notions in the Local Interaction Models to their corresponding notions in the Joint Action Model. Figure 4 shows how the JAM uses links-across-views to relate views of the participants to shared concepts. Links-across-views from the JAM to the LIMs relate roles in the transaction to resources in the linked models of the participants.

The AMIS approach is to build tools once to automate the creation of the semantic maps between a new JAM and the (existing) LIMs. This is the difficult research problem – automating the creation, derivation, or extraction of semantic relationships between models.

Given the links between the JAM and the LIMs, the engineering model for the Joint Action can be built. It is shown in the middle of figure 4. This model uses the interfaces identified in the engineering models of the participating tools to achieve the goal described by the JAM. The technical detail references the appropriate message, data, and datatype(s) for equivalent terminology.

Connector Transformation

Given a sufficiently detailed semantic mapping, it is theoretically possible to build a tool that generates translations corresponding to the mappings. This is shown in figure 5. To achieve arbitrary transformations of syntax, structure and interactions to the lowest levels of abstraction requires that all the information be formalized. Additionally:

- knowledge bases for the middleware technologies must be developed, and

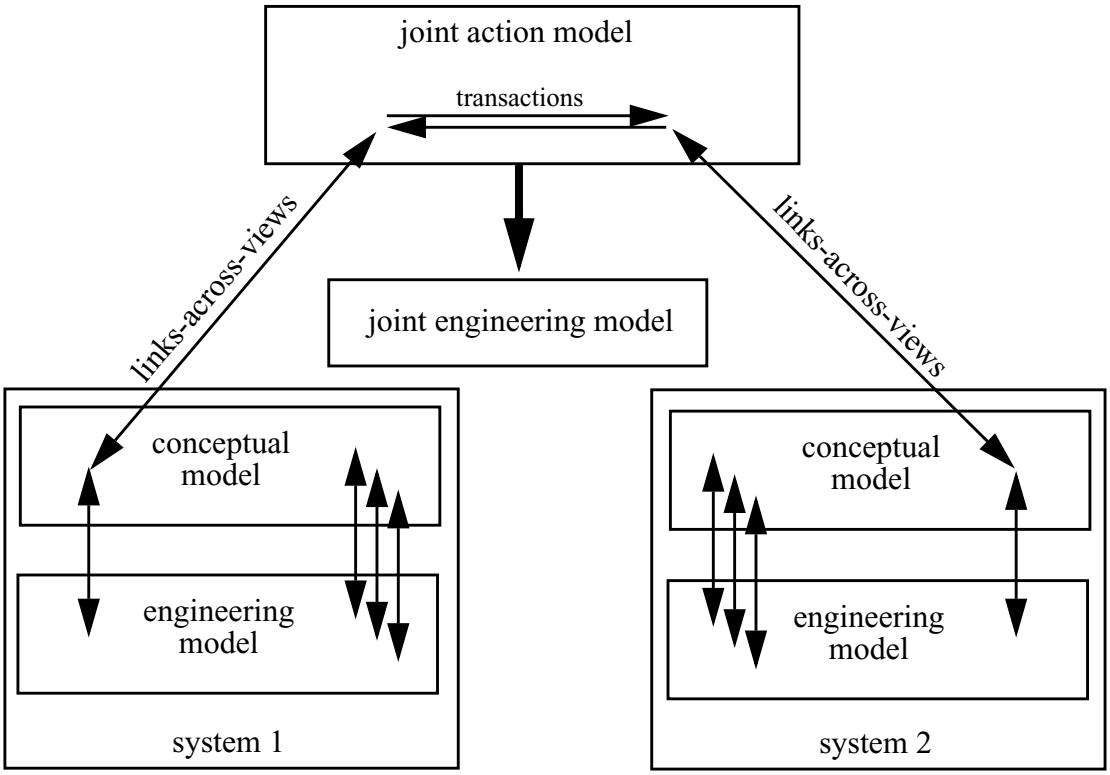

Figure 4. Concepts in the JAM are linked to concepts in each LIM, ultimately linking entities in local engineering models to a new joint engineering model.

- a toolkit of relevant software components must be provided, and
- a characterization of that toolkit must also be formalized.

