Temporal Rule Specification and Management in Object-oriented Knowledge Bases[1]

Stanley Y. W. Su
Electrical Engineering Dept.
and Computer & Information Science Dept.
Database Systems R&D Center
University of Florida, Gainesville, FL 32611
email: su@pacer.cis.ufl.edu

Hsin-Hsing M. Chen
Electrical Engineering Dept.
Database Systems R&D Center
University of Florida
Gainesville, FL 32611
email: hmc@reef.cis.ufl.edu

Abstract

There have been many recent research efforts on temporal databases for managing current and historical data and on active databases for monitoring real-world events as well as data conditions/constraints by rules and triggers. However, the specification and management of temporal rules in active object-oriented databases has not been investigated. This paper deals with the specification and management of temporal requirements and constraints of real-world applications. We present a temporal knowledge model in which temporal rules are defined as part of the semantic specifications of object instances and object classes. We also present a temporal specification language which is characterized by its validity time specification, its trigger specification that involves time and system- and user-defined operations, and the inclusion of temporal conditions and association patterns in temporal rules. Three general types of temporal rules are distinguished: state rules for specifying the legitimate or illegitimate states of a temporal knowledge base, operational rules for specifying the operations to be taken under various temporal conditions, and deductive rules for deducing objects' data values and object associations which are not explicitly stored. Activations of temporal rules are controlled by triggers which specify the various times and conditions for rule evaluation. Rules can be updated resulting in historical rules. Rules can be inherited in a superclass-subclass hierarchy or a lattice as the inheritance of attributes and operations in the object-oriented paradigm. Temporal rules are modeled as first class objects and thus can be managed uniformly as temporal data by a temporal knowledge base management system. The methods for updating, retrieving, triggering, and evaluating temporal rules are also presented.

Keywords: temporal data model, rule specification language, temporal rule, knowledge base management

1. This research is partially supported by the National Science Foundation grant #DMC-8814989 and the National Institute of Standards and Technology grant #60NANB4D0017. The implementation effort is supported by the Florida High Technology and Industry Council grant #UPN 90090708.

1. Introduction

There are two very important topics presently being investigated by researchers in the database community: temporal database management and active databases. The advancement of both topics is essential for the development of sophisticated knowledge base management systems (KBMSs) for supporting future advanced applications. Research and development on temporal database management is motivated by the need to support decision-making based not only on current data but also on historical information. Efforts on temporal models, temporal languages, storage structures, and access methods for the implementation of temporal database systems have been reported in [AND82, CLI83, KLO83, SCH83, LUM84, CLI85, SNO85, ADI86, ARI86, GAD86, TAN86, SEG87, SNO87, LOR88, NAV89, TAN89, ELM90a, ELM90b, ROS91, SU91]. However, these research on temporal DBMS concentrate only on the management of the explicitly stored temporal data [MAI91]; techniques for managing the data which are not explicitly stored but can be derived from the temporal dependencies/relationships of the existing data are not provided. Research on active databases is motivated by the need for a database management system to automatically monitor and react to real-world (or external) events as well as database states and constraints. Research efforts on this subject have been reported in [MOR83, STO85, DIT86, HSU88, DAY88a, DAY88b, CHA90, BEE91]. However, the specifications of triggers and conditions associated with rules (e.g., the *condition* part of HiPAC's ECA rules) in these research are limited to the current information because of the lack of a mechanism for managing and processing temporal data. In many advanced applications such as computer aided design, office information systems, decision support systems, military command and control, etc., the processing and reasoning on temporal data are important. Therefore, it is natural and important to consider the merging of these two areas of efforts and investigate the problem of the specification and management of temporal rules in an active database or, more appropriately, knowledge base management system. To the authors' knowledge and as pointed out in [SIL90, SNO90], this problem has not been investigated, particularly in the context of object-oriented knowledge base management.

Knowledge rule has been the most common form of knowledge representation among a number of knowledge representation paradigms [NEW80, BAR81, CER83, SMI83, WOO83, MYL84, LEV87]. The use of knowledge rules provides an effective mechanism for an intelligent system to predict changes over time, to take different actions under various conditions, and to deduce answers based on observable things and events stored in a knowledge base. Complex application requirements, semantic constraints, and rules for reasoning and logical deduction can all be

expressed in terms of knowledge rules. In knowledge-based systems that incorporate knowledge rules as part of the semantic description of an application world, a rule may make reference to some temporal data conditions in both the condition part and the consequence part of the rule. The trigger associated with a rule may also make reference to temporal conditions under which the rule is to be evaluated. Furthermore, the applicability of rules, like data, may depend on the time these rules are applied. A rule that is meaningful at one time may not be applicable at another time. Similar to data, rules may be updated. Updated rules become historical rules which are applicable only to historical data. We shall call these updatable knowledge rules that make reference to temporal data conditions "temporal rules."

An advanced knowledge base management system should provide users (knowledge base designers and end users) with the language facility for defining temporal rules associated with objects of concern in an application. It should also be capable of managing and enforcing these temporal rules. In this paper, we present the temporal rule specification component of an object-oriented semantic association model and the technique for managing temporal rules. This is the follow-up of the work reported in [SU91] in which a temporal model and its query language were presented. The intended contribution of this paper is the introduction of the concepts and techniques for: (1) the incorporation of historical data reference in knowledge rules, (2) the specification of the valid times of temporal knowledge rules, and (3) the management of temporal knowledge rules and their evolution. In section 2, the use of temporal rules as part of the object-oriented knowledge base definition is presented. In section 3, a temporal rule specification language is

described with examples to illustrate three general types of temporal rules. Section 4 presents the techniques for modeling and processing temporal rules. A conclusion is given in section 5.

2. Temporal Rules as Part of Object-Oriented Knowledge Base Specification

The temporal knowledge model OSAM*/T [SU91] is an extension of OSAM* [SU89] for modeling the temporal aspect of real-world objects and objects' histories. It provides a number of modeling constructs to serve as a conceptual basis for uniformly capturing the semantics and inter-relationships among temporal objects of an application world. All things of interest in an application world, i.e., physical objects, events, processes, functions, etc., are uniformly modeled as objects in OSAM*/T and are grouped into object classes based on some common semantic properties. An object class, thus, captures the structural and behavioral semantics common to a set of objects. Object classes and their associated objects are interrelated through various association types, each of which represents a set of operational rules governing the manipulation of objects of the associated classes. Five system predefined association types (aggregation, generalization, interaction, composition, and crossproduct) are provided in OSAM*/T. Additional user-defined association types or subtypes of the existing association types are possible since association types are modeled as classes in the implementation model of a KBMS to be shown latter.

In OSAM*/T, object histories are recorded discretely by time-interval stamps using an object instance time-stamping technique [SU91]. However, the interpretation of the time semantics for the object evolution is continuous [SEG87]. We also assume the equivalence of physical time and logical time [LUM84, SNO86] in OSAM*/T and use the notion of valid time introduced in [KIM90, SAR90, MAI91] to represent this equivalence concept. The Start-time and the End-time of the valid time are the only time notions adopted in OSAM*/T to uniquely characterize each temporal object instance. Other time notions introduced in [LUM84, SNO86] for some particular applications such as retroactive update of data and metadata (i.e., rules) are specified by temporal rules. This approach is taken to save storage space and processing time and to achieve the needed flexibility in introducing new time notions for diverse applications. For example, the retroactive update of a rule, which is discovered today but should have been effective in the 18th century, can be captured by two temporal rules: the first one is the updated rule itself with today's date as the valid time and the second rule is the rule which says that retrieving the data of the 18th century should trigger the execution of the first rule. With this approach, we can avoid introducing extra time notions such as user-defined time, record time, etc. which need to be

incorporated into every data (and metadata) unit once they are included into the database schema and thus save tremendous storage space. This approach can also avoid the time-consuming database schema evolution which is needed when the extra time notion is not decided in advance during the database design stage.

