Paper to be published in the Fifth
Annual Express User Group
International Conference (EUG '95)
Grenoble, France, 21-22 October 1995

On Bridging and Extending OMG/IDL and STEP/EXPRESS for Achieving Information Sharing and System Interoperability *

Stanley Y. W. Su, Herman Lam, Tsae-Feng Yu
Sooha Lee, and Javier Arroyo

Database Systems Research and Development Center
University of Florida

On Bridging and Extending OMG/IDL and STEP/EXPRESS for Achieving Information Sharing and System Interoperability *

Stanley Y. W. Su, Herman Lam, Tsae-Feng Yu
Sooha Lee, and Javier Arroyo

Database Systems Research and Development Center
470 CSE Building
University of Florida
P. O. Box 116125
Gainesville, Florida 32611-6125
Email: su@cis.ufl.edu
Telephone: (904) 392-2693
FAX: (904) 392-1220

October 4, 1995

Abstract

This paper describes an integrated information modeling language called NCL which combines the features of IDL and EXPRESS as well as the features of the association types and knowledge rule specifications offered by the Object-oriented Semantic Association Model (OSAM*). In this language, frequently used constraints are specified by keywords, association types among object classes are specified in association specifications, behavioral properties of objects are defined in method specifications, and complex semantic constraints of various types are specified by Event-Condition-Action-AlternativeAction (ECAA) rules. ECAA rules also provide a mechanism for specifying the control relationships among various servers in a heterogeneous computing environment. Keyword constraints and association types are defined in a meta-model (the model of the underlying kernel model of NCL) in terms of parameterized rules which are automatically bound to those classes that use these constraint and association types. At run-time, all rules associated with these classes are used by the rule processor of the KBMS to enforce the the semantics of the constraints and association types. Additional semantic properties found in a heterogeneous environment can be easily introduced into NCL due to the extensibility feature of the language and its supporting system. In this paper, we shall explain how such an enriched language can be used to extend STEP/EXPRESS as an information modeling language for defining product data, workflows and software systems and to extend OMG/IDL as an interface definition language for achieving "rule-based interoperability" in the framework of CORBA.

* Acknowledgement: This work is supported by the Advanced Research project Agency under ARPA Order #B761-00. It is a part of the R&D effort of the NIIIP Consortium. The ideas and techniques presented here are those of the authors and do not necessarily represent the opinion of other NIIIP Consortium members.

1 Introduction

Data sharing and program reuse across distributed, heterogeneous computing platforms have become one of the most challenging problems facing the computer science and engineering commu-

nities. In order to access information which are generated, stored and managed by dissimilar information systems and to activate program services which are available in heterogeneous software systems, national/international information infrastructure protocols need to be established. Two independent efforts on standards are particularly relevant to the development of these protocols; namely, the efforts of the Object Management Group (OMG) and the International Standard Organization's Committee on the STandard for the Exchange of Product model data (ISO/STEP).

OMG is formed by a consortium of over 500 industrial companies which aims to define and develop object-oriented technologies for achieving the interoperability among dissimilar computing platforms. It proposed a Common Object Request Broker Architecture (CORBA) [25] for achieving object interoperability. Based on the architecture, objects can make requests which are dispatched to the proper object servers and receive responses from them. The interfaces of all objects of interest are specified in a common Interface Definition Language (IDL) [1, 25]. An interface specification is compiled to generate program skeletons and stubs for inclusion into server programs and client programs, respectively. At run- time, a human or a software client can make remote calls for object services provided by the servers in a heterogeneous network, thus achieving the client/server interoperability. This approach achieves the "method-based interoperability".

The ISO/STEP community, on the other hand, emphasizes the development of standards for product modeling and product data exchange. One of its major efforts is the development of an information modeling language named EXPRESS [11]. EXPRESS provides a rich set of constraint specifications by using keywords, functions, procedures and constraint rules. It is a powerful information modeling language. The language has been widely accepted and used by a number of product design and manufacturing communities.

In spite of the individual success and acceptance of these two standards efforts, the results produced by each does not adequately solve the data sharing and program interoperability problems found in a heterogeneous environment. For example, the underlying object model of OMG's IDL is that of C++. While it may be adequate for achieving program interoperability (since the underlying data models of most of the existing O-O programming languages have the similar modeling power as IDL), the semantics captured by IDL is not rich enough for modeling complex objects processed by many existing application systems. When IDL is used as the common modeling language to model and encapsulate the objects and object services of an existing application system (e.g., a relational database application or a CAD application), much of the semantics of the objects cannot be captured explicitly because IDL does not have the necessary modeling constructs for capturing constraints by keywords and/or by integrity rules. Thus, much of the needed semantics have to be embedded in the application code. Recognizing the limitations of the object models underlying most of the existing object- oriented DBMSs, the Object Data Management Group (ODMG [4, 9]) has made some effort to extend the object model's capabilities to capture semantic constraints. However, these extensions are still far from meeting the actual needs for information modeling of complex manufacturing products.

On the other hand, although EXPRESS is semantically much richer than IDL and has an object-oriented flavor, it is not an object-oriented information modeling language because it does not support the encapsulation of behavioral properties of objects. Unlike methods found in an object-oriented model, functions and procedures defined in EXPRESS are global properties in a schema and are used in rules for constraint specifications. Also, the rule specification facility in EXPRESS has a limited expressive power. It is used mainly for specifying inter-attribute constraints. Also, there is a lack of constructs for specifying different types of associations and constraints among entities. Some of these limitations of EXPRESS have motivated different groups' efforts in extending EXPRESS such as EXPRESS-V, EXPRESS-C, EXPRESS-P, EXPRESS-M, etc. However, if we combine the behavioral specification of IDL and the information modeling power of EXPRESS

with some extensions into a single well-integrated object model and modeling language, then the resulting object model and language can be ideal for modeling objects and object services in a network of heterogeneous computing systems. The language can be designed to conform to the two standard languages (IDL and EXPRESS) semantically and, as much as possible, syntactically. It will be easy to use; particularly for those who already familiar with the two standard languages. The design and development of such a language can also serve as the glue that binds the two very active standards communities which have been working more or less separately so far.

