Meta-Data Components in Support of an Active Deductive Object-Oriented Database System

Taoufik Ben Abdellatif, Hon Wai Rene Chan, Suzanne W. Dietrich,
Babu Siddabathuni, Amy Sundermier, Susan D. Urban
Department of Computer Science & Engineering
College of Engineering and Applied Sciences
Arizona State University
BOX 875406
Tempe, AZ 85287-5406, U.S.A.
phone: (602) 965-3190
fax: (602) 965-2751
{taoufik,rene.chan,dietrich,amy.sundermier,s.urban}@asu.edu

ABSTRACT

This paper describes a layered meta-data approach to the support of active, deductive, object-oriented database (ADOOD) environments. The integration of these three database paradigms, together with the need to support active rule analysis, testing, and debugging activities has created the need for sophisticated data structures to manage meta-data. The unique aspect of our layered approach to meta-data is the division of meta-data into static and dynamic components. Static components manage the meta-data generated from the compilation of structural schema definitions together with the language components of the database application code, including active and deductive rules as well as methods and transactions. Static meta-data is enhanced with additional meta-data that is generated by static rule analysis and testing tools for use by run-time rule analysis and debugging tools. Dynamic components store meta-data that is associated with the evolving database state that occurs as a result of transaction and rule processing. Dynamic meta-data components therefore provide abstractions of state changes associated with different language components, thus allowing the development of more sophisticated tools for run-time analysis and debugging of active database rules. This paper elaborates on the static and dynamic components of our layered meta-data architecture in the context of the tools needed for language evaluation and active rule development.

1.0 INTRODUCTION

Active database systems incorporate rule processing capabilities into traditional database environments for the purpose of providing reactive behavior to events that occur within database applications. Since the development of active applications is a complex task, we have been investigating innovative techniques for the analysis, testing, and debugging of active rules as part of the ADOOD (Active, Deductive, Object-Oriented Database) RANCH Project (NSF Grant No. IRI-9410993). In particular, our work has focused on the use of rules within the Comprehensive, Declarative Object Language (CDOL)[17]. CDOL is a language that integrates active, deductive, and object-oriented capabilities to create a knowledge-oriented approach to the specification of active rules. A rule-based query language is the most central feature of the environment, supporting additional language layers for the declarative specification of constraints, updates, and active rules.

The integration of active, deductive, and object-oriented database concepts, and our objective to support

rule analysis, testing, and debugging activities has created the need for sophisticated data structures to manage meta-data. At a minimum, the system requires traditional meta-data components to describe the structural aspects of the database schema. The additional processing needs of our research, however, have required the development of new forms of meta-data to support ADOOD language and rule execution as well as the creation of meta-data abstractions that better support active rule development tools. In this paper, section 2 initially provides an overview of the meta-data for our ADOOD development environment. The following sections then describe the base meta-data in more detail, where Section 3 describes the class meta-data for CDOL and Section 4 describes the meta-data in support of active and passive rules. An overview of the dynamic meta-data for an ADOOD development environment is presented in Section 5. Section 6 briefly addresses related work. Section 7 concludes the paper with a discussion of our contributions and future research directions.