2.1 Object Class Definition

Temporal rules in OSAM*/T are treated as part of the object class definition. In OSAM*/T, an object class consists of three parts: (1) a specification part which defines the structural properties of the class (i.e., its descriptive attributes and associations with other classes), the meaningful operations (signatures) that can be performed on its object instances, and the temporal rules that are applicable to its instances, (2) an implementation part which contains the methods or program code for carrying out the specified operations, and (3) an extension part which contains the set of object instances belonging to the class. Object instances (similar to relational tuples) are the representations of objects in a class. An object has a system-assigned unique OID and an instance also has a unique instance identification or IID which is the concatenation of a class ID and OID. OSAM*/T allows an object to be a member of more than one class, thus having multiple instances. Temporal rules in OSAM*/T define the temporal constraints that objects of a class should always satisfy or obey. Temporal rules that are applicable to objects in multiple classes are defined in a superclass having these classes as its subclasses. These rules are their common semantic properties, thus are inheritable by their object instances in a manner similar to the inheritance of common attributes and operations (see section 2.3 for more details.) This category of rules is called class rule. Different temporal rules can also be defined specifically for different instances of an object class. This is achieved by storing these rules as values of a common attribute whose data type is Rule. This category of rules is called instance rule. Thus, a temporal rule can be associated with an individual object instance or with a class (if it is applicable to all the instances of the class.) Distinguishing instance rule from class rule has two advantages. First, when some rules are applicable only to some specific object instances, the specification of instance rules is the mechanism for capturing the individual behaviors of the object instances and the processing of other instances will not involve the checking of these rules. Second, the instance rule mechanism allows different sets of rules to be associated with different instances of a class by using several attributes of type Rule. During run-time, a method can explicitly activate or deactivate different sets of rules. For example, one can search all instances of a class that satisfy some conditions and activate/deactivate a specific set of instance rules. Naturally, the set of class rules applicable

to all instances will also be processed.

The template of a class definition is shown below:

```
Class_type <class_name>
{ ASSOCIATION_SECTION:
    Association-type = Association_type_1;
          { association_name_1 : domain;
            association_name_2 : domain;
          ..... }
    Association-type = Association_type_2;
          .....
  OPERATION_SECTION:
   { operation_1();
     operation_2();
    .....}
  TEMPORAL_RULE_SECTION:
   { temporal_rule_1;
     temporal_rule_2;
     ..... }
} /* end of class definition */
```

In OSAM*/T, the structural properties of an object class are defined in terms of its associations with other related classes. As shown in the template, the association section specifies the different system-defined or user-defined association types that an object class has with some other classes. The operation section specifies the operations which are applicable to the instances of the object class. These operations are defined by function and/or procedure declarations (i.e., the signatures) and their methods or program code. The temporal rule section specifies a set of temporal class rules which are applicable to all of its instances.

In the following, we give an example of a class definition and a class rule which says that for those employees who are working on project P2 and have worked on project P1 during the period of T[01-01-73, 12-31-75] should not be fired. Additionally, we also define an attribute Erule of type Rule (i.e., an aggregation association between Employee and Rule) in this class definition for a later explanation of instance rule following this example.

Example of an object class definition and a class rule

We define the Employee class based on the semantic diagram (or S-diagram which is used in OSAM*/T to graphically represent a database schema) shown in Figure 1:

```
ENTITY_CLASS Employee
{
ASSOCIATION_SECTION:
          AGGREGATION OF
```

```
        { salary: Salary;
          title : Title;
          Erule  : Rule;}
        GENERALIZATION OF
        { Engineer, Manager, Secretary; }
END ASSOCIATION_SECTION

OPERATION_SECTION:
        {COUNT(); AVERAGE(); Fire(); CancelFire();}
END OPERATION_SECTION

TEMPORAL_RULE_SECTION:
        {
        Rule 00001
        Valid_T [01-25-77, -]
        Trigger-cond (Before Fire(Employee))
        IF (Employee * Work_On * Project[P#=P2])
                INTERSECT (Employee)
          (WHEN T[01-01-73, 12-31-75]
          CONTEXT Employee * Work_On * Project [P#=P1])
        THEN CancelFire(Employee)
        END /* end of class rule */}
END TEMPORAL_RULE_SECTION
} /* end of class Employee */
```

In this example, a temporal rule is defined in the Employee class with a rule ID 00001. Rule 00001 was defined on Jan. 25, 1977 and is still valid (this fact is represented by the valid time interval expression Valid_T [01-25-77, -], where Valid_T is a reserved keyword and "-" of the valid time interval stands for an infinite time point); it prevents those employees who are working on project P2 and have ever worked on project P1 during the period of T[01-01-73, 12-31-75] from being fired. The rule will be triggered before an attempt is made to fire (a user-defined operation) such an employee. In Rule 00001, Trigger-cond is the keyword used to specify the triggering condition(s) of the rule; CONTEXT is the keyword used to specify the context of the operation (i.e., a subdatabase specified by some association pattern expression); INTERSECT is a Set Operator used to perform on two temporal contexts; WHEN is the keyword used to specify the time information of a temporal context; and "**IF** boolean expression **THEN** statement **ELSE** statement" is the structure used in a temporal rule to capture a cause-effect relationship. All these keywords in the rule structure will be further explained in Section 3.

Example of instance rule

Since Employee class has an attribute Erule, every instance of the class can be associated with a different instance rule. In OSAM*/T, the data type Rule is modeled by a class called Rule. An instance rule associated with an

employee instance is an instance of the Rule class and the value of Erule of the employee instance is the IID of this rule instance. In the following, we give an instance rule for employee John. This rule specifies the formula used to calculate John's actual Salary during the period T[01-07-83, 12-23-86].

```
Rule 00002
Valid_T [01-01-87, - ]
Trigger-cond (Before Retrieve(Salary))
IF (WHEN T[01-07-83, 12-23-86]
   Employee)
THEN Salary =: Salary * 1.2
END /* end of instance rule */
```

This instance rule identified by 00002 was defined on Jan. 01, 1987 for the object John of the Employee class and is still valid (represented by the Valid_T [01-011-87, -]). It captures the application of retroactive update on John's salary during the period T[01-07-83, 12-23-86] and will be triggered before a retrieval of John's salary during this period. Other operations will not cause it to be verified. Rule 00002 says that John's actual salary of the period T[01-07-83, 12-23-86] should be 20% more than the recorded salary and this retroactive update is valid since Jan. 1, 1987. The action specified in the THEN clause of this rule computes the salary value. It does not change the content of the database.

From the above two examples, it is obvious that both instance rules and class rules have the same format. They can be handled uniformly by the same rule processing mechanism of a KBMS.

2.2 Object instance time-stamping

Object instance time-stamping is the technique employed to record the historical data in OSAM*/T [SU91]: every object instance in each class is time-stamped with a valid time interval [Start-time, End-time]. Start-time and End-time of the valid time are the two time notions used in OSAM*/T to uniquely characterize each historical object instance. The Start-time is the time when the information represented by an object becomes valid and the End-time is one time unit before the information becomes invalid. When an object instance is initially created, the Start-time is set to the instance creation time and the End-time is set to infinity represented by "-". As the object instance evolves (i.e., a change in the attributes of the object instance is observed), the current object instance will have a new Start-time which is the time when the object instance is modified and an End-time of infinity. The old version then becomes history and its End-time is set to one time unit before the Start-time of the new version. The old version is shifted into the historical

area.

An example of object instance time-stamping is given below. In this example, object instance Mary of the Employee class was created on 12-15-85 with an instance id "04" and had an infinity as its End-time. When Mary's salary was raised from $20K to $30K on 12-15-88, the End-time of the old version was set to 12-14-88 before it was shifted into the historical area; and the new version has a Start-time of 12-15-88 and an End-time of infinity. Here, a "day" is assumed to be one time unit. Any other time unit (second, minute, hour, month, year, etc.) can be used to suit an application domain.

(1) Initial creation of object Mary on 12-15-85:

IID	Name	Title	Salary	Start-time	End-time
<04	Mary	Secretary	$20K	12-15-85	- >..Current

(2) Update of object Mary on 12-15-88:

IID	Name	Title	Salary	Start-time	End-time
<04	Mary	Secretary	$30K	12-15-88	- >..Current
<04	Mary	Secretary	$20K	12-15-85	12-14-88>..Historical

2.3 Inheritance of Structures, Operations and Rules

The structural and behavioral properties of a superclass are inherited by its subclasses so that these properties do not have to be defined in the subclasses repeatedly. Inheritance is implied by the "generalization" association in OSAM*/T. For example, the attributes Salary and Title, the operations, and the temporal rules of the Employee class defined in Section 2.1 can be inherited by all its subclasses Engineer, Manager, and Secretary.