The limitations of the above two standard languages and the potential advantages of their integration have motivated our work on an extensible and semantically-rich object model and its extensible modeling language. This R&D effort is a part of the project entitled the National Industrial Information Infrastructure Protocols (NIIIP) supported by the Technology Reinvestment Program (TRP) of the Advanced Research Project Agency (ARPA) [6]. A main objective of NIIIP is to develop an information infrastructure to support the operation of a virtual enterprise (VE). This infrastructure would allow a number of organizations to rapidly develop a working environment to manage a collection of resources contributed by the organizations toward the attainment of some common goals. Our objective is to develop a NIIIP Common Language (NCL) for modeling, not only the data and program resources of some existing (or legacy) systems contributed by the organizations of a virtual enterprise, but also the new organizational structure, workflow management, enterprise-wide semantic constraints, integrity and security rules, mediation and negotiation procedures and rules, object associations across systems, etc. The modeling requirements in the NIIIP environment demand that the underlying object model, the modeling language, and their supporting software system be semantically rich and extensible.

The focus of this paper is to describe the integration and extension of IDL and EXPRESS. The remainder of the paper is organized as follows. Section 2 discusses the modeling requirements in the NIIIP environment. Section 3 gives the overall syntactic structure of a schema defined in NCL. The extended features of the language, namely, constraints specified by keywords, association types, and ECAA rules are explained and illustrated by examples. Section 4 describes the key components of a knowledge base management system called OSAM*.KBMS [14, 15, 30, 31, 32, 33] and their build-time and run-time supports to the NCL language. A conclusion is given in Section 5.

2 NIIIP's Modeling Requirements

In order to develop an information infrastructure to support the interoperability of distributed and heterogeneous systems and to control and conduct the business of a virtual enterprise, it is necessary to model all aspects of the virtual enterprise such as data, human and hardware resources, organizational structures, business constraints, production process, and work management activities. The resulting conceptual model is an abstraction and a computational representation of the real-world objects, each of which can be a physical entity, abstract thing, concept, relationship, function, process, or anything of interest to a virtual enterprise. It captures the structural and behavioral properties of these objects as well as the constraints associated with them. We define some modeling requirements below and describe a processing engine in Section 4.

Object-oriented Representations of Existing Data and Application Systems' Resources. Member organizations possess all sorts of data and application systems' resources which are generated and used by different types of information users and/or heterogeneous information systems (e.g., relational application systems, CAD systems, SDAI/STEP applications, etc.). In a virtual enterprise environment, these data and systems resources can be uniformly modeled as objects in an object-oriented framework. Their data properties, associations (i.e., semantic relationships between

or among object types and their instances), constraints and operations (or methods) are defined by their object types resulting in a number of "object-oriented local conceptual schemas". Each of these schemas captures the semantic contents of the data and application systems' resources of a member organization that are deemed useful for the business operations and activities of a virtual enterprise.

Modeling of Virtual Enterprise Components. A number of VE Components are being developed by the NIIIP consortium to control the access and use of data and software resources of heterogeneous systems. They can be modeled in an object-oriented paradigm as object classes and their inter-relationships by various association types. These components' interfaces need to be explicitly defined so that their functionalities (or object services) are made known to NIIIP's human and software clients as well as to the components themselves. Activation of these services are done by either local message passing or by remote method calls through an ORB or other multi-transport data services. The models of these components are again captured in a number of object-oriented local conceptual schemas.

Modeling of Other VE Resources and Global Information. In addition to the modeling of existing data and application systems' resources and VE Components, other resources, such as tools, people, hardware devices, organization units and structures, and physical capital and monetary resources need to be modeled in the object-oriented framework. Some of these resources can be modeled by intelligent agents which represent the "interests" of these resources in the NIIIP environment. Furthermore, VE information which is "global" in nature and is to be shared by some VE Components will have to be modeled to inter-relate the modeled objects. Examples of such global information include:

- new associations that link object types of different local schemas
- new constraints to be imposed on them
- work management models that control the VE project or production activities
- negotiation procedures and rules that deal with the requests and deliverables of services
- mediation rules and operations needed to resolve the naming, structural and semantic conflicts and discrepancies among local and global resources

3 NIIIP Common Language (NCL)

The modeling requirements for a virtual enterprise environment discussed in the previous section have motivated and guided our design and development of NCL. In this section, the overall syntactic structure of a schema defined in NCL is given. Then, the features of the language which are extensions to EXPRESS and IDL are explained and illustrated by examples. Finally, the extensibility feature of NCL is discussed.

3.1 The Overall Structure of an NCL Schema

NCL is an integration of the language features of IDL, EXPRESS and K.3. The third language is the version 3 of an implemented knowledge base programming language developed at the University of Florida [3, 28, 29]. NCL's underlying object model is the extensible object model of K.3 which is founded on the concept of objects and object associations introduced in the Object-oriented Semantic Association Model (OSAM* [30]) and its algebra and calculus ([12, 34]). NCL uses 1) the method specification facility of IDL, 2) the type and entity specifications and the keyword

constraints (i.e., constraints specified by keywords) of EXPRESS, and 3) the knowledge rule specification, the language extensibility features, and the association type specification of K.3. Instead of presenting the detailed syntax and semantics of NCL, we shall describe the overall structure of the language. We shall assume that the reader is familiar with EXPRESS and IDL.