2.0 META-DATA FOR AN ADOOD DEVELOPMENT ENVIRONMENT

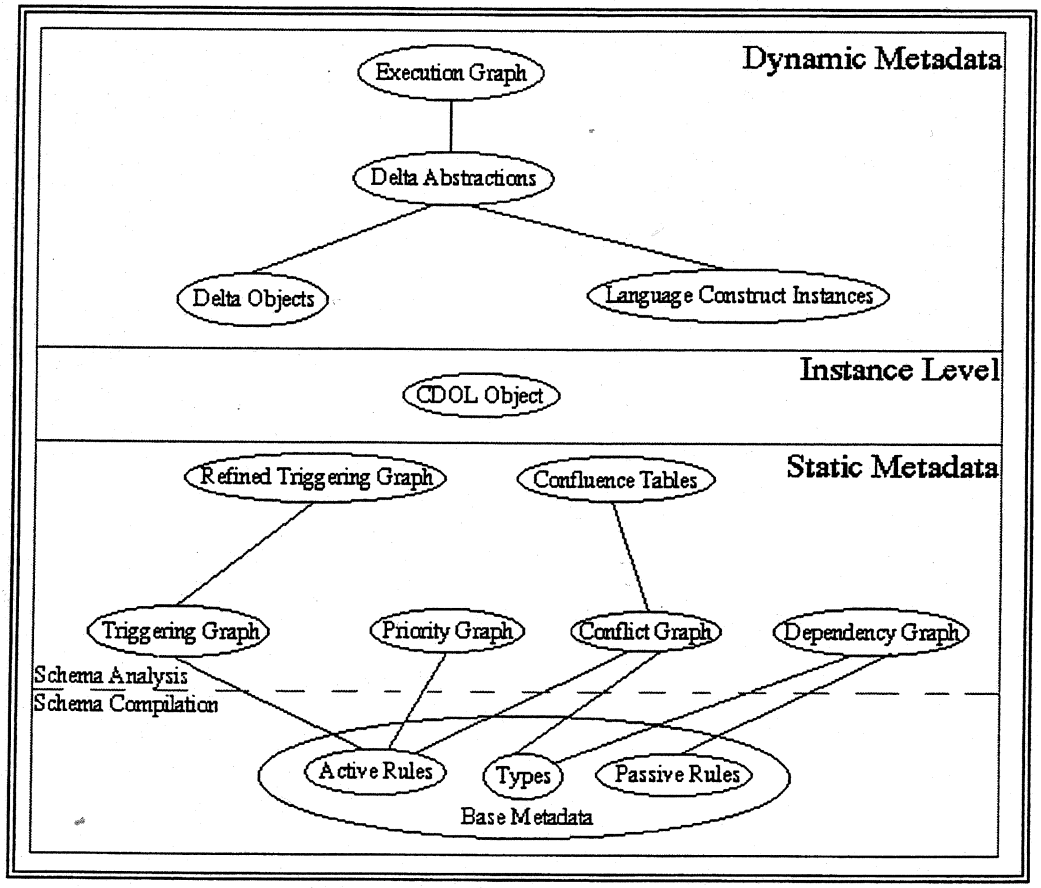

Figure 1. Meta-data Layers.

Our approach to the development of an ADOOD meta-data component for CDOL consists of layers of meta-data, illustrated by Figure 1. A base layer of meta-data known as the Schema Compilation Layer is generated from the compilation of a CDOL schema definition and the CDOL rules that specify the application code. The Schema Compilation Layer of meta-data for CDOL consists of structural schema information (i.e., class and attribute definitions), the description of active and passive (i.e., deductive) rules, the description of behavioral language components such as methods and transactions, and links that establish the relationships between classes, attributes, rules, methods, and transactions. The structural schema meta-data and rule meta-data will be described in more detail in sections 3 and 4,

respectively.

The Schema Compilation Layer is then analyzed to construct an additional layer of meta-data known as the Schema Analysis Layer. The Schema Analysis Layer contains meta-data that is generated by various static rule analysis and testing tools [6, 16] for use by run-time rule analysis and debugging tools [9]. Both the Schema Compilation Layer and Schema Analysis Layer contain meta-data that is static, meaning that it is not modified during the execution of the active database application.

The Schema Analysis Layer of meta-data includes various graphs and tables based upon static analysis of CDOL rules. Triggering graphs are needed to support an active rule testing tool as well as static rule analysis tools that examine active rules for termination and confluence behavior. Basic triggering graphs show the triggering relationships among active rules based upon potential event generation, while refined triggering graphs [16, 18] use more sophisticated analysis that is able to remove false triggering relationships. Priority graphs establish the order in which the rules should be executed according to the rule programmer, based upon keywords in the active rule definitions. Conflict graphs show the rules that access the same underlying database values, therefore potentially resulting in non-confluent behavior if the rules are executed concurrently. Confluence tables are generated from more sophisticated analysis [11] of the potential conflicts, eliminating false conflicts and thereby generating reduced sets of conflicting rules and enabling greater concurrency. Dependency graphs show the relationships between rules and the data used in the rules. Dependency graphs are used to guide the incremental update of materialized virtual data, which are defined by deductive rules, when any underlying objects are modified. Dependency graphs are also needed to support condition monitoring activities to perform incremental evaluation of the condition of Condition-Action rules.

The Database Instance layer shows the root class, CDOL Object, of the application inheritance hierarchy in the ADOOD RANCH system. A CDOL Object contains meta-data associated with each instance of an application object. For example, a CDOL Object contains references to the application object's type meta-data in the Schema Compilation Layer. The Database Instance Layer is not static information since it changes during execution as application object instances are created and destroyed.

