

# Use of Object Oriented Model For Interoperability in Wrapper-Based Translator for Resolving Representational Differences between Heterogeneous Systems<sup>+</sup>

Paul Young, Valdis Berzins, Jun Ge, Luqi

Department of Computer Science  
Naval Postgraduate School  
Monterey, California 93943, USA

Email: {young, berzins, gejun, luqi}@cs.nps.navy.mil

## ABSTRACT

One of the major concerns in the study of software interoperability is the inconsistent representation of the same real world entity in various legacy software products. This paper proposes an object-oriented model to provide the architecture to consolidate two legacy schemas in order that corresponding systems may share attributes and methods through use of an automated translator. A Federation Interoperability Object Model (FIOM) is built to capture the information and operations shared between different systems. An automatic translator generator is discussed that utilizes the model to resolve data representation and operation implementation differences between heterogeneous distributed systems.

Key words: interoperability, object-oriented model, federation interoperability object model, wrapper

## 1. INTRODUCTION

In contemporary object-oriented modeling, an object is a software representation of some real-world entity in the problem domain. An object has identity (i.e., it can be distinguished from other objects by a unique identifier of some kind), state (data associated with it), and behavior (things you can do to the object or that it can do to other objects). In the Unified Modeling Language (UML) these characteristics are captured in the name, attributes, and operations of the object, respectively. UML distinguishes an individual object from a set of objects that share the same attributes, operations, relationships, and semantics—termed a *class* in the UML. [BRJ99]

This view of objects and classes has proven valuable in the development of countless systems in various problem domains encompassing all degrees of size and complexity. However, one common characteristic of the

majority of these object-oriented developments is that a development team that shared common objectives and had a common view of the real-world entities being modeled produced them. Often, the developments also involved a common architecture implemented on a common target platform, using the same implementation language and operating system. As a result a single method of representation of an entity's name, attributes, and operations is the norm. Even on heterogeneous implementations by the same development team, consistency in the names, attributes and operations used for the same real-world entity is likely across the various elements of the architecture. Therefore, capturing the representation of these properties has not been an issue. The software representation of the real-world entity should have the same name, attributes, and operations across all elements of the architecture if the development team enforces consistency.

This is not necessarily the case when independently developed, heterogeneous systems are targeted for integration and interoperation. The different perspectives of the real-world entity being modeled by independent development teams will most likely result in the use of different class names as well as differences in the number, definition, and representation of attributes and operations for the same real-world entity implemented on two or more different systems. It is the same situation for non-object-oriented fashioned systems. These differences in representation of the same real-world entity on different systems must be reconciled if the systems are to interoperate.

This paper proposes an object-oriented model for defining the information and operations shared between systems. The initial use of the model is targeted for integration of legacy systems, which generally have not been developed using the object-oriented paradigm. Defining the

<sup>+</sup> This research was supported in part by the U. S. Army Research Office under contract/grant number 35037-MA and 40473-MA.

interoperation between systems in terms of an object model however, provides benefits in terms of the visibility and understandability of the shared information and provides a foundation for easy extension as new systems are added to an existing federation. The object model defined in this paper can be easily constructed from the external interfaces defined for most legacy systems (whether object-oriented or not).

Section 2 will introduce the object-oriented model for interoperability (OOMI) and its structure. In Section 3, an interoperability object model is defined for a specified federation of systems. Section 4 presents an overview of the use of the Federation Interoperability Object Model (FIOM) by a wrapper-based translator for enabling general solution to construct the wrapper architecture interoperability among legacy systems.

## 2. OBJECT-ORIENTED MODEL FOR INTEROPERABILITY

An extension of the contemporary object model class diagram, depicted in Figure 1, is proposed to model the different possible ways an object might be represented in a federation of independently developed heterogeneous systems. The proposed extension includes information about the different representations that an object's attributes and operations may take in different systems in the federation.