Generation of message converters is then reduced to a search problem: find the composition of available components that can transform the input available into the desired output. Although by no means necessary, optional optimization would also help. There are two significantly different engineering problems here: Conversions of messages and message elements, and dealing with differences in the actual communications protocols (e.g. Java vs. .Net).

How The Pieces Fit Together – An Example Problem

This section steps through an example problem to illustrate how the research elements described previously can be used to automate an integration task. The example scenario is a Request for Quotation and Quotation response between a customer using CIDX (Chemical Industry Data Exchange Specification) and a supplier using OAGIS (Open Applications Group Integration Specification).[3][10]

Step 1: Specify the business interaction.

This is a specification of system requirements in business terms. The form that it takes is not important. It is expected that this step is performed by humans.

Step 2: Abstract and formalize the relevant concepts into a Joint Action Model.

Figure 6 illustrates the envisioned transactions as:

1. The customer issues a ‘request for quote’.
2. A Supplier responds with a ‘quote’.

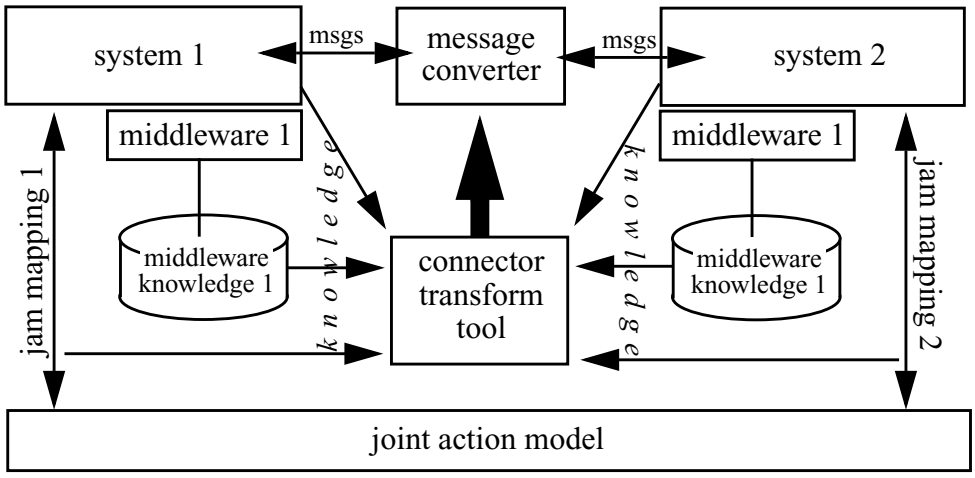

Figure 5. Once formalized, message converters are automatically created by the connector transform tool to map messages between the different systems. (See figure 10 for more detail.)

The concepts for this joint action must be formalized into the JAM.

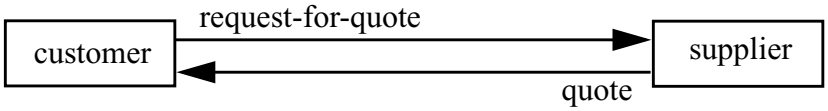

Figure 6. An example transaction that must be formalized by the JAM.

Step 3: Read the specifications.

In our example scenario, the customer is using CIDX middleware. Figure 7 shows an extract from the CIDX specification. This specification shows how the customer (Buyer) expects to interact with a seller.