If the term meta-data is defined simply as "data that represents characteristics of data", then there is no inherent restriction that meta-data consist solely of static information. Due to the complexity of the CDOL processing environment, we have discovered the need to define a Dynamic Layer of meta-data that exists on top of the Database Instance Layer. The Dynamic Layer contains meta-data that is concerned with representing information about the processing of CDOL rules in forms that can be used by dynamic rule debugging and analysis tools. The meta-data at this level is generally short-lived, representing evolving database state information generated by transactions and rule processing. Delta Objects capture incremental changes to objects in the database. Language Construct Instances represent execution instances of CDOL Transactions and CDOL rules. Delta Abstractions organize and filter dynamic meta-data into forms that can be easily used by run-time tools for rule analysis and debugging. Execution Graphs represent scheduling information for concurrently executing active rules. The Dynamic Layer will be described in more detail in section 5.

3.0 META-DATA FOR CDOL STRUCTURAL SCHEMA DEFINITION

The CDOL rule language influenced every aspect of the design of the ADOOD RANCH system. CDOL has a data definition language based on the ODMG ODL (Object Definition Language) [4]. In the ADOOD Ranch system, all meta-data for an application is defined by a programmer through the CDOL

data definition language or through the use of a graphical interface to the meta-data. This section of the paper briefly describes the meta-data for the ADOOD RANCH Project that stores structural schema definitions for classes, attributes, and relationships. The CDOL Class Meta-data is shown in Figure 2 in UML notation [8]. Figure 2 primarily illustrates classes and relationships between classes, but omits most attributes and methods from classes for the sake of brevity. The ADOOD RANCH Project team initially implemented the CDOL meta-data and object storage using Shore (Scalable Hetereogeneous Object REpository) [3, 13] and provided C++ access APIs to the meta-data and object storage. The CDOL object model is similar to C++ [7] because it supports multiple inheritance of both properties and methods. The CDOL meta-data has also been implemented in Java. The Java language [10] does not support multiple inheritance in the same manner, but instead uses Interfaces to promote method signature inheritance without implementation inheritance. Figure 2 therefore represents some class types as interfaces.