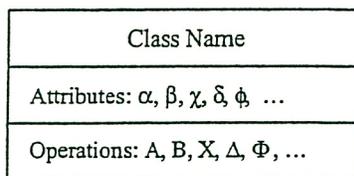


Figure 1. Contemporary Object Model for Each System

This alternative object model includes the following extensions to the contemporary object model. First, as depicted in Figure 2, the object oriented model for interoperability (OOMI) class diagram will contain a representative of all attributes included in any defined representation of the real-world entity modeled by that class. In Figure 2 these are depicted as attributes  $\alpha$  through  $\phi$ . Each attribute may have multiple representations, resulting from differences in interpretation by the component system design teams. From Figure 2, each of attributes  $\alpha$  through  $\phi$  has  $n$  representations, labeled  $\alpha_{R1}$  through  $\alpha_{Rn}$  for attribute  $\alpha$ , and similarly for each of the other attributes. A standard representation for each attribute is also included, labeled  $\alpha_{STD}$  for attribute  $\alpha$  in Figure 2. The standard representation is chosen by the interoperability designer as an intermediate representation to be used during

translation.

For each attribute representation, the interoperability object model class diagram will contain information used in establishing that the different representations refer to a common characteristic of the real-world entity being described. This includes information about both the syntax of the attribute (attribute type, structure, size, etc.) and the semantics of the attribute (attribute role, description, etc.). This information is depicted for attribute  $\alpha$  representation 1 in Figure 2 as  $\alpha_{R1} Syntax$  and  $\alpha_{R1} Semantics$ , respectively.

In addition, the model will contain one or more translations required to convert between different representations of that attribute. These translations can be defined on a pair-wise basis for all possible representations- requiring  $n(n-1)$  translations for  $n$  different representations. Alternatively, they can be defined using the standard representation as an intermediate representation and translation performed in two steps (representation 1 to standard to representation 2), requiring  $2n$  translations. The two-step translation method is depicted in Figure 2, with translation  $\alpha_{R1} ToSTD()$  defined to translate an instance of attribute  $\alpha$  from representation 1 to the standard representation, and translation  $STD To\alpha_{R1}()$  defined to translate an instance of the standard representation of attribute  $\alpha$  to representation 2.

Similarly, the interoperability object model class diagram extends the contemporary object model class diagram to include information about different possible implementations for each operation. Implementation differences may include differences in operation and parameter naming, differences in the number and type of parameters invoked by the operations, and differences in the internal algorithms used by each operation. As long as the dynamic behavior of the two implementations is equivalent for the same input and output conditions, they can be used interchangeably. Thus, the OOMI class diagram includes information necessary to determine if different implementations of an operation are inter-accessible. This includes information about both the syntax of the operation (naming, parameters, etc.) and the semantics of the operation (operation role, behavior, description, etc.). In addition, for each operation, the model will contain one or more translations required to account for operation name and parameter variations found in different operation implementations. Figure 3 illustrates the operation extension provided in the OOMI class diagram.