Overall Structure of an NCL Schema

```
(* SCHEMA declaration. *)

(* The SCHEMA class has an inclusion relationship with its component class
types *)
DEFINE SCHEMA schema_id;

END_DEFINE;

(* TYPE declaration *)
DEFINE TYPE type_id = underlying_type IN schema_id;
     WHERE  (* domain rule in TYPE *)
     rule_label_1: expression_1;
     ...
     METHODS:
     EXCEPTION exception_id (var_1:type_1;..);
     ...
     METHOD [ONEWAY] method_id
          ([IN|OUT|INOUT] para_id:para_type; ...): return_type
          [RAISES (exception_id, ...)];
     ...
END_DEFINE;

(* ENTITY declaration *)
DEFINE ENTITY entity_id IN schema_id;
     SUPERTYPE OF (supertype_expression)  (* supertype declaration *)
     SUBTYPE OF (subtype list)       (* subtype declaration *)
     attr_id: [OPTIONAL] base_type [WHERE ([TOTAL]
                              [CARDINALITY([L1:U1]:[L2:U2])]) ];
     ...
     DERIVE
        ...
     INVERSE
        ...
     UNIQUE
        ...
     [WHERE  (* domain rule in ENTITY *)
     rule_label_1: rule_expression_1;
     ...
     ASSOCIATIONS:   (* Other association types *)
     INTERACTION OF (attr_link_1:Entity_1;attr_link_2:Entity_2;...)
          CARDINALITY
          (attr_link_1:attr_link_2= [L1:U1]: [L2:U2];...);
```

```
          ...
     ...
     METHODS:        (* method declaration *)
     EXCEPTION exception_id (var_1:type_1;..);
     ...
     METHOD [ONEWAY] method_id
          ([IN|OUT|INOUT] para_id:para_type; ...): return_type
           [RAISES (exception_id, ...)];
     ...

     (* Local rule declaration *)
     RULES:
     RULE rule_id
       [TRIGGERED triggered_time trigger_operation, triggered_time
        trigger_operation, .....]
       [CONDITION condition_clause]
       [ACTION
          statement_list]
       [OTHERWISE
          statement_list]
     END_RULE;
     ...
END_DEFINE;

(* Global RULE declaration *)
RULE rule_id;
    [TRIGGERED triggered_time trigger_operation, triggered_time
      trigger_operation,.....]
    [CONDITION condition_clause]
    [ACTION
        statement_list]
    [OTHERWISE
        statement_list]
END_RULE;

(* Method implementation *)
METHOD [class_id::]method_id;
    [LOCAL
      local_var_declaration;
    END_LOCAL;]
    [statement_list]
END_METHOD;
```

The above structure gives the skeletal specifications of schema, type, entity, global rule, and method implementation. The symbols and clauses enclosed in a pair of brackets [ ] can be optional when other mandatory or optional symbols and clauses are present. The syntax of NCL resembles that of EXPRESS. However, minor changes have been made to EXPRESS in order to introduce the language extensibility feature and other added features of NCL.

3.2 Schema, Type and Entity Specifications

NCL provides the constructs for defining schemas, data types and entities. Their definitions are enclosed by a pair of keywords DEFINE and END_DEFINE. In NCL, a schema is treated as a first class object. The resources of different organizations participating in a virtual enterprise can be defined by separate schemas and their interrelationships can be specified by associations among these schema objects much the same way as the associations among data objects. The definitions of TYPE and ENTITY are similar to those of EXPRESS except that their semantic contents are enriched. In the TYPE specification, the behavioral properties of a data type are specified in terms of methods which have the same semantic contents as IDL's interface specifications.

In the definition of an ENTITY, the SUPERTYPE/SUBTYPE specification is the same as EXPRESS. The attribute specification and frequently-used constraints defined by keywords (e.g., UNIQUE, OPTIONAL, DERIVE, and INVERSE which we called "keyword constraints") are the same as EXPRESS. Additional constraints such as 1) TOTAL (i.e., the total participation constraint which specifies that all the objects in the class from which the attribute draws its values have to associate with some objects which have the attribute), 2) CARDINALITY (i.e., cardinality mappings between the class in which the attribute is defined and the class from which the attribute draws its values), and 3) other user-defined constraints associated with the attribute are specified as identifiers which follow the keyword WHERE.

In addition to the above structural and constraint specifications of an ENTITY, methods, rules and associations can be specified as part of the ENTITY declaration. They are described in the following two subsections.

3.3 Association Specifications

The ASSOCIATION specification in the ENTITY declaration is borrowed from the OSAM* model. In OSAM*, entities defined in a schema or across schemas can have different types of semantic associations (or relationships). In fact, the superclass-subclass and attribute specifications in an ENTITY declaration are two general types of associations; namely, Generalization and Aggregation associations which have long been recognized by the database community [27]. These two association types are predefined types in the kernel model of NCL. Other association types can be introduced by means of model extension for capturing the diverse and complex semantic relationship among object classes. For example, the INTERACTION association type shown in the ENTITY declaration can be useful for capturing the relationships between Employee-Department, Part-Supplier, etc. Association types such as SEQUENTIAL, PARALLEL, SYNCHRONIZATION, TESTING, DATAFLOW, and DECOMPOSITION can be used to model the interrelationships among workflow activities or manufacturing processes.

Figure 1 shows a simple model of a manufacturing process defined by a number of interrelated process classes named DispatchParts, DesignProduct, AssembleProduct and TestProduct. In the semantic diagram, process classes are represented by rectangles labeled with a "P". Aggregation associations (labeled with an "A") are used to define the data attributes of these process classes. These attributes are used to describe a process in terms of the data it manipulates. Data flow relationships among processes are represented by DataFlow associations (labeled with an "F"). A DataFlow association models the fact that the defining class (sending process) sends some data objects to one or more constituent classes (receiver processes). In a DataFlow association, a parameter output_data is used to specify the data objects that are transferred between processes. For example, in Figure 1, the process DesignProduct sends "product_design" to process AssembleProduct and process TestProduct; AssembleProduct sends "assembled_product" to TestProduct; TestProduct