In some applications, a temporal rule may be applicable to instances of many object classes which do not have a generalization association among them. One approach is to define the rule in all these classes. However, this approach will result in rule redundancies. An alternative approach used in OSAM*/T is to treat the rule as part of the semantic properties of a superclass which has these classes as its subclasses and the rule is made available to them through inheritance. For example, in Figure 2, Graduate_Student can be defined as a superclass of two pre-existing classes TA and RA. A rule such as "if a graduate student is either a TA or an RA, then only the academic advisor has the authority to see his (or her) GPA" can be defined in Graduate_Student and be inherited by TA and RA. Since rules are the semantic properties of objects just like attributes and operations, they are also inheritable by objects of subclasses.

3. Temporal Rule Specification Language

The general form of a temporal rule in OSAM*/T is given in Table 1 and the BNF is given in the appendix. In Table 1, "rule-id" is the rule identification (RID). A rule is valid during "Valid_T [A, B]" which is a reserved keyword representing a valid time interval between Start-time A and End-time B; the Start-time is the time when a rule is valid to the temporal KBMS while the End-time is one time unit before a rule terminates its validity. "Trigger-cond (Trigger-time, Trigger-operation)" specifies the situation (or events) in which the temporal rule will be triggered. The "Trigger-time" is drawn from four possible situations: before, immediate-after, delayed-after, parallel; the "Trigger-operation" specified either system-defined or user-defined operations such as InsertInstance(), InsertObject(), DeleteInstance(), HireEmployee(), etc. The trigger time specifies the time for evaluating the rule relative to the time for carrying out the trigger operation. It is also the mechanism used in our implementation model for the coupling/decoupling between the original transaction and the spawned transactions from fired rules to achieve modularity as suggested in [DAY88b]. "Rule body" is expressed by an "if_clause", an optional "then_clause" and an optional "else_clause". The if_clause specifies the conditions in terms of boolean expressions which are to be evaluated when a rule is triggered; the then_clause and else_clause specify the alternative knowledge base states to be maintained or operations to be performed by the system if the IF condition is evaluated to True or False, respectively.

```
Rule rule-id
Valid_T [A, B] /* valid time interval */
Trigger-Cond (Trigger-time, Trigger-operation)

Rule body
        /* IF if_clause
           THEN then_clause
           ELSE else_clause */
End
```

Table 1: General form of a temporal rule in OSAM*/T.

The if_clause of a temporal rule can be specified by either a simple boolean expression or a guarded boolean expression. In the former case, there is only one state to be verified against a knowledge base. The result of evaluating the if_clause will be either True if the boolean expression is evaluated to true or False if the boolean expression is evaluated to false. In the later case, the guarded boolean expression has the form "IF boolean_exp1, ..., boolean_expN-1 | boolean_expN" in which boolean_exp1 to boolean_expN-1 serve as guards for boolean_expN. The result of evaluating the guarded expressions will be one of the following values: True, False, or Skip. In its evaluation, the first N-1 expressions are evaluated sequentially. If they are all true and the Nth expression is also true, the if_clause is True and the then_clause of the rule will be taken. If they are all true and the Nth expression is false, the if_clause is False and

the else_clause of the rule will be taken. If any one of the first N-1 expressions is false, the then_clause and the else_clause will be skipped (i.e., the rule will not be fired). The guarded expression allows the interdependency relationship among the guards to be explicitly specified. Also, it allows the skipping of the entire rule if any of the guards is false. We note that the semantic of the sequential evaluation of the guards and the skipping of the entire rule if any one of them is evaluated to false is quite different from logically ANDing the guards. This is because, due to the optimization of evaluating boolean expressions, the conjunction of these guards do not guarantee their sequential evaluation in a proper order. We also note that, although the semantics of a guarded expression can be specified by a nested if-then-else expression, the former is simpler and more declarative way of specifying a string of ordered conditions. As noted in [DAY88a, ULL91], the declarative property is an important feature of a high-level language.

In a temporal rule, each boolean expression consists of one or more temporal contexts connected by some Set Operators as shown in Table 2. Each temporal context (delimited by a pair of parenthesis for clarity) specifies a snapshot temporal knowledge base and is expressed by an optional WHEN clause and a CONTEXT clause. The WHEN clause specifies the scope of temporal reference defined by either an explicit time interval or a time interval defined implicitly by data reference [SU91]. If the WHEN clause is not given, the scope of the temporal reference is assumed to be the "current" knowledge base. The CONTEXT clause is used to further restrict that part of the knowledge base identified by the temporal reference of the WHEN clause. It is expressed by an "association pattern expression" which specifies that object instances of those classes in the knowledge base that satisfy certain pattern of object instance association (a linear, tree or network structure) should be retained to form a temporal sub-knowledge base. The sub-knowledge base can be further reduced by the inter-class restrictions stated in the WHERE subclause. An association pattern expression is composed of a list of identifiers (i.e., names or alias of object classes) separated by either an association operator (*) or a non-association operator (!). The association operator (*) specifies the retention of those object instances between two classes that are associated with each other. The non-association operator (!) specifies the retention of those non-associated object instances between two object classes.

```
(temporal context_1)
    Set Operator_1 (Target Classes)
(temporal context_2)
    Set Operator_2 (Target Classes)
.....

where
"temporal context_i =
 WHEN temporal condition_i
 CONTEXT association pattern
expression_i
          WHERE condition_i"
```

Table 2: The representation of a boolean expression.

For example, the association pattern expression "Teacher * Teaching * Section" (see Figure 3) describes the snapshot temporal knowledge base of a certain time which contains those Teacher instances that are associated with some Section instances (i.e., teachers who teach sections). Those teachers who do not teach (or are not associated with) any section do not satisfy the association pattern and thus are not included in the snapshot temporal knowledge base. Different from most of the existing constraint languages, association pattern expressions in this rule specification language allow very complex temporal conditions that involve multiple classes to be specified as the conditions and/or consequences of temporal rules in a relatively simple way.

Multiple temporal contexts in a boolean expression of the if_clause can be operated on by set operators such as INTERSECT, DIFFERENCE, and UNION. A set operator is always accompanied by some "target classes" which is a subset of the intersecting classes of two temporal contexts. "Target classes" is used to specify the common classes of two temporal contexts over which a set operator will be performed. For instance, the INTERSECT operator intersects two temporal contexts based on their common patterns specified in the target classes and returns a temporal sub-knowledge base which consists of both temporal contexts containing object association patterns that have the same structure over the target classes. The example shown in Figure 6 illustrates the intersecting operation between the temporal context A*B*C at T1 (see Figure 4) and the temporal context A*B*D*E at T2 (see Figure 5). After intersecting these two temporal contexts over class A, the result is a temporal sub-knowledge base consisting of both temporal contexts having those object association patterns that satisfy the intersection condition (see Figure 6). Since the retained association patterns from the two temporal contexts in the resultant temporal sub-knowledge base exist at different time points (that is, T1 ≠ T2), they are not combined. This temporal sub-knowledge base can be further processed by other operators. For instance, a temporal projection operator can be applied to the resultant temporal sub-knowledge base in Figure 6 to keep the object association patterns of a certain time interval (e.g., T1 or T2).

3.1 Temporal Rules

For the convenience of explanation and illustration, temporal rules in OSAM*/T are classified into three types: temporal state rules, temporal operational rules, and temporal deductive rules [ALA89, CHU90, SIN90, SU91]. Temporal state rules are used to specify legitimate or illegitimate states of a temporal knowledge base. A state rule verifies the state of a knowledge base but does not alter or amend the state of the knowledge base or cause any external operation to

take place. Temporal operational rules are used to perform an operation under various temporal conditions (or states). The operation specified in an operational rule will either alter the state of a knowledge base by a system-defined or user-defined operation or cause external event to occur such as triggering alarm, outputing message to a monitor, etc. by user-defined operations. Both state and operational rules are used to verify and maintain the correct state of a knowledge base according to some application constraints. Temporal deductive rules, on the other hand, are used only to deduce objects' data values and object associations which are not explicitly stored in the knowledge base.