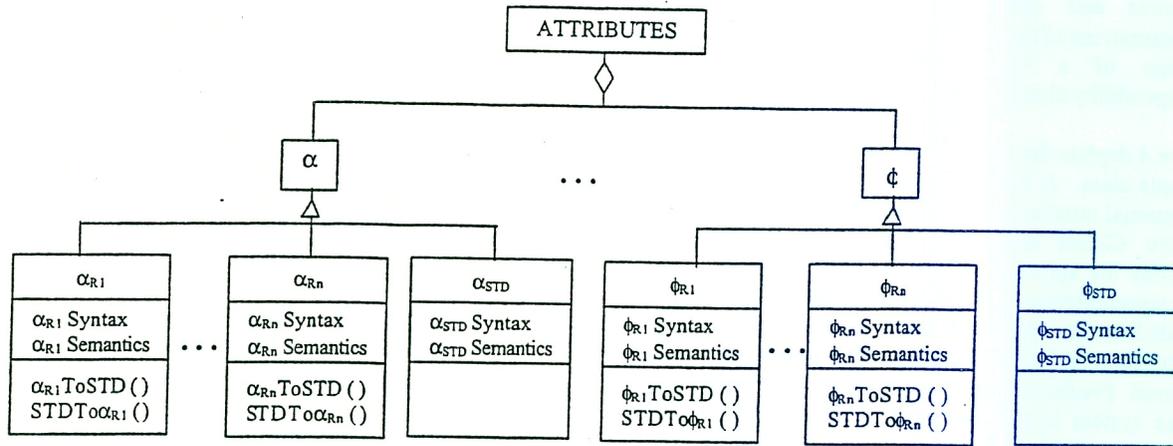


Figure 2. OOMI Class Diagram Attribute Extension

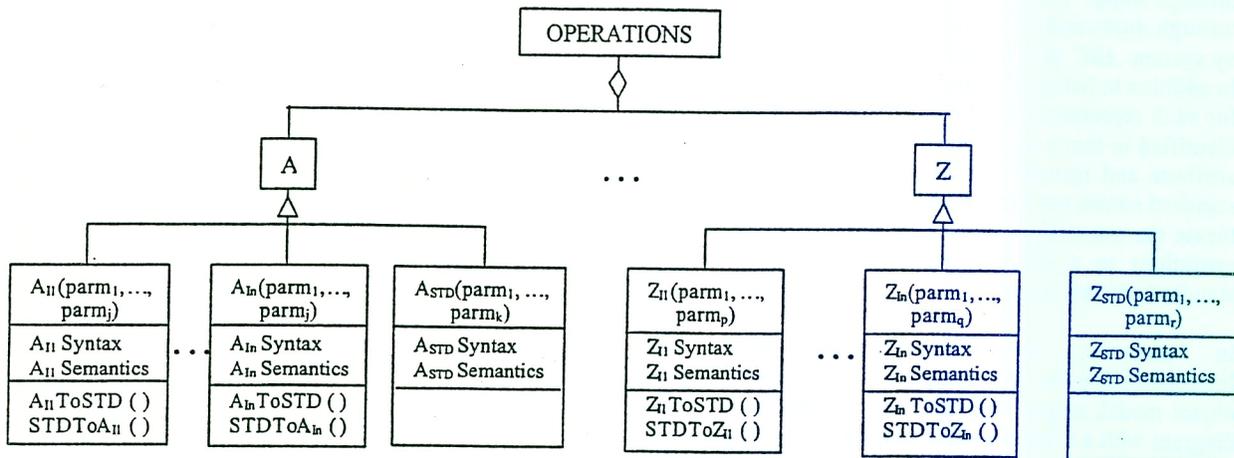


Figure 3. OOMI Class Diagram Operation Extension

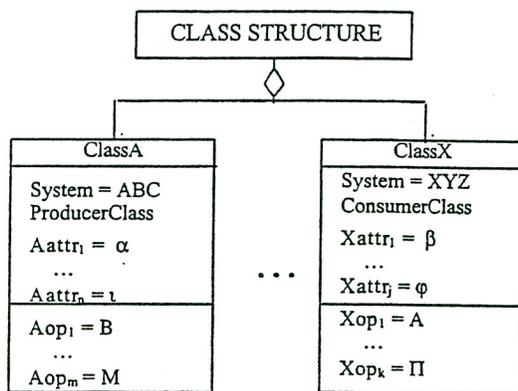


Figure 4. OOMI Class Diagram Class Structure

From Figure 3, it can be seen that the depicted class diagram contains operations A through Z and that each operation has a number of different implementations. For example, operation Z has implementations  $Z_{I1}$  through  $Z_{In}$ , each with a potentially different set of parameters. For

each operation, the interoperability designer defines a standard implementation for that operation which is used as an intermediate representation during translation. For each implementation syntactic and semantic information is provided in order to establish a correspondence with other operation implementations that are equivalent-for example  $Z_{In}$  Syntax and  $Z_{In}$  Semantics for operation Z implementation n. Finally, translations  $Z_{In}ToSTD()$  and  $STDToZ_{In}()$  are used to translate operation and parameter names from operation Z implementation n to the standard representation for operation Z's name and parameters, and vice versa.