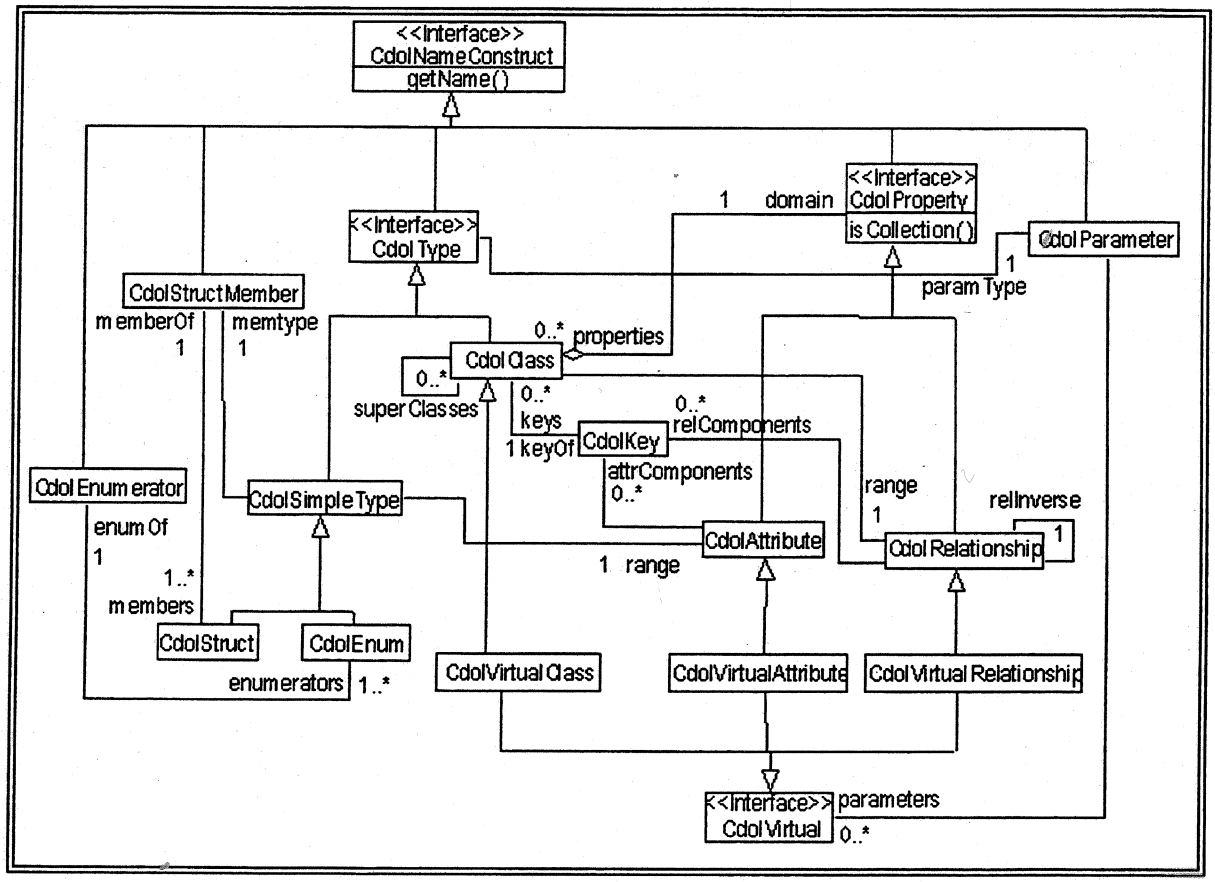

Figure 2. CDOL Class Meta-data.

The *CdolNameConstruct* in Figure 2 is one of the most commonly used interfaces in our schema, requiring that many meta-data classes implement a *getName* method to provide named access. A *CdolClass* represents the class of a *CDOL Object*, and has recursive relationship *superclasses* to represent inheritance hierarchies. In Figure 2, *CdolClass* is a subclass of *CdolType*, representing the fact that classes are types. A class can have properties, represented by a relationship with *CdolProperty*, which are either attributes or relationships. *CdolAttribute* represents attributes of simple types, while *CdolRelationship* represents relationships with other classes. Since CDOL always specifies inverse relationships, instances of *CdolRelationship* are paired for related classes. CDOL provides some additional features to describe structured, multi-valued, and enumerated properties, hence the *CdolStruct*

and *CdolEnum* meta-data classes.

CDOL derived classes and properties are described in the reference for the CDOL Language [17] in detail, but the basic notion of a derived type is to create an intensional value based on a deductive rule. The intensional value is not stored in the database, but instead is computed upon use. *CdolVirtual* is an interface that relates a virtual property or virtual class to the deductive rule that defines its value. *CdolVirtualRelationship* and *CdolVirtualAttribute* override inherited operations from their superclasses *CdolRelationship* and *CdolAttribute* to return type, name, or class domain information. A virtual property appears like a database value to the referencing query or rule. *CdolVirtualClass* uses a deductive rule to derive a subset of the membership of a base *CdolClass*.

CdolParameter represents formal parameters to several of the CDOL constructs. For example, *CdolMethod* or *CdolTransaction*, which will be described in the section on rule meta-data, may require parameters for invocation. In addition, virtual properties or classes may be parameterized thereby restricting derived data to certain limits.

4.0 META-DATA FOR CDOL RULES

Structural schema meta-data for representing classes and properties is only part of the base layer of CDOL meta-data. The other component of the base layer represents CDOL rules as shown in Figure 3. The left side of Figure 3 represents both passive and active rules in CDOL. The right side of Figure 3 represents CDOL Transactions and Methods, which are integral to the specification of rules.