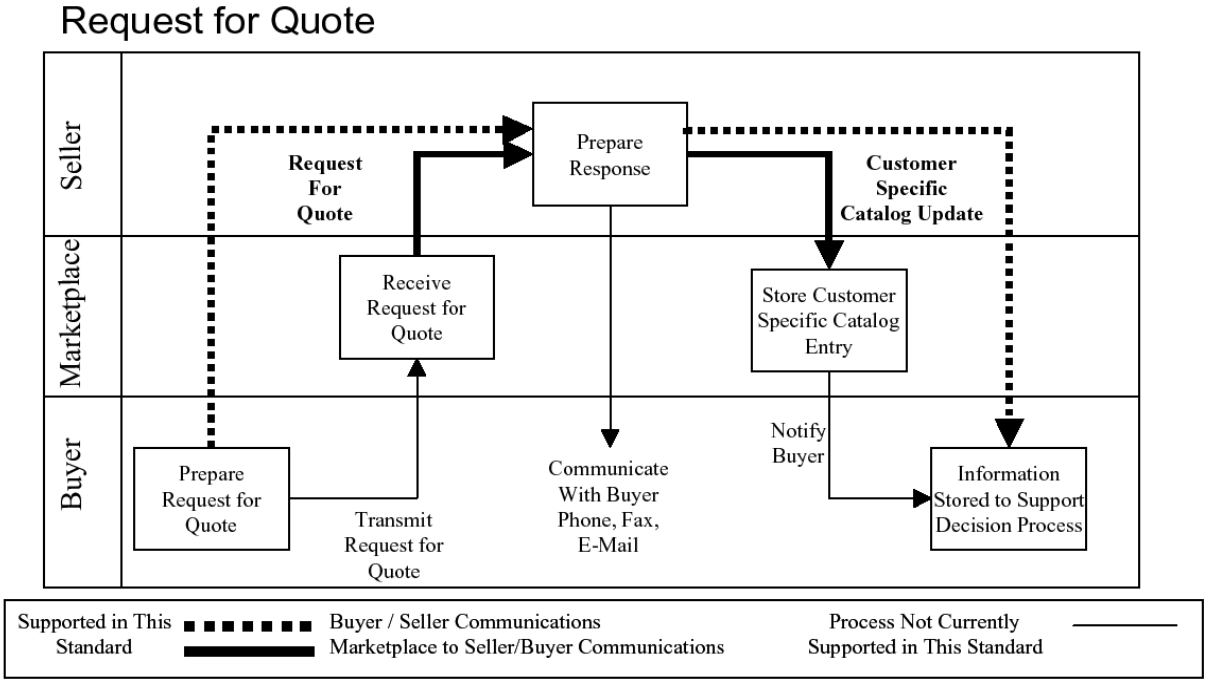

Figure 7. An example of CIDX specifications that need to be interpreted.

Step 4: Abstract and formalize the relevant concepts from the specifications.

Formulate two LIMs – one for the customer and one for the supplier. Figure 8 shows the conceptualization of such an abstraction for our CIDX customer.

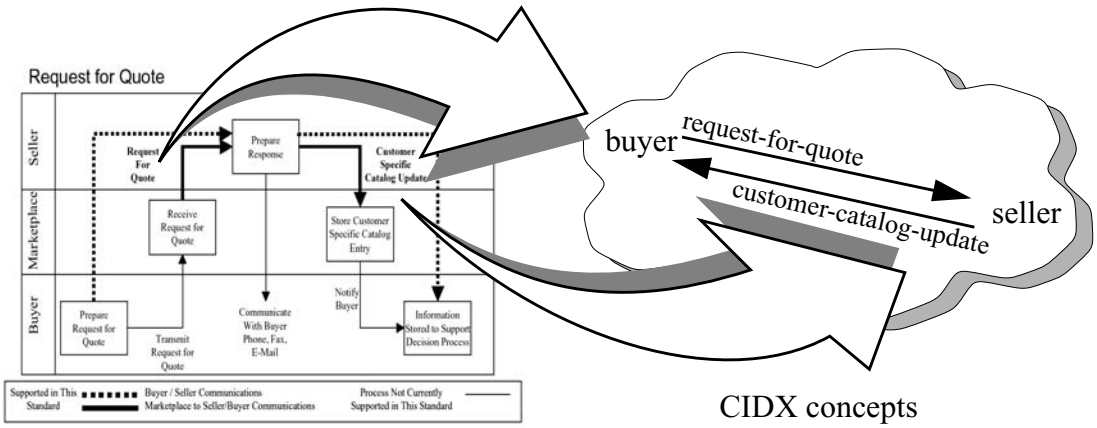

Figure 8. CIDX Concepts Abstraction – The LIM relates the conceptual model to the engineering model. (See Figure 7 for a larger view of the same CIDX specification.)

Step 5: Perform the semantic analysis.

Map the customer's view of how it interacts with the world (from its Local Interaction Model) into the interactions defined for its role for the joint action – these are in the Joint Action Model. Do the same for the supplier using its Local Interaction Model.