In addition to having different representations for the same attribute or different implementations for an operation, heterogeneous object designers may provide different numbers and types of attributes and operations for the same real-world entity. One representation of that real-world entity might include attributes and operations that another representation omits. Because of this difference, a mechanism must be provided to capture the

attributes and operations present in the various representations of the entity. This is provided through the addition of a Class Structure property to the interoperability object model class diagram.

Figure 4 depicts the OOMI class structure property for an example class. A representation of this class is found in the external interface of a number of systems, as specified by the *ClassA* through *ClassX* class diagrams that comprise the aggregate Class Structure property. For each representation, a list of the attributes and operations included in that representation is included. In addition, the system of origin of the class and whether the class is exported (*ProducerClass*) or imported (*ConsumerClass*) by the system is also included in the class's attribute property. As indicated in Figure 4, *ClassA* contains attributes *Aattr<sub>1</sub>* through *Aattr<sub>n</sub>*, and operations *Aop<sub>1</sub>* through *Aop<sub>m</sub>*. Attribute and operation names for *Aattr<sub>1</sub>* through *Aattr<sub>n</sub>* and *Aop<sub>1</sub>* through *Aop<sub>m</sub>* are the names used by system *ABC* as contained in *ABC*'s external interface. In addition to listing the attributes and operations included for each representation, the attributes and operations are identified in terms of the standard names provided in the attribute and operation properties of the class. These standard names are used together with the local names to locate the translations used to convert the attributes and operations to a different representation (to or from a standard representation).

In summary, the Object-Oriented Model for Interoperability is an extension of the contemporary object model, augmenting the contemporary model class diagram with a Class Structure property and extending the Attribute and Operation properties to capture the different representations possible for those properties in a federation of autonomous heterogeneous systems. The model is extensible in that adding new representations for an attribute or operation or for a class merely adds a class to the existing properties while preserving the existing representations. The model increases the level of abstraction dealt with by the interoperability engineer by enabling him to think in terms of the real-world entities participating in the interoperation between systems and not in terms of the different representations used. And finally, by capturing the information needed to represent the relationships between entity representations and the translators necessary to convert between representations, the OOMI supports automated conversion between object representations. Figure 5 provides a top-level summary of the proposed OOMI Class Diagram.

<<Interoperability Class>>	
Name	
Class Structure	(Figure 4)
Extended Attributes	(Figure 2)
Extended Operations	(Figure 3)

Figure 5. OOMI Class Diagram

### 3. CONSTRUCTING INTEROPERABILITY OBJECT MODEL FOR FEDERATION OF HETEROGENEOUS SYSTEMS

The previously introduced Object Oriented Model for Interoperability enables information sharing and cooperative task execution among a federation of autonomously developed heterogeneous systems. Using the information contained in the OOMI class diagrams computer aid can be applied to the resolution of data representational differences between heterogeneous systems. In order to apply computer aid, a model of the real-world entities involved in the interoperation, termed a Federation Interoperability Object Model (FIOM), is constructed for the specified system federation. Construction of the FIOM is done prior to run-time by a system designer with the assistance of a specialized toolset, called the Object Oriented Model for Interoperability Integrated Development Environment (OOMI IDE).

The process of constructing a FIOM for a specified system federation essentially consists of identifying the real-world entities that reflect the shared information and tasks and capturing the different representations used by systems in the federation for that entity. Each real-world entity is represented in the FIOM as a class, termed a Federation Interoperability Class (FIC), constructed from the classes contained in the component systems' external interface.

Determination of the real-world entities that define the interoperation of a federation is not merely a matter of identifying the classes involved in the external interfaces of the systems in the federation. Because of the independently developed, heterogeneous nature of the systems in the federation, each system may have a different representation for the real-world entities involved. Thus, the classes and objects that realize the external interfaces of the component systems must be correlated to determine which representations reflect the same real-world entity. Correlation software is included as part of the OOMI IDE in order to assist the system designer by providing a small set of selected correspondences to be reviewed by domain experts.