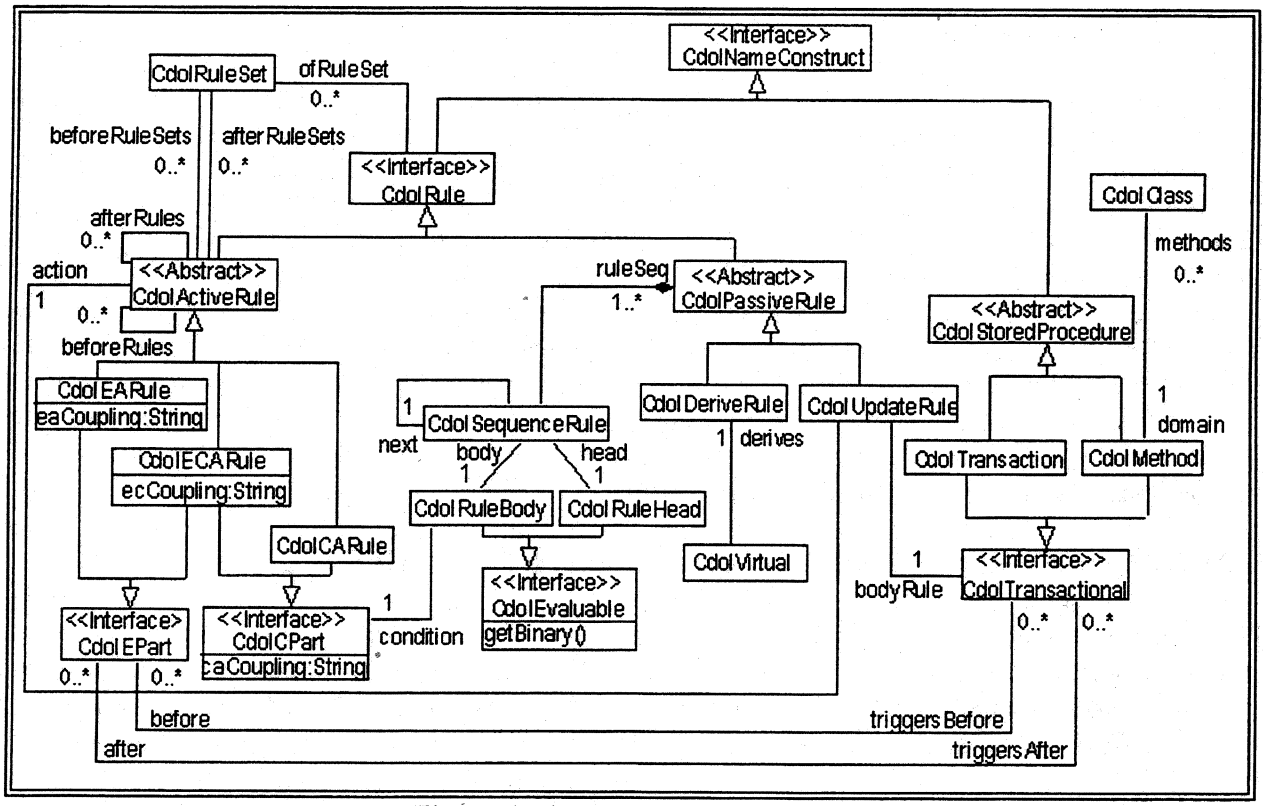

Figure 3. CDOL Rule Meta-data.

The *CdolRule* interface extends from *CdolNameConstruct*, thereby requiring rules to have names.

CdolRule is implemented by classes for both passive and active rules. The term passive rule typically refers to a declarative rule that derives data. An active rule is a rule that is triggered by some occurrence during transaction processing. In CDOL, any method invocation is an event that can trigger an active rule.

The *CdolPassiveRule* abstract class is a superclass for *CdolDeriveRule* and *CdolUpdateRule*. In the CDOL language, a passive rule may consist of a sequence of rules. The *CdolPassiveRule* class therefore aggregates an ordered list of *CdolSequenceRule* objects. Each *CdolSequenceRule* in the CDOL language has a head and body, represented by the *CdolRuleHead* and *CdolRuleBody* classes. The CDOL language evaluator can evaluate rule heads and rule bodies, therefore both *CdolRuleHead* and *CdolRuleBody* implement the *CdolEvaluable* interface. In general, bindings obtained from the evaluation of the rule body are passed to the rule head. The *CdolEvaluable* interface has a *getBinary()* method to retrieve intermediate code [14] that is used for interpreting CDOL by the evaluation engine. When CDOL is compiled, the intermediate code is stored as part of the meta-data. A *CdolUpdateRule* is a sequence of rules to update values in the database. A *CdolDeriveRule* is a deductive rule to derive a virtual value from the database, therefore derived rules have a relationship to the *CdolVirtual* property or class they derive.