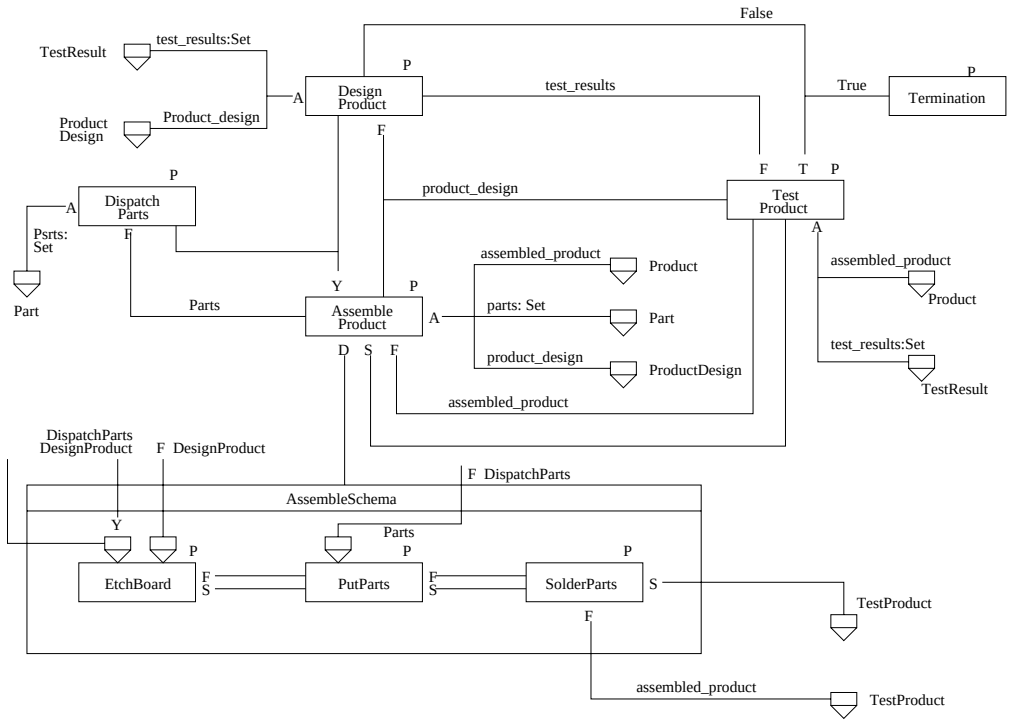

Figure 1: A Simple Model of a Manufacturing Process

sends "test_results" to DesignProduct, and DispatchParts sends "parts" to AssembleProduct.

Notice also in Figure 1 the use of a Decomposition association, labeled with a "D", to connect the process AssembleProduct to the schema class AssembleSchema. A Decomposition association is used to specify that a class (the defining class) is decomposed into a more detailed model defined by a schema (the constituent class). In our example, the process AssembleProduct is decomposed into a set of interrelated subprocesses EtchBoard, PutParts and SolderParts, which are modelled in the schema AssembleSchema. The Decomposition association type allows a process in a top-level view of a process model, which is treated as a blackbox with its input and output information, to be linked to a detailed and decomposed view of the process. The decomposed model of the process may again contain a process which is further decomposed and represented by a third level schema. The decomposition association type is not only applicable to processes, but also to any other type of classes for which a schema is defined as a detailed representation (such as a software component). This hierarchical decomposition of object classes is not supported by most of the existing object- oriented data models. However, it is very useful for modeling objects with different level of abstractions.

In most of the existing information system development methodologies (e.g., IE [24] and SSADM [8]), separate tools are used for managing data resources and business workflows/processes. For example, data resources are defined by data models such as the relational model [7], the Entity-Relationship model [5] and its variations, IDEF-1X [20]. Whereas, businessworkflows/processes are defined using models like IDEF0 [10], KI-SHELL [13] and DFD [22]. However, in the real world, data resources and business workflows/processes are closely related and highly inter-dependent. They model complementary aspects of a real or virtual enterprise. Thus, it is essential for information systems of the future to model and manage data and business workflows/processes in an integrated manner. Efforts in integrating different modeling languages can be found in projects such as and

AMADEUS [21] and TEMPORA [26]. By doing so, the inter-relationship among data resources and business workflows/ processes can be examined, their consistency can be verified, the flow of data/information among the business functional components or production processes can be studied to identify and, thus, avoid bottlenecks, and the performance metrics associated with these components and processes can be gathered. This is essential for achieving the concepts of enterprise integration and virtual enterprise.

3.4 Method and Rule Specifications

In modeling products, software components, or any other type of objects, it is not sufficient to model only their structural properties. The behavioral properties in the form of methods or operations need to be defined also so that only meaningful operations can be performed on them. Furthermore, in addition to frequently used constraints expressed by keywords, many complex semantic constraints, business rules, government policies, etc., which can only be expressed in a powerful rule specification language, need to be explicitly defined. In an entity class declaration in NCL, a number of methods and knowledge rules can be defined which are used for processing the instances of the entity class. The method and exception specifications are borrowed from IDL. However, the syntax has been changed to conform to the syntax of K.3.

The rule specification is borrowed from K.3. Different from the constraint rules of EXPRESS, which are used for specifying inter-attribute constraints, rules in NCL are Event- Condition-Action-AlternativeAction rules (or ECAA rules). An ECAA rule contains 1) an event (or Trigger) specification which includes the operations that trigger the evaluation of the CAA parts of a rule and the time for the evaluation relative to the event that triggers it (Before/ immediate_after/after), 2) a condition specification which may involve the verification of a complex pattern of object interconnections in multiple object classes, a complex quantified expression that involve multiple attributes of different classes, and/or a guarded expression that involves multiple data conditions which are to be evaluated sequentially, 3) an action specification which specifies a list of system-defined operations (i.e., knowledge base retrieval and manipulation operations) and/or user-defined operations (i.e., methods) which are to be processed if the condition specification is evaluated to True, and 4) an alternative action specification whose operations are to be processed if the condition specification is evaluated to False.