#### 4. AUTOMATIC WRAPPER GENERATION

System interoperability involves both the capability to exchange information between systems and the ability for joint task execution among different systems. [PIT97] Both capabilities involve one or more of the following kinds of actions:

- *Send* One system transmits a piece of information to another
- *Call* One system invokes an operation on another
- *Return* Returns a value to the caller
- *Create* Creates an object on the called system
- *Destroy* Destroys an object on the called system [BRJ99]

Information exchange is accomplished through means of a *Send* operation, where one system, the producer, exports information that another system, the consumer, imports. Information transmitted by the producer system can be in the form of an object of some class defined for the producer, or it can consist of one or more attributes of an object defined for the producer.

Joint task execution is accomplished through the use of a *Call* operation where one system, a client, invokes an operation on another, acting as a server for the requested action. In invoking an operation on a server, a client system must provide the name of the operation requested as well as any parameters required by the server to perform the operation. Required parameters can be in the form of one or more attributes, operations, or objects. In addition, in response to a client *Call* operation, a server may return a set of attributes, operations, or objects to a client via a *Return* operation. *Create* and *Destroy* actions are special instances of a system call.

When information exchange or joint task execution is performed between heterogeneous systems, the participating systems must account for differences in representation of the transmitted information. The Interoperability Object Model constructed during the pre-runtime phase for a specified federation of component systems is used to resolve differences in representation between interoperating systems. A *translator* that serves as an intermediary between component systems accomplishes representational difference reconciliation at runtime.

The translation function is anticipated to be implemented as part of a *software wrapper* enveloping a producer or consumer system (or both) in a message-based architecture, or alternatively as part of the data store (actual or virtual) in a publish/subscribe architecture. A software wrapper is a piece of software used to alter the view provided by a component's external interface

without modifying the underlying component code. Figure 6 presents an overview of the use of software wrappers and the involvement of the Federation Interoperability Object Model in the translation process.

The translations required by the wrapper-resident translator for both information exchange and joint task execution are similar. For information exchange, the source system provides the exported information in the form of a set of attributes or objects of a producer class in the native format of the producer. In order to be utilized by a consumer system, the exported information must be converted into the format expected by the destination system. For joint task execution, a client system provides an operation name and a set of parameter values to a server system in the native format of the producer. The parameters may be attributes, operations, or objects of a client class. Again, this information must be provided to the destination system in a format recognized by that system. Thus the operation name, operations, and parameter values must be converted to the server representation.

As indicated above, the translator must be capable of converting instances of a class's attributes and operations (or both attributes and operations in the form of an object of the class) from one representation to another. The information required to effect these translations is captured as part of the Interoperability Object Model for a specified system federation during federation design. As presented in Figures 3 and 4, each attribute and operation of a class representing a real-world entity defining the interoperation includes methods to enable the translation between attribute and operation representations. Then, at run time, the translator accesses the information contained in the model to effect the translation between representations.

The first action the translator must perform is to determine the class defining the real-world entity corresponding to a transmitted object, attribute, or operation. This can be accomplished through the use of a mapping developed from the FIOM that maps attribute, operation, or object representations to the class representing the corresponding real-world entity in the model. For instance, from the example provided in the previous section, objects of class *ClassA* and *ConsumerX* as well as the attributes and operations for these classes would map to a real-world entity represented by prototypical class instance *RealWorldEntityA*. Once the class corresponding to the transmitted object, attribute, or operation is determined, the methods defined for each attribute and operation can be used to effect the translation between representations.