The *CdolActiveRule* abstract class is a superclass for all active rules. All active rules have an Action that is represented by the sequence of rules in a *CdolUpdateRule*. The subclasses of *CdolActiveRule* such as *CdolEARule*, *CdolECARule*, and *CdolCARule*, are all concrete classes. *CdolEARule* and *CdolECARule* implement the *CdolEPart* interface to specify the events that trigger the active rule. *CdolCARule* and *CdolECARule* implement the *CdolCPart* interface to specify the condition of the rule, which is a CDOL query represented by a *CdolRuleBody*.

The right side of Figure 3 shows the *CdolStoredProcedure* abstract class. The *CdolStoredProcedure* class represents executable operations that can be called on the database directly from a user program. For example, *CdolTransaction* represents the list of user transactions defined on the database. *CdolMethod* represents the list of user accessible methods that can be called on application objects. *CdolTransaction* and *CdolMethod* are different only because a *CdolMethod* is associated with a particular *CdolClass*. In both cases, *CdolTransaction* and *CdolMethod* are written in the CDOL language as declarative update rules, not in an imperative programming language like C++ or Java. The rules that perform the operations of the transaction or method are defined within a *CdolUpdateRule* object accessed through the *bodyRule* relationship defined by the *CdolTransactional* interface. Active rules are triggered before or after the execution of a *CdolTransactional* type based upon a before/after specifier in the declaration of an active rule. This is shown as the relationships to the *CdolEPart* interface implemented by event-driven active rules.

5.0 META-DATA EXTENSIONS FOR RULE ANALYSIS AND EXECUTION

The Dynamic Layer of meta-data shown in Figure 1 contains meta-data that is concerned with representing information about the processing of CDOL rules in forms that can be used for rule execution and for dynamic rule debugging and analysis tools [9]. The Dynamic Layer contains Language Construct Instances, Execution Graphs, Delta Objects, and Delta Abstractions.

Language Construct Instances represent meta-data for the execution-time instances of CDOL Transactions and CDOL rules. The CDOL processing environment creates meta-data objects to represent the status of rules that are executing. The active rule processing algorithms [2] manipulate

Language Construct Instances similarly to the way an operating system manipulates and schedules processes. The Execution Graph contains the active rules that have been triggered and scheduled for execution. The Execution Graph uses the Refined Triggering Graph, the Priority Graph, and the Confluence Tables to guide a rule execution schedule in avoiding non-terminating and non-confluent rule behavior. In addition, the Execution Graph is capable of logging the execution history of active rules in the context of the executing user transaction. During rule debugging activity, the logged information can be used dynamically to make decisions about rule scheduling during the execution of user transactions or in a post mortem analysis process to assist in redesigning the user transactions if the execution behavior of the active rules was not as expected.

Object deltas provide an object-oriented version of relational deltas for capturing the incremental changes to objects that occur in the database state due to the execution of update rules within transactions and active rules. Object deltas are needed within the condition monitoring facility and within the rule analysis and debugging component for active rules. We have a unique structure in our Dynamic Layer of meta-data for capturing object deltas [15]. Object deltas were designed to behave as a natural extension to objects through object-oriented concepts such as inheritance, delegation, encapsulation, and abstraction. All database objects inherit the ability to create and manage deltas from features built into the CDOL Object class. Objects are responsible for creating and managing their own deltas as user transactions modify the state of the database objects. The meta-data structure for representing object deltas was created as a generic, collapsible object known as a *DeltaObject*, which dynamically grows and shrinks according to the incremental changes in the associated CDOL Object.

Due to the magnitude of the number of deltas that can potentially be created, a mechanism was needed to organize and abstract the manipulation of deltas. Delta abstractions [1] are a unique form of meta-data in that they are delegated with the responsibility of providing a view of the internal database changes associated with different language components that can cause changes to the database state. For example, in CDOL, delta abstractions exist for objects, properties, update rules, active rules, methods, and transactions. Using delta abstractions, active database programmers can more closely examine the execution of active rules, displaying the database state before and after the execution of different language components and performing rollback and replay of rule execution sequences. Delta abstractions therefore provide an intelligent and interactive state and meta-data management facility, thus allowing the development of more sophisticated tools for run-time analysis and debugging of active database rules.

6.0 RELATED WORK

We are not aware of any published results on the meta-data for an active, deductive, and object-oriented database. However, meta-model standards have recently emerged for object-oriented systems, such as the Object Data Management Group's ODMG 2.0 standard [5] and the Object Management Group's object model for CORBA [12].