The CAA parts of a rule can be triggered before or immediate-after the occurrence of a trigger_operation specified in the event part of the rule or after a transaction commit. A triggered operation may in turn trigger other rules. The automatic enforcement of these rules makes the underlying system active. It has been widely recognized that the concepts and techniques of agents, mediators and negotiators [18, 19, 36, 37] are very useful for achieving information and program sharing in a complex, heterogeneous computing environment. The rule specification mechanism of the NCL is useful for implementing agents, mediators and negotiators in a heterogeneous computing environment because their implementations can make use of the active capability of monitoring events and automatically causing some intelligent behaviors to be carried out. It should also be emphasized that the rule specification language of NCL is far more powerful than the constraint rule of EXPRESS because the latter does not have the event/trigger specification capability nor the specification of complex conditions that involve the processing of attributes and object instances of multiple classes. Although, EXPRESS allows the inclusion of functions and procedures in constraint specifications (e.g., f(x) = 2), they are not operations or methods that can be activated based on the result of a condition evaluation. The ECAA rules are useful for defining security and integrity constraints, business constraints, policies and regulations, negotiation rules, mediation rules, agent activities, etc. Some examples on the application of rules are given below:

Example 1: (Security constraint) In class ServerA, before executing the method called name_service, execute the method get_permission() defined in the object class SecurityManager:

```
RULE get_perm1
TRIGGERED BEFORE name_service()
ACTION
  SecurityManager.get_permission(SELF,"name_service");
END_RULE;
```

The above is an example of a security rule which checks the authorization of a request for a name service. Notice that a method is invoked on another object, in the class SecurityManager, with the ServerA object (SELF) and method name as parameters.

Example 2: (Referential constraint) Before deleting a Department object, delete all the working records of the employees that work in the department. In class Department:

```
RULE remove_emp_records
TRIGGERED BEFORE DELETE()
ACTION
  CONTEXT SELF * [dept] w:Works_on * [emp] Employee
    w.DELETE();
  END_CONTEXT;
END_RULE;
```

Notice the use of the CONTEXT statement, which is used for the retrieval of objects from the knowledge base using an association pattern specification. In this example, all the "w" Works_on instances which are associated with the Department (SELF) and any Employee are deleted. The attributes "dept" and "emp" are defined in the class Works_on.

Example 3: (Mediation) Before retrieving a design in EXPRESS, check whether the design is defined in CATIA. If so, convert from CATIA to EXPRESS.

```
RULE mediation1
TRIGGERED BEFORE retrieve_design(design_id : String)
CONDITION EXIST d IN CONTEXT d:Design
          WHERE d.design_id = design_id AND d.representation = "CATIA"
ACTION
  convert_CATIA_2_EXPR(design_id);
END;
```

3.4.1 Global Rules and Method Implementations

Global rules are those rules which are applicable to the objects of all the classes. If some rules are applicable only to the objects of some specific classes, they can be defined as local rules in a superclass of these classes and be inherited by them. This is similar to the definition of common attributes and/or methods in the superclass of a number of subclasses. Thus, NCL supports not only the multiple inheritance of attributes and methods, but also ECAA rules. Rules are naturally distributed among classes in a class hierarchy or lattice.

The method implementation provides the actual program code for implementing the corresponding method specification. It can be coded in any programming language. In our work, we use K.3 as a programming language for method implementation for NCL. Therefore, with the definition facilities shown in the above structural declaration and the programming language facilities adopted from K.3, NCL can be considered as a full-fledged, high-level, extensible, object-oriented programming language.

3.5 NCL Extensibility

The existing data models and their modeling languages, such as the relational model, the ER model and its extensions, the object-oriented models and these models' DDLs, have a fixed set of modeling constructs. The semantics of these constructs are generally hard-coded in the DBMSs that use them. EXPRESS, as an information modeling language, is not an exception. It also has a fixed set of modeling constructs (e.g., type, entity, rule, function, keywords for specifying constraints, etc.). When a modeling language and its supporting system are used in different applications, the users often find that the fixed set of modeling constructs are not adequate for capturing the semantics found in data, programs, and other resources needed in these applications. They would like to see that the language and the system be extended to increase their functionalities. Unfortunately, any change or addition made to an underlying data model and its language will entail difficult and costly changes to the language translator as well as the supporting system. Ideally, both the language and the system should be extensible. Any change made to the underlying data model will automatically be reflected in the language and the system without changing the language translator or recoding the system. Thus, they can be extended and tailored to fit different application environments and needs.

In the NIIIP environment, different virtual enterprises may have different requirements for modeling the objects, object services, constraints and association types found in their own environments. New semantic properties in terms of new data types, association types, and constraint types often need to be added to the modeling language to meet the changing needs of these virtual enterprises. For this reason, NCL is designed and implemented as an extensible information modeling language.

The extensibility features of NCL are provided through the use of a meta-model, parameterized rules, a rule-binder, and a rule processor. A meta-model is the result of using the modeling constructs of NCL to model itself. The extensible meta-model for NCL begins with a simple kernel which has the similar structural and behavioral specification capabilities as the object models of many object-oriented systems, including the object model of OMG's IDL. The main difference is that knowledge rules and parameterized rules can be specified in object classes to capture the semantics of different types of constraints, classes and associations and to implement the extensibility features of NCL.

Starting from the kernel object model, new modeling constructs such as new class types, constraint types, and association types can be added into the kernel object model by adding new meta-classes to define these new types. The semantics of these new modeling constructs are defined in the respective meta-classes in terms of parameterized rules. These rules defined the operations that need to be performed on the affected objects when they are processed. The parameterized rules are converted into bound rules at the schema compilation time by a rule binder and the bound rules are incorporated into the object classes that make use of the modeling constructs. At runtime, these rules are used by the rule processor to enforce the semantics of the modeling constructs. Thus, by extending the meta-model, we can extend the modeling capabilities of the object model.

To achieve constraint-type extensibility, the semantics of all the keyword constraints (the ones in EXPRESS and the extended ones) used in the language are defined by a set of parameterized

rules in the meta-model. When a schema is compiled, these rules are bound to those attributes or object classes which have these constraint types. If a new constraint type and its semantic specification in terms of parameterized rules have been added to the meta-model by a knowledge base customizer, the constraint type can be used in a schema declaration without having to change the NCL compiler.

In an EXPRESS schema, ENTITY and TYPE are treated as keywords since they are fixed constructs of the EXPRESS's underlying data model. In NCL, however, ENTITY, TYPE, SCHEMA, and the names of other new class types are treated as identifiers instead of keywords to the NCL compiler. The keyword DEFINE signals the compiler that the character string following it should match with a class type defined in the meta-model. Other class types can be added to the meta-model by the knowledge base customizer who customizes the object model to meet the modeling needs of an application domain. For example, for workflow modeling, a class type named PROCESS can be added to the meta model and the structural and behavioral properties and constraints can be defined in the corresponding meta class. Any identifier can be used as the name of a class type as long as the name used in NCL is consistent with the name used in the meta model in which the class type is defined. Thus, by adding new class types into the meta model, we can extend the modeling capability of NCL, thus achieving NCL's class-type extensibility.