In our KBMS implementation, we incorporate the concept and technique of "object instance binding" in the specifications of the IF clause of all three types of temporal rules. If the same class name is used in both the trigger clause and the boolean expressions in the guard of the IF clause of a rule, the multiple occurrences of the class are bound to the same object instance in the rule evaluation. If the binding restriction is not intended, the keyword CONTEXT is used in a temporal context specification of the rule body to indicate that all instances of the class should be involved in the context creation rather than just the instances involved in the trigger condition. For example, Employee appears in both the trigger condition and the rule body of Rule 00001 shown in Section 2.1, its multiple occurrences are bound to the same object instance. Thus, only the employee who is to be fired is involved in the context evaluation of IF Employee * Work_On * Project[P#=P2]. Whereas, the occurrence of Employee in the expression CONTEXT Employee * Work_On * Project[P#=P1] does not bind to the same instance. In this expression, all employees are involved in the context evaluation. We illustrate the three general types of rules below.

3.2 Temporal State Rules

A temporal state rule in OSAM*/T represents a user-defined semantic constraint. It states how the knowledge base activities should be constrained by the past, current, or future activities to ensure semantic consistency and correctness of a knowledge base. A temporal state rule usually involves the verification of more than one knowledge base state which are associated with one knowledge base activity. The knowledge base states to be verified are expressed in either simple or guarded boolean expressions.

When a knowledge base evolves due to an update, insert, or delete operation, the state of the knowledge base changes. The change will trigger relevant temporal state rules to verify the consistency and correctness of the knowledge base. If any violation against the triggered temporal state rules is detected, the system will abort the operation to avoid

the inconsistency. Temporal state rules are useful for a KBMS to verify and maintain legal knowledge base states and to automatically enforce application constraints. The users can therefore be relieved from writing tedious application programs to enforce the constraints.

A temporal state rule is given in Example 1. This rule specifies a knowledge base constraint on current employees based on their current salaries, current activities, and past activities during the time period T[06-01-82, 05-31-84]. It says that those employees who are working on project P7 with salaries greater than $60K and who had worked for TOY Department during the period T[06-01-82, 05-31-84] should also work for SALES Department. This rule was defined on January 1, 1987 with a user-defined RID "Rule 00004" and has been valid ever since; the validity of the rule is represented by the valid time interval Valid_T [01-01-87, -].

Example 1: Those employees who are working on project P7 with salaries greater than $60K and who had worked for TOY Department during the period T[06-01-82, 05-31-84] should also work for SALES Department.

```
Rule 00004
Valid_T [01-01-87, - ]
Trigger-cond (After Update(Employee.Salary))

IF
  (Employee[Salary>$60K] * Work_On * Project[P#=P7])
      INTERSECT (Employee)
  (WHEN T[06-01-82, 05-31-84]
  CONTEXT Employee * Department[Name="TOY"]) |
  Employee * Department[Name="SALES"]

ELSE Abort
End
```

Example 1: An example of a temporal state rule.

The knowledge base constraint in this example is expressed in the Trigger-cond and the if_clause and the system's response to the violation of this constraint is expressed in the else_clause. The triggering condition in Trigger-cond specifies that after updating an employee's salary (e.g., with 10% increase), the rule should be triggered to check whether the updated employee instance of the Employee class satisfy the conditions stated in the guard of the if_clause; if so, we want to make sure this employee also works for the SALES Department before the update transaction can be committed. Since we only need to check for the employee instance whose salary is being updated, the keyword CONTEXT is not used (i.e, a binding situation) for the first temporal context (as specified in the first pair of parenthesis) in the if_clause. The first temporal context statement specifies the condition for verifying whether the employee in the current knowledge base (due to the absence of a WHEN clause) works on project P7 and has salaries greater than $60K. The result from evaluating the first temporal context will be intersected with the result from the second WHEN-CONTEXT expression which returns the group of employees who had worked for TOY Department during the time

period T[06-01-82, 05-31-84]. The result of intersecting the two temporal contexts is the temporal sub-knowledge base containing the employee who is working on P7 with salaries greater than $60K and who had worked for TOY Department during T[06-01-82, 05-31-84]. This resultant temporal sub-knowledge base will be used as the guard for the evaluation of the boolean expression after the bar (|) which says that the employee in the resultant context should also work for SALES Department. If violation is detected (i.e, this qualified employee is not working for SALES Department), the rule results in False which will cause the system to abort the update operation specified in the else_clause. If no violation exists, the rule has been verified since no then_clause is given (note, the "THEN clause" could have been used in this state rule if there were more states to be verified) and the update transaction can be committed. If the Intersect operation results in an empty set (i.e., no employee is working on project P7, no employee's salary is greater than $60K, or no employee worked for TOY Department during the period T[06-01-82, 05-31-84]), the guard is false and the expression after the bar is skipped. In this situation, this rule has no effect on the update transaction and the update can be committed if no other constraint is specified.

3.3 Temporal Operational Rules

A temporal operational rule performs an operation under some data conditions. When a temporal operational rule is triggered, the data condition specified in the if_clause will be verified to determine if the action specified in the then_clause or the one in the else_clause should be performed. The operation can be either a system-defined or a user-defined operation.

A temporal operational rule is given in Example 2. This rule will be triggered after a Work_On instance is inserted. The instance is linked to an employee instance and a project instance (i.e., each instance of Work_On specifies that an employee works on a particular project.) The rule is used to ensure that those employees who work

Example 2: Those employees who work on project P5 but not P6 and who participated in project P1 during T[01-01-74, 12-31-83] should also participate in project P6.

```
Rule 00005
Valid_T [01-01-87, - ]
Trigger-cond (After InsertObject (Work_On))

IF (Employee AND (*Work_On*Project[P#=P5], !Work_On_1*Project_1[P#=P6]))
       INTERSECT (Employee)
   (WHEN T[01-01-74, 12-31-83]
   CONTEXT Employee * Work_On * Project[P#=P1])
THEN ASSOCIATE (Employee * Work_On_2 * Project[P#=P6])
End
```

Example 2: An example of a temporal operational rule.

on project P5 but not P6 and who participated in project P1 during the period T[01-01-74, 12-31-83] are involved in project P6. The AND condition following Employee specifies that the Employee instance must be associated with the project instance P5 (the associate operator *) and must not be associated with the project instance P6 (the non-associate operator !). Since we are concerned only with the employees who currently work on P5 but not P6 in this example, the keyword CONTEXT is not used in the first temporal context expression of the if_clause. Also, in the first temporal context specification, aliases for Work_On and Project (i.e., Work_On_1 and Project_1) are used to represent different scans of classes Work_On and Project, respectively. Only Work_On in the first temporal context is bound to the Work_On in the trigger clause which represents the instance just inserted. Work_On_1 ranges over all the instances of Work_On class. The rule was defined on January 1, 1987 with a user-defined RID "Rule 00005" and is still valid. Since Rule 00005 makes reference to employees' current status as well as past activities during the period T[01-01-74, 12-31-83], two temporal contexts linked by an INTERSECT operator are needed in the if_clause. The then_clause contains an operation for associating those employees, who satisfy the IF condition, with project P6. Since the operation specified in the then_clause will associate an employee to P6 through a Work_On instance which is different from the one being inserted, the alias Work_On_2 is used.

3.4 Temporal Deductive Rules

A temporal deductive rule is used to deduce some data values for an object or some object associations based on some temporal information. In a temporal deductive rule, the temporal information which will be used to deduce new fact is expressed as boolean expressions in the if_clause; the deduced data value or object association is expressed in the then_clause or else_clause. The effect of a temporal deductive rule can be on the past, current, or future state of a knowledge base.

Temporal deductive rules for deducing object's data value and object association are given in Example 3 and 4, respectively. Example 3 concludes that those employees whose salaries are greater than $60K and who ever participated in project P1 during the period T[01-01-74, 12-31-83] must have a (derived) Title "Honor Employee". Example 4 concludes that those teachers of the Electrical Engineering Department who taught courses offered by the Computer Science Department during the time period T[01-06-83, 12-23-85] must have been affiliated to both departments (i.e.,

derived object associations) in the same time period. In this example, Dept_1 (the alias of Dept) is used so that each ranges over all department instances.