Derive or extract the semantic relationships among the models. For our scenario, the excerpt below is from the set of equivalent terms. The OAGIS term 'Quote' and the JAM's term 'Quote' are equivalent, as are the CIDX term 'CustomerSpecificCatalog.lot-price' with the Local Interaction Model's term 'Quote.total-price'.

```
TermEqv (OAGIS::Quote, IO::Quote)
TermEqv (CIDX::CustomerSpecificCatalog.lot-price, IO::Quote.total-price)
```

Step 6: Derive the semantic message map.

From the semantic relations determined in Step 5, we can derive a semantic message map (figure 9).

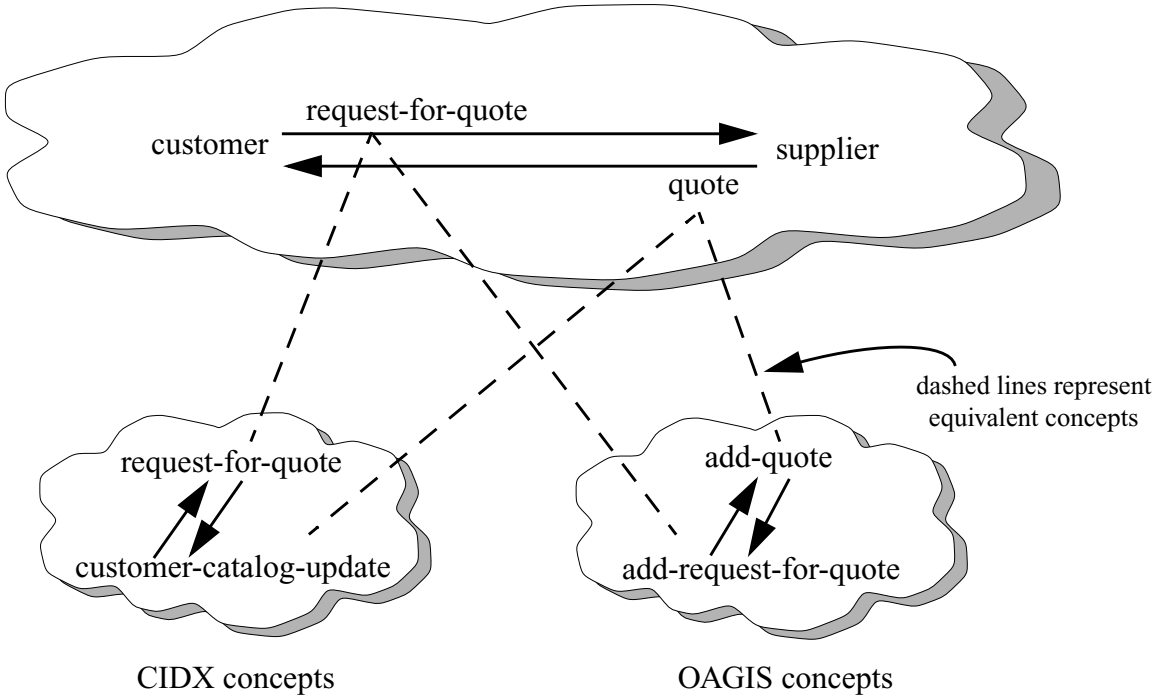

Figure 9. Semantic message maps are derived from the semantic analysis.

Step 7: Transform connector

By now, we should have the semantic map that identifies which data fields are equivalent (potentially in a computable sense). The next step is to use that map, along with formal characterizations of the fields themselves, to identify the adaptations that are needed. With semantic mapping to a sufficient level of detail, a message converter could be produced (figure 10) that converts messages during the joint action for the customer and supplier. A knowledge base for ebXML, the middleware in this scenario, must be developed to provide information about the middleware to the connector transform tool.