To achieve association-type extensibility, the association types used in the example in Figure 1 and other user-defined association types can be defined in the meta-model by a knowledge base customizer so that they can be used by the users of NCL. Similar to class types and keyword constraint types, all association types are defined in the meta-model in terms of parameterized rules. These rules defined the operations that need to be performed when the objects of those classes which enter these association types are processed. They are converted into bound rules at the schema compilation time and incorporated into the object classes that make use of the association types as if the person who defines the schema has explicitly written these bound rules. At run-time, these rules are used by the rule processor to enforce the semantics of the association types. This feature of association-type extensibility allows any semantic relationships between or among object classes (thus, among their instances) that are frequently used in an application to be defined as association types. Once defined, the user can use these association types to relate object classes without having to repeatedly specify their rules in all the classes that involve in these types of associations.

The details of data model extensibility in NCL is beyond the scope of this paper. Interested readers should refer to [17, 35].

4 NCL's Supporting System and Tools

The processing of NCL is supported by a knowledge base management system OSAM.KBMS and a number of tools [32, 33]. Figure 2 shows the key components of the KBMS and tools. A number of graphical tools called XGTOOLS (its earlier version was described in [16]) can be used to define, edit and browse knowledge base schemas and to graphically query the schemas against their instances stored in the knowledge bases. The defined schemas can then be translated into NCL in a textural form. XGTOOLS also provides the editing facility for the user to enter NCL schemas in textual forms directly. In addition to these graphical tools, an EXPRESS-to-NCL compiler is available to import the existing EXPRESS schemas into NCL . We have tested this compiler using over 30 existing EXPRESS schemas. The mappings of the constructs of these two languages are shown in Table 1. A translator is being developed to convert the existing IDL specifications into NCL. In the NIIIP project, EXPRESS schemas and IDL specifications which define the data and

progrom interfaces of some legacy systems can be converted into NCL and additional resourses of a virtural enterprise (data, software systems developed in the NIIIP project, hardware devices, tools, organizational structures, personnel resources, etc.) can be defined in NCL. Mediation rules are then introduced to specify the mediation processes needed to bridge the structural and semantic descrepancies between the modeled properties of heterogeneous systems. The result is a global mediated schema which provides the global view of the virtual enterprise's knowledge base. The global knowledge contains 1) the meta data useful for the controlled access to data and program resources in a heterogeneous network (note: the actual data and programs are distributed among different systems), and 2) the common data which are used by the NIIIP-developed software systems to control the accesses of the data and software resources in the heterogeneous network.

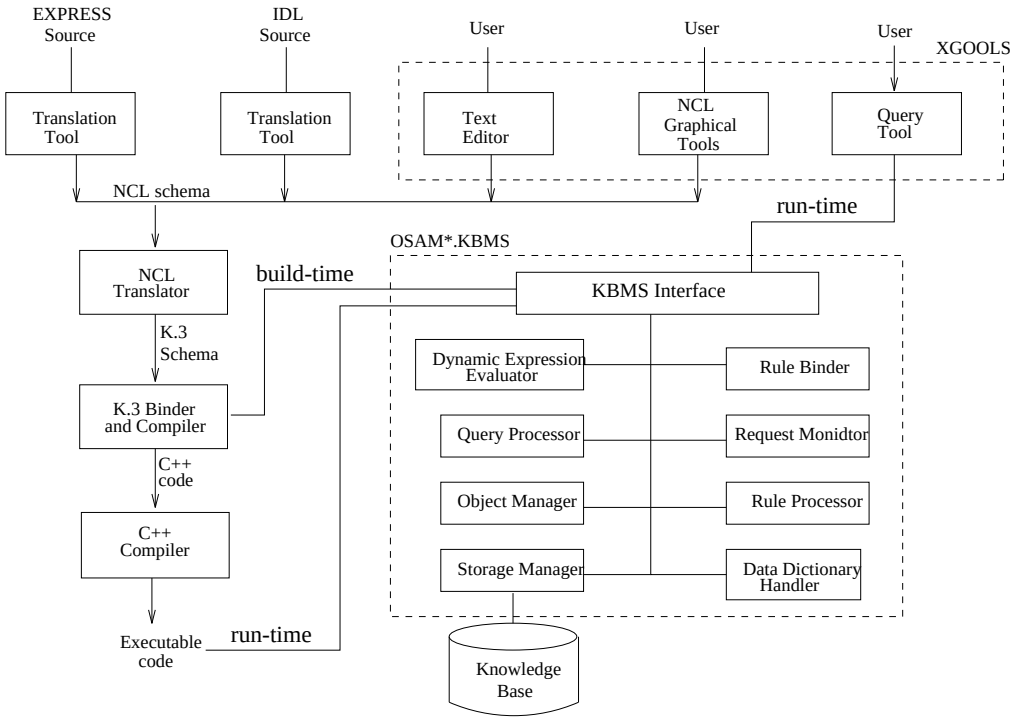

Figure 2: KBMS Components and Tools for NCL

Using these two types of data, the KBMS provides two general types of services: Run-time and build-time services. At build-time, if NCL is used to model a software system and to implement the methods of its components (i.e., NCL is used as a knowledge base programming language), the resulting model and method implementations are first translated into K.3. Then, a K.3 compiler is used to generate .h files and C++/C code that corresponds to the structural and semantic properties of the K.3 schema. During the translation, the semantics of keyword constraints, association types and class types, which are stored in the corresponding meta classes as parameterized rules, are translated into bound rules by a rule binder. These rules are bound to the classes in which the constraint, association and class types are used. During this translation, the event specifications of all the knowledge rules defined in the schema are used to produce code for program binding. The CAA parts of all the rules are translated into C++/C methods. The C++ code produced by the K.3 compiler is then compiled by a C++/C compiler to generate the executable code which is linked with the components of the KBMS. At run-time, the KBMS provides the query processing and rule processing functions and persistence support to the program.