Example 3: Those employees whose salaries are greater than $60K and who ever participated in project P1 during the period [01-01-74, 12-31-83] must be an Honor Employee.

```
Rule 00006
Valid_T [01-01-87, - ]
IF (CONTEXT Employee [Salary > $60K])
       INTERSECT (Employee)
   (WHEN T[01-01-74, 12-31-83]
   CONTEXT Employee * Work_On * Project[P#=P1])
THEN Employee.title := "Honor Employee"
End
```

Example 3: A deductive rule which deduces object value for the current status of a KB.

Example 4: Those teachers of the Electrical Engineering Department who taught courses offered by the Computer Science Department during the period T[01-06-83, 12-23-85] must have been affiliated to both the Electrical Engineering and Computer Science Departments then.

```
Rule 00007
Valid_T [01-01-87, - ]
IF (CONTEXT Teacher * Dept[Name="E.E."])
       INTERSECT (Teacher)
   (WHEN T[01-06-83, 12-23-85]
   CONTEXT Teacher * Teach * Course *
Dept_1[Name="C.S."])
THEN
  WHEN T[01-06-83, 12-23-85]
   Teacher AND(*Dept[Name="E.E."],
*Dept_1[Name="C.S."])
End
```

Example 4: A deductive rule which deduces object associations of past status in a KB.

4. Managing Temporal Rules

Temporal rules are defined in class definitions in the database schema as explained in Section 2. Each rule is parsed and stored in the form of a tree structure similar to a query tree. The information of the tree is then kept in a Rule Descriptor Table and a Rule Body Table of the data dictionary of the KBMS. The Rule Descriptor Table contains information of a rule such as the rule identification, the valid time, the trigger condition, the rule tree pointer, and the rule tree length, etc. The Rule Body Table stores the trees of the parsed rules. Detailed descriptions of the implementation of rules can be found in [QIU88, CHU90, SIN90].

Temporal rules of a knowledge base may have conflicts and redundancies. The validation of temporal rules is essential to ensure the consistency and correctiveness of the knowledge base. As data are entered into the knowledge base, the validated temporal rules can be used to maintain the knowledge base. When a temporal rule is updated, the rule validation process need to be performed again, and the existing data will have to be evaluated against this new rule. Rule validation is a non-trivial problem and is out of the scope of this paper. It is being studied in a separate project. In this work, we assume the validation of rules and the consistency checking of existing data with a new rule or an updated rule

have been properly done and shall limit the rest of this paper only to the management of temporal knowledge rules.

4.1 Modeling rules as objects

Temporal rules are metadata or meta knowledge of a knowledge base. They are part of the semantic properties of object instances and classes and are useful to the users of the knowledge base. For instance, in defining a new temporal rule for a system, a user may need the meta information of the existing temporal rules. Therefore, it is important and necessary for a system to model and manage temporal rules so that they can be retrieved and processed just like application data for various purposes.

In our implementation model based on C++, temporal rules are modeled as first class objects [YAS91] similar to the approach taken in [DAY88b]. Figure 7 shows part of the meta model of the system. The root of the class system is the class named OBJECT which contains all the objects of a knowledge base. E-CLASS_OBJECT and D-CLASS_OBJECT are the subclasses of the class OBJECT and are used to model system-named and self-named objects, respectively. The former models all objects of interest in an application domain whose identifiers are assigned by the system. The latter models objects named by their values which are used to describe or characterize E-CLASS and/or D-CLASS objects (e.g., integer 5, character string "John", etc.) Since the modeling constructs of OSAM*/T such as temporal rules, classes, associations, and methods are treated as system-named objects, they are modeled as subclasses of E-CLASS_OBJECT and named as TEMPORAL_RULE, CLASS, ASSOCIATION, and METHOD, respectively. The class named CLASS in the figure contains all the definitions of classes in the entire system. A class is defined by a class name, a set of temporal rules, a set of associations and a set of methods. The definitions of all the classes (entity and domain classes) shown in the figure including the class CLASS itself are instances of CLASS.

The class TEMPORAL_RULE in Figure 7 contains all the temporal rules used in the system as its instances. As object instances, they can be retrieved and processed as ordinary object instances. Each of these instances has a system-defined identifier (IID) and a user-defined rule identifier (RID) for unique identification when it is initially created. As illustrated in Figure 8, the components of a rule (e.g., RID, trigger_condition, if_clause, then_clause, etc) are modeled as descriptive attributes of the class TEMPORAL_RULE. "RID" is a user-defined identifier, "belongs_to_class" specifies the class that a temporal rule belongs to, "trigger_condition" specifies the triggering condition of a temporal rule, "if_clause" specifies the condition part of a temporal rule, "then_clause" and "else_clause" specify the consequence part of

a temporal rule. The three types of temporal rules described before (i.e. state rules, operational rules, and deductive rules) are modeled as subclasses of TEMPORAL_RULE and named as STATE_RULE, OPERATIONAL_RULE, and DEDUCTIVE_RULE, respectively.

4.2 Uniform Treatment of Temporal Data and Rules

Temporal rules define the temporal constraints of a knowledge base. These constraints can become out-of-date as a knowledge base evolves. In that case, the rules become invalid and need to be updated or deleted. The updated or deleted rules become historical rules. They are no longer valid to the current knowledge base. However, they are valid to some historical data.

A historical temporal rule and the semantic constraint it represents can be useful to some applications. For instance, a query such as "why Ritter could be a senior engineer and a sales manager at the same time five years ago?" will need both the data and the rules of "five years ago" for an adequate answer. The modification of a temporal rule in a knowledge base evolution may depend on its previous modifications. Therefore, it is important for a system to maintain the histories of temporal rules.

In OSAM*/T, we use the same object instance time-stamping technique to model and record the histories of temporal rules. Using this technique, the evolution of a temporal rule will be recorded and managed in the same way as the evolution of any ordinary application object. A temporal rule has a Start-time to indicate the time the rule becomes valid and an End-time to indicate the time before the termination of its validity. Whenever a rule evolves, a new rule is created to replace the old rule which is shifted and stored in the historical area. It is the current temporal rules which affect and constrain the current knowledge base activities such as update, delete, insert, retrieve, and user-defined operations. The historical rules, on the other hand, are needed only when the historical data are referenced. In the following, we present some examples of rule update, rule retrieval and rule evaluation.

4.2.1 Update of Temporal Rules

Updates to temporal rules are carried out by update transactions in the same manner as updating data because temporal rules are modeled as ordinary objects. An update transaction operated on a temporal rule would involve rule-modification, rule-parsing, and rule-validation before it can be committed. We use Rule 00005 as an example to illustrate the update of a temporal rule. The original version and the updated version of Rule 00005 are given below:

```
Rule 00005
Valid_T [01-01-87, 12-31-88]
Trigger-cond (After InsertInstance (Work_On))
IF      (Employee AND (*Work_On*Project[P#=P5], !Work_On_1*Project_1[P#=P6]))
              INTERSECT (Employee)
        (WHEN T[01-01-74, 12-31-83]
        CONTEXT Employee * Work_On * Project[P#=P1])
THEN ASSOCIATE (Employee * Work_On_2 * Project[P#=P6])
End
```

(a) An invalid rule to the current knowledge base.

```
Rule 00005
Valid_T [01-01-89, - ]
Trigger-cond (After InsertInstance (Work_On))
IF      (Employee AND (*Work_On*Project[P#=P5], !Work_On_1*Project_1[P#=P7]))
              INTERSECT (Employee)
        (WHEN T[01-01-74, 12-31-83]
        CONTEXT Employee * Work_On * Project[P#=P1])
THEN ASSOCIATE (Employee * Work_On_2 * Project [P#=P7])
End
```

(b) A valid rule to the current knowledge base.