The ADOOD project began in 1995, and developed its object model to be consistent with release 1.1 of the ODMG-93 standard [4]. CDOL supports the standard features of ODMG-93, extending the ODMG-93 Object Definition Language (ODL) with additional features required to define the derivation of virtual attributes and classes, integrity constraints and active rules. Since CDOL is a comprehensive declarative object language, it also includes a manipulation language for the declarative specification of update rules used to define methods, transactions and the action of active rules. A comprehensive example of CDOL can be found on the ADOOD web (http://www.eas.asu.edu/~adood

Although the ODMG-93 standard provided a detailed description of the object model, it did not specify a standard meta-model. Since meta-data was crucial to the implementation of our system, we designed and implemented our own meta-model, which is shown in Figures 2 and 3. The CDOL meta-model is a superset of the meta-model of an ODMG-93 compliant database, since CDOL supports additional features beyond that of a standard object-oriented database system.

The successor of ODMG-93, ODMG 2.0 [5] provided the definition of a standard meta-model. There are similarities and differences between the CDOL meta-model and the ODMG 2.0 meta-model. Both the CDOL and ODMG 2.0 meta-models are defined in terms of their target object model. For example, CDOL defines its meta-model in terms of the CDOL object model and ODMG 2.0 defines its meta-model in terms of the ODMG 2.0 object model. Although it is not required to define the meta-model using the target object model, doing so usually provides ODBMS implementers an easier path to bootstrap meta-data implementation with a limited functioning version of the ODBMS implementation itself. For the same reason, relational DBMS implementers usually implement their meta-data with a limited functioning version of their own relational DBMS. There are also various similarites between the structural meta-data maintained for CDOL and ODMG 2.0. For example, both meta-models provide an interface near the root of the meta-model class hierarchy to require many meta-data classes to implement a getName method in order to support named access. The interfaces are CdolNameConstruct in CDOL and MetaObject in ODMG-2.0.

Despite these similarities, there are differences between the two meta-models after ODMG 2.0 made substantial changes to the object model for increased compatibility with the object model of the CORBA standard [12] by the Object Management Group (OMG). The single most significant change in the object model is the introduction of the state versus behavior inheritance. In ODMG-2.0, interfaces define only the behavior of objects. Classes, instead, define the state of objects, which can also implement interfaces. State inheritance is provided through the EXTENDS relationship, which supports single class inheritance with the capability to inherit multiple interfaces. Another significant change of ODMG-2.0 from ODMG-93 is the revised interpretations of attributes and relationships. In ODMG-93 Release 1.1, attributes can only take values of literal (immutable object) types such as Integer and Enumeration. Properties of (mutable) object values are, instead, defined by relationships, which can support either unidirectional or bi-directional traversal. In ODMG-2.0, attributes with object values become the new approach to define unidirectional relationships, while relationship definitions always define bi-directional relationships. Furthermore, ODMG-2.0 also introduces the module concept to organize name scopes. In summary, although the current CDOL meta-data is compliant with ODMG-93, it is not ODMG-2.0 compliant.

Although we do not anticipate changing the existing object and meta-models of CDOL to be compliant with ODMG 2.0, we will use the ODMG 2.0 standard and the knowledge obtained from our implementation of the CDOL meta-data to form the basis of our future work on the development of meta-data in support of active rule processing in a distributed object computing environment.

7.0 CONCLUSION

This paper has presented a layered approach to the management of meta-data for an ADOOD environment with support for rule analysis, testing, and debugging activities. The contribution of our work is found in the extension of structural meta-data components for an object-oriented schema to include 1) meta-data for active and deductive rules as well as ADOOD language components, and 2) meta-data for static rule analysis and testing tools. In addition, we have defined a dynamic meta-data

component for capturing semantic and run-time abstractions that more readily support the development of tools for examining rule execution. Due to the complex nature of active applications, the development of such tools is critical to the successful use of active rule technology. Our meta-data framework follows similar developments with current standards for object-oriented models and provides a prototype of the meta-data components that are needed as such standards expand to include more sophisticated forms of rule processing. Our future work is focused on the development of meta-data in support of active rule processing in distributed object computing environments.

8.0 REFERENCES

[1] Ben Abdellatif, T., Urban, S. D., Sundermier, A., and Dietrich, S. W., "Delta Abstractions: A Technique for Managing Database States in Active Rule Processing," Technical Report, August 1998.