EXPRESS	NCL
SCHEMA	SCHEMA class
ENTITY	ENTITY class
TYPE	TYPE class
PROCEDURE	METHOD
FUNCTION	METHOD
Attribute specification and standard constraints defined by keywords (e.g., UNIQUE,OPTIONAL,DERIVE INVERSE,etc)	Same as EXPRESS
Additional and user-defined constraints (e.g., TOTAL, CARDINALITY,etc)	Follow the keyword WHERE in attribute delcaration
RULE	ECAA RULE
Domain-rule (WHERE clause)	Same as EXPRESS
More semantic association types (e.g. INTERACTION, SEQUENTIAL, PARALLEL,etc)	Defined in ASSOCIATIONS section

Table 1: The mapping of EXPRESS and NCL

A second build-time service is to receive a NCL schema created by a client using a copy of the XGTOOLS installed at the client site and establish the knowledge base for it. The client can then populate the instances of the knowledge base at run-time using a query language. At run- time, if a client (human or software client) wants to access the meta data or data stored in the KBMS, he/she/it can make a method call to the query processor QP of the KBMS (see Figure 2) by passing a query as a parameter (note: the functionality of KBMS and its QP are also defined in NCL). The query language OQL (an early version of this language is reported in [2]) is used for this purpose. During the execution of a query, if a system-defined operation (retrieve, delete, or update) or a user-defined operation (display_part (), hire_employee(), etc.) given in the query meets the event specification of an ECAA rule or rules stored in the KBMS, the condition part of the rule(s) will be automatically triggered for evaluation. The action taken based on the result of the rule evaluation may again trigger other rules. Query processing and rule processing make use of the functionalities of an Object Manager which in turn makes use of those of the Storage Manager (Exodus) shown in the figure.

A third build-time service of NCL and KBMS is to use knowledge rules to generate program binding for achieving client-server interoperability. In the conventional CORBA environment, the interfaces of these systems are defined in IDL. They are translated into program language bindings (e.g., C and C++ stubs and skeletons) which are incorporated in the client and server programs to achieve the run-time interoperability. The control and logical relationship among the clients and servers are written in program code. Thus, the interoperability of the OMG architecture is "method-based". Whereas, in the NIIIP environment, the interfaces are defined in NCL which, after the translation process described in a previous section, are essentially IDL specifications plus

ECAA rules which capture the semantics of keyword or other user-defined constraints, association types and class types. Rules can be used to define not only the constraints associated with data but also the control logic that inter-relate the services of NIIIP VE servers. For example, upon the receipt of a request from a client, a method in the session server is triggered to start a session, the session service may in turn trigger a method in the workflow server. Furthermore, they can be modified more easily than program code to reflect the new interrelationships among clients and servers should their interrelationships change.

ECAA rules captured in NCL together with the NCL's method specifications, which are equivalent to IDL specifications, can be used to generate language bindings for client and server programs at the build-time. At run-time, the activations of object services as specified in the rules will be carried out automatically across the ORB, meaning programs will be calling each other through the ORB following the trigger and action specifications. For this reason, the proposed technique is called "rule-based interoperability". To achieve this, we need to generate program stubs and skeletons to implement the methods in NCL specifications but also a mechanism to monitor the execution of the methods (request/event monitoring) and to trigger the processing of methods which implement the CAA parts of rules (rule processing). For more details on the use of rules to achieve "rule-based interoperability" among heterogeneous systems, the reader is referred to a full length paper [35].

5 Summary and Conclusion

In this paper, we have stressed the importance of having an object model and language which is standard-based, extensible, and semantics-rich for modeling the data, software and other resources of a large real or virtual enterprise and for achieving rule-based interoperability among heterogeneous systems. We presented an extensible language NCL which combines the features of EXPRESS, IDL and our own K.3 languages. The key features of NCL are as follows:

1. The design of NCL is standards-based, incorporating the features of the two standard languages, IDL and EXPRESS, and conforming to them as much as possible. It follows the basic principles of the CORBA architecture for interface definition, with the added semantic richness of EXPRESS.

2. NCL supports additional association types; i.e., different semantics properties can be attached to different types of semantic associations between or among object classes so that, during the processing of their object instances, the semantic properties of the association types can be automatically enforced or maintained.

3. NCL contains a high-level rule specification component. Rules in NCL can used for defining integrity and security constraints, government or enterprise policies and regulations, and other types of semantic constraints that are local or global to heterogeneous systems. It allows the specification of events that could automatically trigger the verification of the states of distributed data and take alternative actions based on the result of the verification.

4. NCL is extensible. Additional semantic properties found in a heterogeneous environment can be easily introduced into NCL and its object model as new constraint types, class types, and/or association types. These extensions to NCL are automatically supported during execution, causing no changes to the language compiler nor the supporting KBMS.

NCL is supported by the build-time and run-time services of an object-oriented KBMS and is being applied in the NIIIP project to provide rule-based interoperability.

References

[1] L. Acker, et al., "SOMobjects Development Toolkit Users Guide," IBM, Version 2.0, June 1993.

[2] A. Alashqur, S. Y. W. Su and H. Lam, "OQL– A Query Language for Manipulating Object-oriented Databases", in *Proc. of 15th Int. Conf. Very Large Databases*, Amsterdam, Netherlands, pp. 433-442, August 1989.

[3] J. A. Arroyo-Figueroa, "The Design and Implementation of K.1: A Third Generation Database Programming Language," Technical Report, Database Systems R&D Center, University of Florida, August 1992.

[4] R. G. G. Cattell et al., "The Object Database Standard - ODMG 93. Morgan Kaufmann", 1993.

[5] P. P. S. Chen, "The entity-Relationship Model: Toward a unified view of data," ACM TODS, vol. 1, no. 1, 1976.

[6] N. Christopher, et al. "The NIIIP Reference Architecture," Prepared for OMG Review, March 1995.