In this example, Rule 00005 which was defined on 01-01-87 says that those employees who work on project P5 but not P6 and who participated in project P1 during the period T[01-01-74, 12-31-83] should also join project P6. Two years after the rule was defined, however, project P6 was completed and it is no longer meaningful to assign an employee to P6. Therefore, this rule was out-of-date and was updated on 01-01-89. The updated version says that those employees who work on project P5 but not P7 and who participated in project P1 during the period T[01-01-74, 12-31-83] should also join project P7. After this rule is updated, those qualified employees who work on project P5 will also be assigned to project P7 instead of P6. A sequence of evolutions of Rule 00005 due to "Insert", "Update" and "Delete" operations is shown in Figure 9 (a), (b), (c), and (d).

4.2.2 Retrieval of Temporal Rules

Since temporal rules are modeled as ordinary objects, they can be retrieved in the same way as other ordinary data. For example, if a user needs to see the information of the if_clause of Rule 00005 on 01-10-88, this query can be expressed as follows:

```
Q1: AT 01-10-88
    CONTEXT Temporal_Rules [RID = 00005]
    Retrieve if_clause
```

When this query is evaluated, the system will retrieve the if_clause of the old version of Rule 00005 and the answer will be the following:

```
IF  (Employee AND (*Work_On*Project[P#=P5], !Work_On_1*Project_1[P#=P6]))
        INTERSECT (Employee)
    (WHEN T[01-01-74, 12-31-83]
    CONTEXT Employee * Work_On * Project [P# = P1])
```

4.2.3 Evaluation of Temporal Rules

The evaluation of temporal rules in OSAM*/T is based on the Match-Modify-Execute (MME) cycle proposed in [RAS88] and the nested transaction model proposed in [MOS85]. A first level database transaction (i.e., update, retrieve, insert, delete, etc.) is parsed into a query tree and is matched with the trigger conditions (i.e., trigger time and trigger operation) of the rules associated with the object(s) being operated on. If there is a match between the trigger operations of these rules and the operation in the transaction, then those rules would be selected for rule evaluation. Once the "match" is successful, the original transaction is "modified" to incorporate the processing of the triggered rules (i.e., both the evaluation of the "condition" and the execution of the "action"). The processing of the triggered rules then are treated as sub-transactions which are nested under the first level transaction. It is possible that a database transaction will go through several MME cycles (i.e., several layers of nested transaction) before it commits due to the continuous triggering of temporal rules (i.e., a triggered rule triggers another rule). More detailed descriptions of the implementation and evaluation of rules can be found in [RAS88, CHU90, SIN90].

In the following, we explain simply the evaluation of temporal rules using the updated Rule 00005 as an example. When the operation of inserting an object instance into the Work_On class is detected by the system (i.e., the application is assigning an employee to a project), the updated Rule 00005 will be triggered to evaluate the boolean expression specified in the if_clause since the trigger condition is satisfied. In evaluating the boolean expression of this rule, the system will bind the inserted Work_On instance to the Work_On (but not Work_On_1) class in forming the association pattern of the context "Employee AND (* Work_On * Project[P#=P5], ! Work_On_1 * Project_1 [P#=P7])." With the binding restriction between variables in the if_clause and the trigger clause, the system will only evaluate the employee who is being assigned to a project instead of evaluating all the current employees. If the expression is evaluated to true (i.e., an employee who works on P5 but not P7 and who worked on P1 during the time period T[01-01-

74, 12-31-83]), the ASSOCIATE() operation specified in the then_clause will be activated to associate the employee with project P7. For example, if the instance W1, which assigns employee John to project P5, is entered into Work_On class. Assuming that John is not involved in project P7 and had participated in project P1 during T[01-01-74, 12-31-83], the result of evaluating the boolean expression of this rule will be the temporal sub-knowledge base shown in Figure 10(a) in which the inserted instance W1 is the only instance used in the Work_On class for forming the association pattern of the current context. Since the result shown in Figure 10(a) is non-empty (i.e., there is an employee named John who works on P5 but not on P7 and who worked on P1 during T[01-01-74, 12-31-83]), the if_clause is evaluated to true and the ASSOCIATE() operation in the then_clause will be executed. The result of executing the ASSOCIATE() operation is the establishment of the association pattern "John * W11 * P7" in the knowledge base as shown in Figure 10(b).

5. Conclusion

In this paper, we have pointed out the importance of the specification of knowledge rules which make reference to temporal events and the management of histories of these rules in a knowledge base management system. A temporal knowledge model OSAM*/T which incorporates temporal rule specifications as part of object instance and object class definitions has been presented. Temporal rules associated with superclasses can thus be inherited by subclasses just like structural associations and operations. Temporal rules can also be defined for instances of a given class, thus allowing different rules to be operated on different instances. A temporal rule specification language has also been presented. The language is characterized by its validity time specification, the trigger specification which involves time and system- and user-defined operations, and the inclusion of temporal conditions and association patterns in temporal rules. The last feature allows very complex patterns of object associations to be specified in a relatively simple way as the conditions and consequences of rules. Three general types of temporal rules are explained with examples. In this paper, we have also described the techniques of modeling and processing temporal rules as first class objects, thus the management of the histories of rules can be carried out in a uniform way as regular objects. The methods for updating, retrieving, triggering and evaluating temporal rules have also been presented. It is the authors' belief that the modeling and language concepts and implementation considerations presented in this paper will contribute to the realization of future active knowledge base management systems.

References

[ADI86] Adiba, M., and Quang, N.B., "Historical Multi-Media Databases," Proc. of the Int'l Conf. on VLDB, 1986, pp. 63-70.

[ALA89] Alashqur, A.M., S.Y.W. Su, and H. Lam, "OQL - An Object-Oriented Query Language", Proc. of the Int'l Conf. on VLDB, 1989, pp. 433-442.

[AND82] Anderson, T.L., "Modelling Time at The Conceptual Level," in Improving Database Usability and Responsiveness, P. Scheuermann (ed.), North Holland, 1982.

[ARI86] Ariav, G., "A Temporally Oriented Data Model," ACM TODS, Vol. 11, No. 4, 1986, pp. 499-527.

[BAR81] Barr, A. and Davidson, J., "Representation of Knowledge" in Handbook of AI, Barr, A. and Feigenbaum, E., (eds), William Kaufman, Los Altos, CA, 1981.

[BEE91] Beeri, C., and T. Milo, "A Model for Active Object Oriented Databases", Proc. of the Int'l Conf. on VLDB, 1991, pp. 337-349.

[CER83] Cercone, N. and McCalla G. "What Is Knowledge Representation?", Chapter 1 in The Knowledge Frontier: Essays in the Representation of Knowledge, Nick Cercone and Gordon McCalla (eds), Springer-Verlag, 1983.

[CHA90] Charkravarthy, U.S., and S. Nesson, "Making An Object-Oriented DBMS Active: Design, Implementation, and Evaluation of a Prototype", in Proc. of the Int'l Conf. on Extended Data Base Technology, March 1990.

[CHU90] Chuang, H.S., "Operational Rule Processing in A Prototype OSAM* Knowledge Base Management System", Master Thesis, Department of Computer Information Science, University of Florida, Gainesville, Florida, 1990.

[CLI83] Clifford, J., and Warren, D.S., "Formal Semantics for Time in Databases," ACM TODS, Vol.8, No.2, 1983, pp. 214-254.

[CLI85] Clifford, J., and Tansel, A.U., "On an Algebra for Historical Relational Databases: Two Views," Proc. of the ACM SIGMOD Int'l Conf., 1985, pp. 247-265.

[DAY88a] Dayal, U., "Active Database Management Systems", Proc. of the 3rd Int'l Conf. on Data and Knowledge Bases, 1988, pp. 150-169.

[DAY88b] Dayal, U., A. Buchmann, and D. McCarthy, "Rules are Objects Too: A Knowledge Model for an Active, Object-Oriented Database Management System", Proc. 2nd Int'l Workshop on Object-Orieneted Database Systems, Sept 1988.

[DIT86] Dittrich, K.R., A.M. Kotz, and J.A. Mulle, "An Event/Trigger Mecchanism to Enforce Complex Consistency Constraints in Design Databases", ACM SIGMOD record 15, No. 3, 1986, pp. 22-36.