[2] Ben Abdellatif, T., *An Architecture for Active Database Systems Supporting Rule Analysis Through Evolving Database States,* Ph.D. Dissertation, Arizona State University, Spring 1999.

[3] Carey, M. J., DeWitt, D. J., Franklin, M. J., Hall, N. E., McAuliffe, M. L., Naughton, J. F., Schuh, D. T., Solomon, M. H., Tan, C. K., Tsatalos, O. G., White, S. J., and Zwilling, M. J., "Shoring Up Persistent Applications," *Proceedings of the 1994 ACM SIGMOD Conference on the Management of Data*, Minneapolis, MN, May 1994.

[4] Cattell, R. (editor), *The Object-Oriented Database Standard: ODMG-93*, Morgan Kaufmann, San Francisco, CA, 1994.

[5] Cattell, R. (editor), *The Object-Oriented Database Standard: ODMG 2.0*, Morgan Kaufmann, San Francisco, CA, 1997.

[6] Chan, H. W. R., Dietrich, S. W., and Urban, S. D., "On Control Flow Testing of Active Rules in a Declarative Object-Oriented Framework," *1997 International Conference on Rules in Database Systems*, pp. 165-180.

[7] Ellis, M. A., Stroustrup, B., *The Annotated C++ Reference Manual*, Addison-Wesley, Reading, MA, 1990.

[8] Fowler, M., and Scott, K., *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, Reading, MA,1997.

[9] Jahne, A., Urban, S. D., and Dietrich, S. W., "PEARD: A Prototype Environment for Active Rule Debugging," *Journal of Intelligent Information Systems, Special Issue on Active Database Systems* , Volume 7, Issue 2, October 1996, pp. 111-128.

[10] Jaworski, J., *Java Developer's Guide*, Sams.net, 201 West 103rd Street, Indianapolis, IN, 1996.

[11] Karadimce, A., *Termination and Confluence of Active Rules in Active Object Databases*, Ph.D. Dissertation, Arizona State University, Spring 1997.

[12] Object Management Group, *The Common Object Request Broker: Architecture and Specification,* Rev.2.2, February, 1998.

[13] The Shore Project Group, *An Overview of Shore,* Version 0.9.3 Beta, Computer Sciences Department, University of Wisconsin-Madison, September 15, 1995.

[14] Siddabathuni, B., Dietrich, S. W., and Urban, S. D., "An AQUA-based Intermediate Language for Evaluating an Active Deductive Object-Oriented Language," *International Workshop on Practical Aspects of Declarative Languages (PADL99)* , Springer Verlag Heidelberg, pp. 163-177.

[15] Sundermier, A., , Ben Abdellatif, T., Dietrich, S. W., and Urban, S. D., "Object Deltas in an Active Database Development Environment," *1997 International Conference on Deductive and*

Object-Oriented Database Systems, pp. 211-229.

[16] Tschudi, M. K., Urban, S. D., Dietrich, S. W., and Karadimce, A. P., "An Implementation and Evaluation of the Refined Triggering Graph Method for Active Rule Termination Analysis," *1997 International Conference on Rules in Database Systems*, pp. 133-148.

[17] Urban, S. D., Karadimce, A. P., Dietrich, S. W., Ben Abdellatif, T., and Chan, H. W. R., "CDOL: A Comprehensive Declarative Object Language," *Data & Knowledge Engineering*, 1997, pp. 67-111.

[18] Urban, S. D., Tschudi, M. K., Dietrich, S. W., and Karadimce, A. P., "Active Rule Termination Analysis: An Implementation and Evaluation of the Refined Triggering Graph Method," To appear in the *Journal of Intelligent Information Systems*, 1999.

ACKNOWLEDGEMENTS

This research was supported by the National Science Foundation (NSF Grant No. IRI-9410993).

Third IEEE Computer Society

Metadata Conference

Natcher Building & Conference Center, NIH Campus, Bethesda, Maryland

April 6-7, 1999

Sponsored by:

- IEEE Computer Society Technical Committee on Mass Storage Systems Technology
- National Oceanic Atmospheric Administration
- National Imagery and Mapping Agency

In Cooperation with:

- IEEE Computer Society Digital Libraries Task Force
- Raytheon ITSS Corporation

http://computer.org