[7] E. Codd, "Relational Databases: A Practical Foundation for Productivity," CACM, 25:2, December 1982

[8] E. Downs (ed), Structured Systems Analysis and Design Method - Application and Context, Prentice Hall International (UK) Ltd, 1988

[9] D. Florescu, L. Raschid and P. Valduriez, "Query Reformulation in Multidatabase Systems using Semantic Knowledge," Paper Draft, 1995.

[10] "Integrated computer aided manufacturing (ICAM) architecture part II, Volume IV -function modeling manual (IDEF0)", Report Number: AFWAL-TR-81-4023 V.4. SofTech Inc., 460 Totten Pond Road, Waltham, MA 02154.

[11] Subcommittee 4 of ISO Technical Committee 184, "Product Data Representation and Exchange - Part 11: The EXPRESS Language Reference Manual," *ISO Document*, ISO DIS 10303-11, August 1992.

[12] N. Kamel, P. Wu and S. Y. W. Su, "A Pattern-Based Object Calculus," *International Journal on Very Large Data Bases*, Boxwood Press, Vol. 3, No. 1, Jan. 1994, pp. 53-76.

[13] Knowledge-based Integration Methodology for Workflow Support (KIShell), UES Inc., 1993.

[14] H. Lam, S. Y. W. Su and A. Alashqur, "Integrating the Concepts and Techniques of SEmantic Modeling and the Object-oriented Paradigm," *Proc. 13th Int'l Computer Software & Applications Conference (COMPSAC)*, pp. 209-217, October 1989.

[15] H. Lam, et al., "Prototype implementation of an Object-oriented Knowledge Base Management System," extended abstract in *The Second Florida Conference on Productivity through Computer*, Integrated Engineering and Manufacturing, pp. 68-70, November 1989.

[16] H. Lam, S. Y. W. Su, et al., "GTOOLS: An Active GUI Toolset for an Object-oriented KBMS," *International Journal of Computer System Science and Engineering*, Vol.7,No.2,pp. 69-85, April 1992.

[17] H. Lam, S. Y. W. Su, et al., "Model Extensibility in an Extensible Knowledgebase Management System," submitted to *IEEE Transactions on Knowledge and Data Engineering*.

[18] Lander, S. and Lesser, V., "Customizing Distributed Search Among Agents with Heterogeneous Knowledge," *Proceedings of the First International Conference on Information and Knowle dge Management*, Baltimore, MD, November 1992.

[19] Lander, S. and Lesser, V., "Understanding the Role of Negotiation in Distributed Search Among Heterogeneou s Agents," In *Proceedings of the International Joint Conference on Artificial Intellig ence*, August/September 1993.

[20] M. E. S. Loomis, "Data Modeling – The IDEF-1X Technique," IEEE communication, March 1986.

[21] P. Loucopoulous, W. J. Black, A. G. Sutcliffe, and P. J. Layzell, "Towards a unified view of systems development methods," Int. J. Inf. Manage., 7:205-218, 1987.

[22] T. De Marco, Structured Analysis and System Specification, Prentice-Hall, 1982.

[23] J. Martin, Information Engineering (Book 1: Introduction), Prentice Hall, Inc., Englewood Cliffs, NJ, 1989.

[24] J. Martin, Information Engineering (Book 3: Design and Construction), Prentice Hall, Inc., Englewood Cliffs, NJ, 1990.

[25] OMG Committee, "The Common Object Request Broker: Architecture and Specification," *OMG Document*, Revision 1.1, No. 91.12.1, December 1991.

[26] TEMPORA. Project Manual - Esprit Project 2469. Technical Report, BIM, Belgium, 1992.

[27] J. Smith and D. Smith, "Database Abstractions: Aggregation and Generalization", TODS, 2:2, June 1977.

[28] Y. M. Shyy, S. Y. W. Su, "K: High-level Knowledge Base Programming Language for Advanced Database Applications," *ACM SIGMOD Int'l Conf. on Management of Data*, pp. 338-347, 1991.

[29] Y.M. Shyy, J. Arroyo-Figueroa, S. Y. W. Su, and H. Lam, "The Des ign and Implementation of K: A High-level Knowledge Base Programming Language of OSAM*.KBMS," accepted for publication in the *VLDB Journal*, 1995.

[30] S. Y.W. Su, V. Krishnamurthy and H. Lam, "An Object Oriented Semantic Association Model (OSAM*)," *AI in Industrial Engineering and Manufacturing: Theoretical Issues and Applications*, American Institute of Industrial Engineering, 1989.

[31] S. Y. W. Su and A. M. Alashqur, "A Pattern-based Constraint Specification Language for Object-oriented Databases," *Proc. of IEEE COMPCON'91*, San Francisco, Febrary 25- March 1 1991.

[32] S. Y. W. Su and H. Lam, "An Object-oriented Knowledge Base Management System for Supporting Advanced Applications," *Proc. of the 4th Int'l Hong Kong Computer Society Database Workshop*, pp. 3-22, December 12-13, 1992.

[33] S. Y. W. Su and H. Lam, et al., "OSAM*.KBMS: An Object-Oriented Knowledge-Base Management System for Supporting Advanced Applications," *Proc. of the 1993 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 540-541, 1993.

[34] S. Y. W. Su, M. Guo and H. Lam, "Association Algebra: A Mathematical Foundation for Object-Oriented Databases,' ' *Transactions on Knowledge and Data Engineering*, IEEE, Vol. 5, No. 5, Oct. 1993, pp. 775-798.

[35] S. Y. W. Su and H. Lam, et al. "NCL: A Common Language for Achieving Rule-Based Interoperability among Heterogeneous Systems", Submitted for publication *Journal of Intelligent Information Systems, Special Issue*, 1995.

[36] G. Wiederhold and M. Genesereth, "The Basis for Mediation," *Proc. COOPIS'95 Conference*,Vienna Austria, May 1995.

[37] G. Wiederhold, "Interoperation, Mediation, and Ontologies", presented at the *FGCS/ICOT Workshop* in Tokyo, 13 Dec. 1994.