[ELM90a] Elmasri, R., and Gene T.J. Wuu, "A Temporal Model and Query Language for ER Databases", Proc. of the IEEE Data Eng. Int'l Conf., 1990, pp. 76-83.

[ELM90b] Elmasri, R., I. El-Assal, and V. Kouramajian, "Semantics of Temporal Data in An Extended ER Model", Proc. of the ER Conf., 1990.

[GAD86] Gadia, S.K., "Toward A Multihomogeneous Model For A Temporal Database," Proc. of the IEEE Data Eng. Int'l Conf., 1986, pp. 390-397.

[GUO91] Guo, M., S.Y.W. Su, and H. Lam, "An Association Algebra for Processing Object-Oriented Databases", Proc. of the IEEE Data Eng. Int'l Conf., 1991.

[HSU88] Hsu, M., R. Ladin, and D. McCarthy, "An Execution Model for Active Database Management Systems", Proc. of the Int'l Conf. on Data and Knowledge Bases, 1988, pp. 171-179.

[KLO83] Klopprogge, M.R., and Lockemann, P.C., "Modeling Information Preserving Databases: Consequence of the Concept of Time," Proc. of the Int'l Conf. on VLDB, 1983, pp. 399-416.

[LEV87] Levesque, H.J., "A View of Knowledge Representation", in On Knowledge Base Management Systems, M.L. Brodie and J. Mylopoulos (eds), Springer-Verlag, 1987.

[LOR88] Lorentzos, N. A., and Johnson R. G., "Extending Relational Algebra to Manipulate Temporal Data." Information Systems, Vol.13, No.3, 1988, pp. 289-296.

[LUM84] Lum, V., Dadam, P., Erbe, R., Guenauer, J., and Pistor P., "Designing DBMS Support for The Temporal Dimension," Proc. of the ACM SIGMOD Int'l Conf., 1984, pp.115-130.

[MAI86] Maier, D., and J. Stein, "Development of an Object Oriented DBMS", Proc. of the OOPSLA Conf., 1986, pp. 472-482.

[MOR83] Morgenstern, M., "Active Databases as a Paradigm for Enhanced Computing Environments", Proc. of the Int'l Conf. on VLDB, 1983, pp. 34-42.

[MOS85] Moss, J.E.B., "Nested Tranasactions: An Approach to Reliable Distributed Computing," Ph.D. Dissertation, MIT. 1985.

[MYL84] Mylopoulos, J. and Levesque, H., "An Overview of Knowledge Representation", in On Conceptual Modelling: Perspectives From Artificial Intelligence, Databases and Programming Languages, Brodie, M., Mylopoulos, J., and Schmidt, J. (eds), Springer-Verlag, 1984.

[NAV89] Navathe, S. B., and Ahmed, R., "A Temporal Relational Model and A Query Language," An international Journal of Information Science Journal, Vol.48, No.2, 1989, pp. 57-73.

[NEW80] Newell, A., "The Knowledge Level", Presidential Address, American Association for Artificial Intelligence, AAAI80, Stanford University, Stanford, CA (19 august 1980), in AI Magazine, Vol. 2, summer 1981, pp. 1 - 20.

[QIU88] Qiu, J., H. Lam, and S.Y.W. Su, "The Dictionary Ssytem for OSAM* schema", Report from Database Systems Research and Development Center, University of Florida, 1988.

[RAS88] Raschid, L., and S.Y.W. Su, "A Transaction Oriented Mechanism to Control processing in a Knowledge Base Management System", Proc. of the Int'l Conf. on Expert Database Systems, 1988.

[ROS91] Rose, E., and A. Segev, "TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints," Proc. of the ER Conf., 1991, pp. 205-229.

[SCH83] Schiel, U., "An Abstract Introduction To the Temporal- Hierarchic Data Model," Proc. of the Int'l Conf. on VLDB, 1983, pp. 322-330.

[SEG87] Segev, A., and Shoshani A., "Logical Modeling of Temporal Data," Proc. of the ACM SIGMOD Int'l Conf., 1987, pp. 454-466.

[SIL90] Silberschatz, A., M. Stonebraker, and J.D. Ullman, "Database Systems: Achievements and Opportunities", ACM SIGMOD Rec., Vol. 19, No. 4, Dec. 1990, pp. 6-22.

[SIN90] Singh, M., "Transaction Oriented Rule Processing in An Object-Oriented Knowledge Base Management System", Master Thesis, Department of Electrical Engineering, University of Florida, Gainesville, Florida, 1990.

[SMI83] Smith, R.G., "STROBE: Support for Structured Object Knowledge Representation", Proc. of the IJCAI, August 1983, pp. 855-858.

[SNO85] Snodgrass, R., Ahn, I., "A Taxonomy of Time in Database" Proc. of the ACM SIGMOD Int'l Conf., 1985, pp. 236-246.

[SNO86] Snodgrass, R., "Research Concerning Time in Databases: Project Summaries," ACM SIGMOD Rec., Vol.15, No.4, 1986, pp. 19-39.

[SNO87] Snodgrass, Richard, "The Temporal Query Language TQuel," ACM TODS, Vol.12, No.2, June 1987.

[SNO90] Snodgrass, R., "Temporal Databases Status and Research Directions", ACM SIGMOD Record, Vol.19, No. 4, Dec. 1990, pp.83-89.

[STO85] Stonebraker, M., "Triggers and Inference in Database Systems", in On Knowledge Base Management Systems (Brodie and Mylopoulos, eds) Springer-Verlag.

[SU89] Su, S.Y.W., Lam, H., Krishnamurthy, V., "An Object-Oriented Semantic Association Model (OSAM*)" Chapter 17 in Artificial Intelligence: Manufacturing Theory and Practice, edited by S.T. Kumara, A.L. Soyster, and R.L. Kashyap, Published by the Institute of Industrial Engineers, Industrial Engineering and Management Press, Norcross, GA, 1989.

[SU91] Su, S. Y.W. and H.M. Chen "A Temporal Knowledge Representation Model OSAM*/T And Its Query Language OQL/T", Proc. of the Int'l Conf. on VLDB, 1991.

[SU91a] Su, S.Y.W. and A.M. Alashqur, "A Pattern-Based Constraint Specification Language for Object-Oriented Databases", Proc. of COMPCON, 1991.

[TAN86] Tansel, A. U., "Adding Time Dimension to Relational Model and Extending Relational Algebra" Information Systems, vol.11, no.4., 1986, pp.343-355.

[TAN89] Tansel, A.U., Arkun, M.E., and Ozsoyoglu, G., "Time-by-Example Query Language for Historical Databases," IEEE Trans. on Soft. Eng., Vol. 15, No. 4, 1989, pp. 464-478.

[ULL91] Ullman, J.D., "A comparison Between Deductive and Object-Oriented Database Systems", Proc. of the Int'l Conf. on Deductive and Object-Oriented Databases, 1991, pp. 263-277

[WOO83] Woods, W.A., "What's Important about Knowledge Representation?" IEEE Computer, Vol. 16, No. 10, October 1983, pp. 22-27.

[YAS91] Yaseen, R., S.Y.W. Su, and H. Lam, "An Extensible Kernel Object Management System", Proc. of the OOPSLA Conf., 1991.