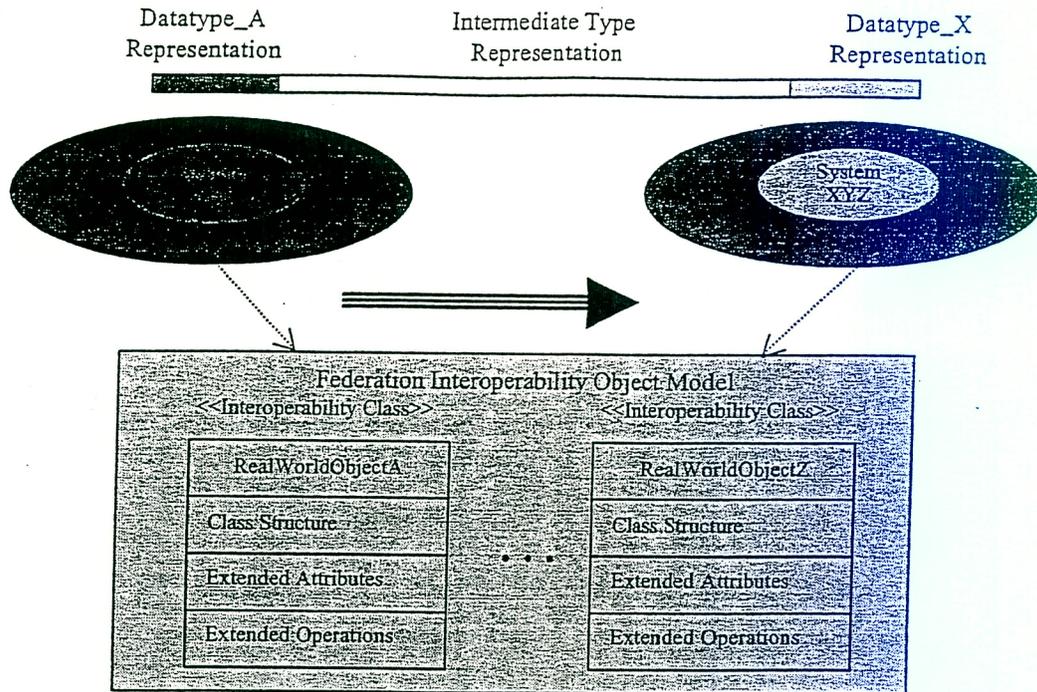


Figure 6. FIOM in Automatic Wrapper Generation

If the transmitted entity were set of attributes, such as would be the case during information exchange, then for each attribute in the set the appropriate translation method must be selected. The appropriate translation method is located by using the Class Structure property to determine the standard representation for each attribute and then finding the translations for that attribute in the Attributes property for the class representing the real-world entity. The translation provided would either be in terms of a source-to-destination or a source-to-intermediate representation conversion depending on the approach used by the system designer for the federation. In this manner the translator invokes the appropriate translation method for each attribute to convert the attribute from the source system representation to either the destination system or intermediate representation. The translated attribute set is then forwarded to the destination system for appropriate disposition. If an intermediate representation is used in the translation process, this process is repeated by the destination system to convert from the intermediate to destination system representation.

For instance, continuing our example from the previous section, suppose System ABC were to transmit the attributes  $Aattr_1$  and  $Aattr_2$  from class *ClassA* to System XYZ. Then presuming that the representation used for System ABC is not useable by System XYZ,  $Aattr_1$  and  $Aattr_2$  must be translated to a form useable by System XYZ. For our example a wrapper-based translator on Systems ABC and XYZ will conduct the translation with

the translation performed in two steps using an intermediate representation of the real-world entity's attributes.

As depicted in Figure 7 below, the System ABC wrapper would intercept the transmitted attributes from System ABC. Then, using the mapping outlined above, the wrapper-based translator would first determine that the intercepted attributes were of class *ClassA* that corresponds to class *RealWorldEntityA* representing the real-world entity participating in the interoperation. Then, for each attribute, the appropriate translation method must be determined. This translation method can be determined from the Attributes property, given the standard representation for the attribute. From *RealWorldEntityA*'s Class Structure property (see Figure 4), it is determined that *ClassA* attribute  $Aattr_1$  corresponds to *RealWorldEntityA*'s type *Attribute\_α* and  $Aattr_2$  corresponds to type *Attribute\_β*. The appropriate translation method is then selected *Attribute\_α* translation 1 ( $Aattr_1ToSTD()$ ) for *ProducerA* attribute  $Aattr_1$  and *Attribute\_β* translation 1 ( $Aattr_2ToSTD()$ ) for *ProducerA* attribute  $Aattr_2$ . The translator would apply these translation methods to each attribute as appropriate and forward the resultant intermediate representation to System XYZ.