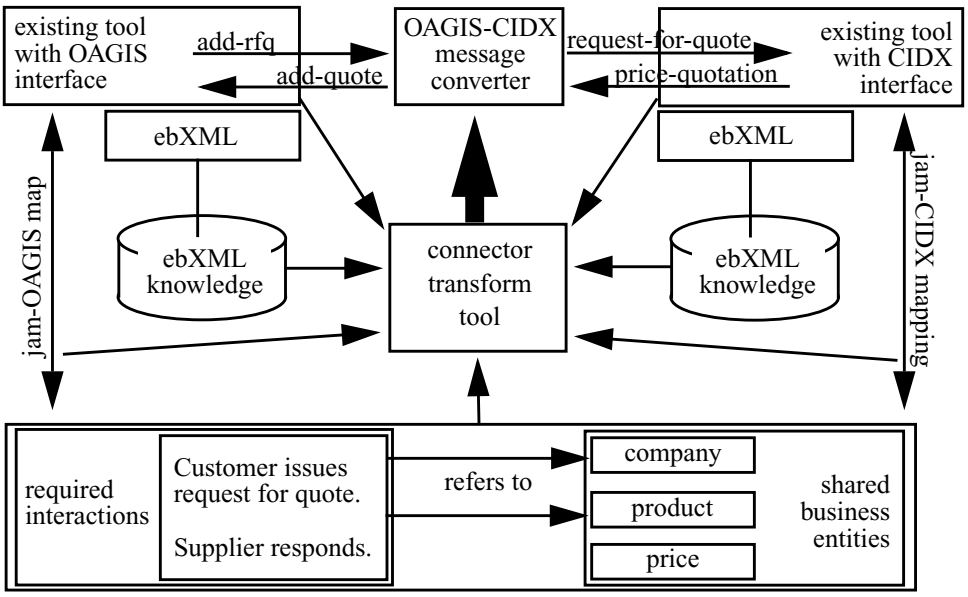

Figure 10. Using knowledge from the JAM and the application-specific ebXML middleware, a connector transform tool produces a converter to convert the OAGIS and CIDX messages in the transaction. (See figure 5 for more detail.)

The connector transform tool will not normally have a display. It will be non-interactive, producing as output the message converter. However, to make it easier to see the sorts of things it will do, we have produced an interactive demonstration. Figure 11 shows a visualization of what the connector transform tool will do at the lowest level of abstraction, where individual pieces of information are transformed.

The resulting experience is not entirely unlike what commercial Enterprise Application Integration (EAI) tools are already doing. The difference is that this project is hoping to automate out the interaction. Software performing automated systems integration does not exist today.

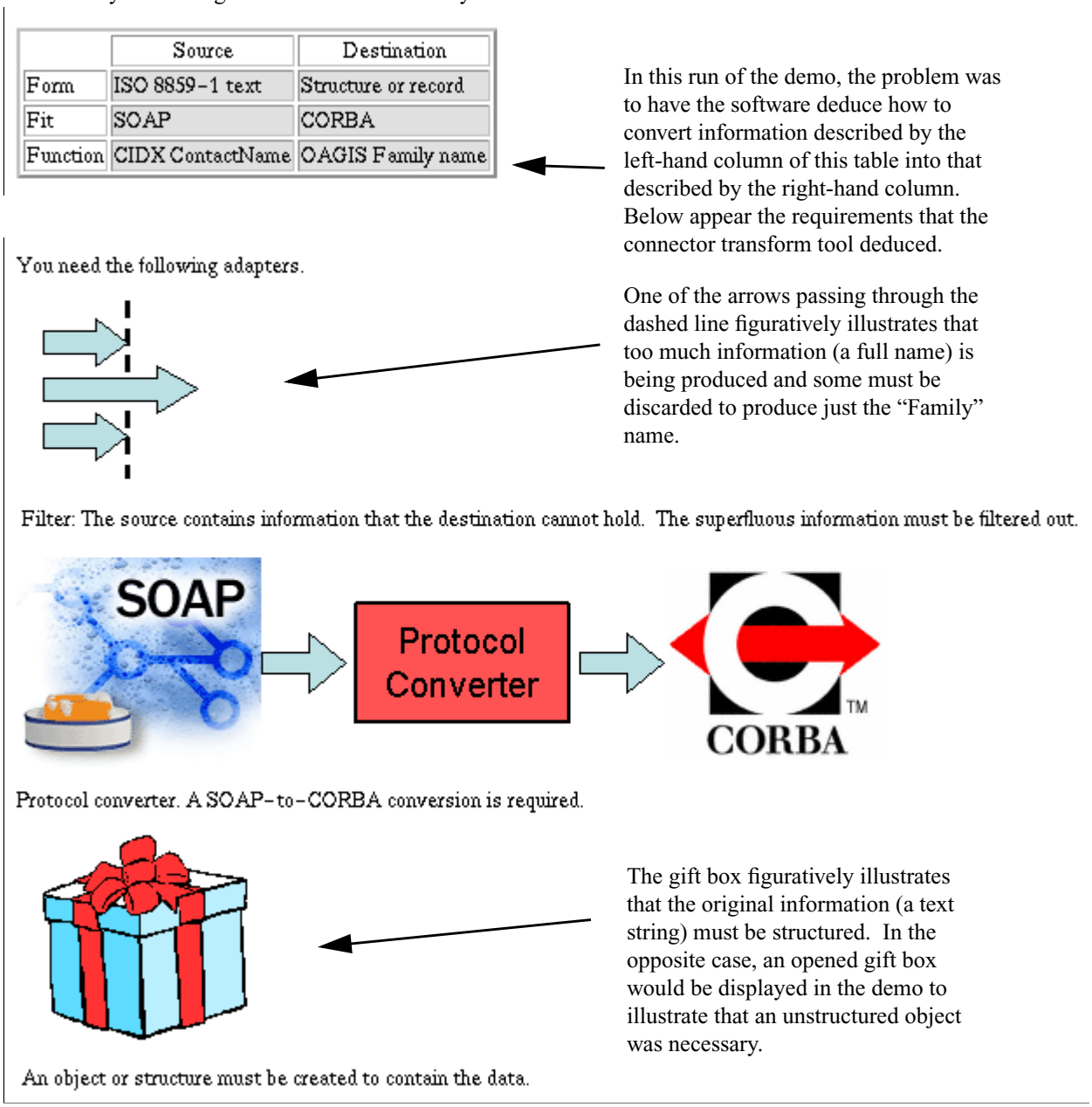

Figure 11. Screen snapshot showing output from connector transform tool demonstrating conversion between string representation of the CIDX ContactName using SOAP and a structure of the OAGIS FamilyName using CORBA.

Possible Problems and Other Thoughts

The AMIS approach is by no means a panacea. For example, it is not applicable to a large class of integration problems. In controlled environments, it is possible to enforce a common ontology, common interfaces, and enterprise-wide modern modeling practices that make subsequent integrations relatively simple.

The applicability of AMIS is in relatively uncontrolled environments. However, by this very nature, the large number of possibilities presents problems. For example, AMIS hypothesizes the existence of software and algorithms that may be untenably difficult to create or intractably expensive to run. For instance, the connector transformation problem is NP-hard in the general case. So either the problem space must be constrained or heuristics must be required, but the latter foregoes assurances of optimality and, in some cases, correctness. [6]

Equally difficult problems are faced in the development of the LIMs for each system to be integrated. Systems may have no existing LIM in any sense that qualifies for machine reasoning. Deriving such LIMs requires reverse engineering, a potentially difficult task. Engineering models may exist, but not to the degree required by AMIS. Alternatively, engineering models may exist but be out-of-date because of, for example, ad hoc undocumented changes. In addition, LIMs may fail to model information that is inconsequential to the individual system internally but is of crucial importance to multi-system integration.

Finally, there is a trade-off between the resources required by the AMIS approach and taking a one-off approach. In the former case, resources may be consumed in the belief or possibility that models will have substantial future value. Whether this is true cannot always be predicted and if incorrect, means potentially large unnecessary expenditures. Of course, the opposite approach has its own pitfalls – integration projects involving one-off types of systems are frequently burdened with costs that would have been unnecessary had they been properly modelled at the outset.

Because of these and other issues, it may be readily evident at the extremes whether to apply the AMIS ideas. However, there is a large class of integration problems in the middle of the spectrum for which it will not be at all clear.

Comparisons

It is worth comparing the AMIS approach to other approaches.

The closest similarity is the approach that uses Enterprise Application Integration (EAI) tooling. [13] Such tooling assists the engineer in developing the conceptual model for the interaction, usually by tailoring an existing conceptual model provided by the EAI vendor, and often linked to the generalized models of a particular major system. The tooling then walks the engineer through the process of developing the local interaction models for the component systems (or for all the components other than the target major system) and the linkages to the joint action model. In many cases, the EAI vendor has pre-developed LIMs for commonly used component systems, with corresponding translators. That is, the EAI tooling assists the engineer in developing the joint action model and perhaps some of the LIMs. Off the shelf EAI tooling is provided or modified to generate the message-specific integrating code.