The System XYZ wrapper would intercept the incoming transmission and repeat the process outlined above to convert the attributes from their intermediate representation to the *ConsumerX* representation as

depicted in Figure 7. The resultant translated attributes would then be forwarded to System *XYZ* for disposition.

If the transmitted entity is an operation with a set of parameters, such as would be the case during joint task execution, then the translator must enable conversion of both the operation name and parameters and translation methods for both operation name and parameter set must be selected. The appropriate translation method for converting the operation name is located by using the Class Structure property to determine the standard representation for the operation name and then finding the translations for that operation name in the Operations property for the class representing the real-world entity. The translation provided would either be in terms of a source-to-destination or a source-to-intermediate representation conversion depending on the approach used by the system designer for the federation. The translator would then invoke the appropriate translation method for the operation to convert the operation name from the source system representation to either the destination system or intermediate representation.

Operation parameters can either be attributes, objects, operations, or their combinations. For attribute parameters, translation of each attribute is conducted as described in the attribute translation process above. Translation of object parameters will be discussed in the next paragraph. Operation parameter translation would involve both operation name and parameter translation as described above. The translated operation name and parameter list is then forwarded to the destination system for appropriate disposition. As described above for attribute translation, if an intermediate representation is used in the translation process, this process is repeated by the destination system to convert from the intermediate to destination system representation.

As an example of operation translation, suppose System *ABC* wanted to invoke an operation on System *XYZ* that corresponded to System *ABC* operation *Aop<sub>1</sub>*. Such a situation might arise if operation *Aop<sub>1</sub>* involved a query of system *ABC*'s database and an equivalent operation to find the same information in System *XYZ*'s database was desired. In order for System *ABC* to perform such a task, an equivalent implementation of operation *Aop<sub>1</sub>* must exist on System *XYZ* and any differences in representation between *Aop<sub>1</sub>*'s name and parameters must be resolved for System *XYZ* to be able to execute the operation call. Resolution of representational differences is accomplished by wrapper-based translators on Systems *ABC* and *XYZ* using an intermediate representation of the real-world entity's operations and parameters in a similar manner as was previously done for attributes.

As depicted in Figure 8 below, the System *ABC* wrapper would intercept the transmitted operation from System *ABC*. Then, using the mapping outlined above, the wrapper-based translator would first determine that the intercepted operations were of class *ClassA* that corresponds to class *RealWorldEntityA* representing the real-world entity participating in the interoperation. Then, for each operation name and parameter, the appropriate translation method must be determined. For the operation name, the translation method can be determined from the Operations property, given the standard representation for the operation name. From *RealWorldEntityA*'s Class Structure property (see Figure 3), it is determined that *ClassA* operation *Aop<sub>1</sub>* corresponds to *RealWorldEntityA* *Operation\_B* and operation *Aop<sub>2</sub>* corresponds to *Operation\_A*. The appropriate translation method is then selected- *Operation\_B* translation 1 (*Aop<sub>1</sub>ToSTD()*) for *ProducerA* operation *Aop<sub>1</sub>* and *Operation\_A* translation 1 (*Aop<sub>2</sub>ToSTD()*) for *ProducerA* operation *Aop<sub>2</sub>*.