The difference is that most EAI tooling takes no advantage of existing models other than those provided by the EAI vendor; although it does create and archive the component models and interaction models developed under its tutelage. However, in EAI, the critical semantic mappings have been pre-developed by the EAI vendor, and the engineer is required to map his “joint action” to that model. The AMIS approach advocates the development and maintenance of such models during construction and acquisition of the individual systems, thereby formalizing the LIM as understood by the developers. Then the integration can be performed with automated tools using models that already exist.

Other approaches are even less automated and rely heavily on textual representations or non-automated organizations that require humans to bear the burden of repeatedly recognizing and mapping concepts that relate.

Conclusions

This paper describes the AMIS approach for automation in software integration. Three major areas of work are anticipated and described. Local Interaction Model formation is concerned with capturing the business and engineering interaction concerns for existing systems in a form suitable for reasoning. Semantic mapping pertains to building tools to create semantic maps between conceptual models. Connector transformation is concerned with creating generators for dynamic message and protocol converters. Efforts in each of these areas must come together to support automation in integration activities.

This project poses many challenges. They include:

- constructing a useful knowledge framework for joint action models
- developing joint action models from specifications as received
- reverse engineering to produce local interaction models
- defining semantic mapping algorithms
- building a sufficient knowledge base for connector transformations
- automating analysis and resolution of technical mismatches

We expect the outcome of this work to have both direct benefits and indirect benefits. The direct benefits are clear: automated integration which could greatly reduce the time and cost of systems integration projects. In the future, it is possible that more difficult integration projects could be attempted that would not otherwise be attempted today with current technology. The indirect benefits are less so but significant nonetheless and include improved interface/service specifications and improved knowledge capture for existing software systems and standards. Lastly, it is inevitable that the AMIS research will provide knowledge for new toolkits as well as identify more significant unsolved problems for future work.

References

1. Aberdeen Group, http://www.aberdeen.com, June 2001.

2. Edward Barkmeyer, Allison Barnard Feeney, Peter Denno, David Flater, Don Libes, Michelle Steves, Evan Wallace, "Concepts for Automating Systems Integration," National Institute of Standards and Technologies, Gaithersburg, MD, NISTIR 6928, http://www.nist.gov/msidlibrary/doc/AMIS-Concepts.pdf, February 2003.

3. CIDX, "CIDX Overview," http://www.cidx.org/AboutCIDX/CIDXOverview.asp?Level=2&SecondLevelURL1=/AboutCIDX/AboutCIDX.asp, 2003.

4. David Connelly, "Plug and Play Business Software Integration: The Compelling Value of the Open Applications Group," http://www.openapplications.org/downloads/whitepapers/whitepaperdocs/whitepaper.htm, 2000.

5. Peter Denno, Michelle Steves, Don Libes, and Edward Barkmeyer, "Model-driven Integration Using Legacy Models," IEEE Software, 2003.

6. David Flater, "Automated Composition of Conversion Software", in preparation.

7. Sam Gentile, "Beyond (COM) Add Reference: Has Anyone Seen the Bridge," http://msdn.microsoft.com/netframework/?pull=/library/en-us/dndotnet/html/bridge.asp, October 2003.

8. Carol Geyer, "ebXML Messaging Service Specification Approved As OASIS Standard," http://www.oasis-open.org/news/oasis_news_09_05_02.php, September 2002.

9. Don Libes, David Flater, Evan Wallace, Michelle Steves, and Edward Barkmeyer, "Challenges of Automated Integration," IASTED, Innsbruck, Austria, February 2004.

10. Open Applications Group, http://www.openapplications.org, 2003.

11. Object Management Group, *OMG's CORBA (Common Object Request Broker Architecture) Website*, http://www.corba.org, 1997-2003.

12. Uche Ogbuji, "The Past, Present and Future of Web Services, part 1," http://www.webservices.org/index.php/article/articleview/663/1/61/, September 2002.

13. William Ruh et al, "Enterprise Application Integration: A Wiley Tech Brief," John Wiley & Sons, October 2000.

14. John Schettino et al., "CORBA For Dummies," John Wiley & Sons, October 1998.