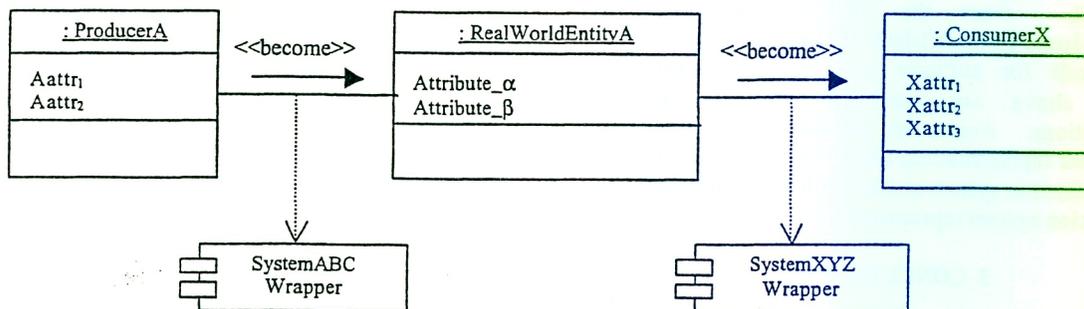


Figure 7. Mapping Translation to Wrapper Architecture

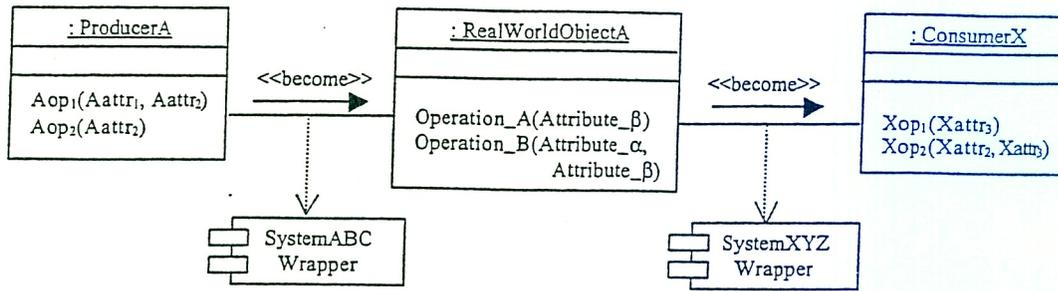


Figure 8. Wrapper-based Translator

In addition to translating the operation name, differences in representation of the operation's parameters must also be resolved. For our example, converting parameter representations would be accomplished in the same manner, as done previously for converting attribute representations. The translator would apply these translation methods to each operation name and parameter as appropriate and forward the resultant intermediate representation for the operation to System XYZ.

The System XYZ wrapper would intercept the incoming transmission and repeat the process outlined above to convert the operation names and parameters from their intermediate representation to the ConsumerX representation as depicted in Figure 8. The resultant translated operations would then be forwarded to System XYZ for disposition.

Translation of object representations involves a combination of the procedures for attribute and operation conversion outlined above. First though, a correspondence between the source and destination object's class attributes and operations must be determined from the Class Structure property. If an intermediate representation is used to effect the translation, the correspondence between the source and intermediate representation of the object's class must be determined. Once the attribute and operation correspondence is established between representations, the methods for attribute and operation translation outlined above are used to convert between representations. Again, for translations involving an intermediate representation, the process must be repeated by the destination system to convert from the intermediate to destination system representations.

## 5. CONCLUSIONS

An Object-Oriented Model for Interoperability (OOMI) is proposed in this paper to solve the data and operation inconsistency problem in legacy systems. A Federation Interoperability Object Model (FIOM) is defined for a specific federation of systems designated for interoperation. The data and operations to be shared

between systems are captured in a number of Federation Interoperability Classes (FICs) used to define the interoperation between legacy systems. Software wrappers are generated according to the FIOM that enable automated translation between different data representations and operation implementations..

At this stage, XML-based message translation is being studied for implementation of the proposed model. The capability provided by the XML family of tools coincides nicely with the requirement for data and operation representation capture and translation.

Some important issues, such as security, real-time, etc., are not discussed in this paper. However, the structure of the semantic and/or syntactic information integrated in the OOMI preserves the capability of being extended to address such concerns.

## REFERENCES

- [BRJ99] Booch, G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley Longman, Inc., Redding, MA, 1998.
- [Pit97] Pitoura, E., "Providing Database Interoperability through Object-Oriented Language Constructs", *Journal of Systems Integration*, Volume 7, No. 2, August 1997, pp. 99-126.