Appendix:

BNF of the Knowledge Definition Language for OSAM*/T

```
schema_declaration : SCHEMA schema_name
                     domain_declaration
                     entity_declaration
                         END schema_name ;
schema_name : IDENTIFIER;
domain_declaration : DOMAIN_CLASSES
                              classname ':' data_type { ';' classname ':' data_type }
                      END DOMAIN_CLASSES;
classname : IDENTIFIER;
data_type : INTEGER
          | REAL
          | STRING
          | CHARACTER
          | BOOLEAN
          | COMPUTE;
entity_declaration : ENTITY_CLASSES
                              entity_class_block { ';' entity_class_block }
                     END ENTITY_CLASSES;
entity_class_block : ENTITY_CLASS classname
                              association_declaration
                              operation_declaration
                              temporal_rule_declaration
                     END classname ;
association_declaration : ASSOCIATION_SECTION
                                    { association_specification ';' }
                          END ASSOCIATION_DECLARATION ;
association_specification : generalization_declaration
                          | aggregation_declaration
                          | interaction_declaration
                          | crossproduct_declaration
                          | composition_declaration ;
generalization_declaration : GENERALIZATION OF '{' classname {';' classname } '}' g_constraints;
g_constraints : classname ',' classname ':' g_const {';' classname ',' classname ':' g_const };
g_const : SX
        | SI
        | SE
        | ST-SS;
aggregation_declaration : AGGREGATION OF '{' attribute_declaration { ';' attribute_declaration } '}';
attribute_declaration : attribute_name ':' classname '(' A_const ')';
attribute_name : IDENTIFIER;
A_const : OPTIONAL
        | TOTAL;
interaction_declaration : INTERACTION OF '{' classname ';' classname { ';' classname } '}' i_constraints;
i_constraints : classname ',' classname ':' cardinality { ';' classname ',' classname ':' cardinality };
cardinality : CHAR_VALUE;
crossproduct_declaration : CROSSPRODUCT OF '{' classname { ';' classname } '}';
composition_declaration : COMPOSITION OF '{' classname {';' classname } '}';
operation_declaration :
        OPERATION_SECTION
```

```
                    func_oper { ';' func_oper }
          END OPERATION_SECTION;
func_oper : ufunction
                  | uoperation;
ufunction : function_name '(' arguments ')' ':' data_type;
function_name : IDENTIFIER;
arguments : variable_name ':' classname { ';' variable_name ':' classname };
variable_name : IDENTIFIER;
uoperation : operation_name '(' arguments ')';
operation_name : dml_clause
                       | message_spec;
dml_clause : delete_clause
                    | update_clause
                    | insert_clause
                    | retrieve_clause
                    | operation4
                    | operation5;
delete_clause : DELETE_OBJECT classNlist
                      | DELETE_INSTANCE classNlist;
classNlist : '(' class_names ')'
                 | classname;
class_names : class_names ',' classname
                    | classname;
update_clause :  UPDATE '(' classname ',' attribute_value_list ')';
attribute_value_list : attribute_value_list ',' attribute_value
                           | attribute_value;
attribute_value : attribute_name '=' value_expression
                      | attribute_name comp_op value_expression;
value_expression : value_expression '+' term
                       | value_expression '-' term
                       | term;
term : term '*' factor
       | term '/' factor
       | factor;
factor : '+' primary
        | '-' primary
        | primary;
primary : attribute_specification
          | value_specification
          | '(' value_expression ')'
          | IDENTIFIER '[' search_condition ']';
attribute_specification : attribute_specification '.' IDENTIFIER
                                | IDENTIFIER;
value_specification : CHAR_VALUE
                          | NUMERIC_VALUE
                          | NULL;
search_condition : search_condition ',' boolean_term
                         | boolean_term;
boolean_term : boolean_term AND boolean_factor
                     | boolean_factor;
boolean_factor : NOT boolean_primary
                     | boolean_primary;
boolean_primary : predicate;
predicate : comparison_predicate;
```

```
comparison_predicate : comparison_argument comp_op value_expression;
comparison_argument : value_expression;
comp_op : '>'
                | '<'
                | '='
                | '!='
                | '>='
                | '<=';
insert_clause : INSERT_INSTANCE '(' target_and_source ')';
target_and_source : target ',' source;
target : classname;
source : classname;
retrieve_clause : RETRIEVE IDENTIFIER;
message_spec : MESSAGE  messagebody;
messagebody : '(' CHAR_VALUE ')';
temporal_rule_declaration :
        TEMPORAL_RULE_SECTION
                domain_rule { ';' domain_rule }
        END TEMPORAL_RULE_SECTION ;
domain_rule : RULE rule_id
                  valid_time
                  'Trigger-cond' '(' trig_conds ')'
                  rule_body
                  END;
rule_id : NUMERIC_VALUE;
valid_time: 'Valid_T' '['start_time ',' end_time']'
start_time : scalar;
scalar : NUMERIC_VALUE;
end_time : '-'
              | scalar;
trig_conds : trig_conds ',' trig_cond
                | trig_cond;
trig_cond :        option operation;
option : BEFORE
           | AFTER
           | PARALLEL
           | IMMED_AFTER;
operation : operation1
              | operation2
              | operation3
              | operation4
              | operation5;
operation1 : UPDATE '(' argument2 ')';
operation2 : ins_arg '(' argument1 ')';
ins_arg : INSERT_OBJECT
            | INSERT_INSTANCE;
operation3 : del_arg '(' argument1 ')';
del_arg : DELETE_OBJECT
            | DELETE_INSTANCE;
operation4 : RETRIEVE '(' argument2 ')';
operation5 : USER_DEFINED '(' CHAR_VALUE ')';
argument1 : argument1 ',' class_reference
                | class_reference;
argument2 : argument2 ',' attribute
```

```
                    | attribute;
attribute : class_reference '.' attribute_reference
                  | attribute_reference;
attribute_reference : IDENTIFIER;
class_reference : IDENTIFIER;
rule_body : temp_context_statement statement_list;
temp_context_statement :  | temp_subdb_definition;
temp_subdb_definition : when_clause
                          subdb_definition
                        | when_clause
                          subdb_definition
                          inter_cross_reference;
statement_list : statement_list ';' statement
                       | statement;
statement : not_exist_expression
                  | if_then_else_rule
                  | '(' search_condition ')';
not_exist_expression : NOT_EXIST temp_subdb_definition;
if_then_else_rule : IF if_part
                            THEN then_part
                          | IF if_part
                            ELSE else_part
                          | IF if_part
                            THEN then_part
                            ELSE else_part ;
if_part : guarded_boolean_exp
              | guarded_boolean_exp '|' boolean_exp;
guarded_boolean_exp : guard_exp {',' guard_exp};
guard_exp : boolean_exp;
boolean_exp : not_exist_expression
                    | temp_subdb_definition;
then_part : then_part_of_state
                  | then_part_of_transition;
then_part_of_state : not_exist_expression
                            | temp_subdb_definition
                            | cardinality_const
                            | '(' search_condition ')';
then_part_of_transition : dml_clause;
else_part : not_exist_expression
                  | temp_subdb_definition
                  | cardinality_const
                  | '(' search_condition ')';
subdb_definition : link_pattern
                       | CONTEXT link_pattern;
link_pattern : link_expr
                    | link_pattern lbranch '(' pattern_list ')'
                    | link_pattern lbranch '(' pattern_list ')' rbranch link_expr
                    | '(' pattern_list ')' rbranch link_expr;
link_expr : link_expr non_assoc_op link_term
                  | link_term;
lbranch : *AND
                | *OR
                | !AND
                | !OR;
```

```
rbranch : AND*
                | AND!
                | OR*
                | OR!;
link_term :  link_term assoc_op lfactor
                 | lfactor;
lfactor : IDENTIFIER
         | IDENTIFIER '[' search_condition ']'
         | '(' link_expr ')';
pattern_list : pattern_list ',' link_pattern
                    | link_pattern;
assoc_op : '*';
non_assoc_op : '!';
cardinality_const : MAPPING classNlist ':' classNlist cardinality;
when_clause : | when_clause_1;

when_clause_1 : WHEN references
                        | AT time_point_reference;
references : time_interval_reference
                    | data_reference;
time_interval_reference: time_int;
time_int : T '[' scalar_exp ',' scalar_exp ']';
scalar_exp : specified_time
                    | time_exp;
specified_time : scalar;
time_exp : TIME '(' time_para ')';
time_para : NOW scalar_type;
scalar_type : '+' scalar
                    | '-' scalar;
time_point_reference: scalar_exp;
data_reference : temp_func
                        | temp_func
                          WHERE boolean_condition
                        | mov_win;
boolean_condition : temp_func int_op temp_func
                            | temp_func int_op time_int
                            | time_int int_op temp_func;
temp_func : INTERVAL '(' temp_func1 ')';
temp_func1 : link_pattern
                    | temporal_functions '(' link_pattern ')';
temporal_functions : FORMER | NEXT | START | END
                             | FIRST | SECOND | THIRD | NTH | LAST
                             | B_FIRST | B_SECOND | B_THIRD | B_NTH | B_LAST;
int_op : BEFORE | AFTER | PRECEDE | FOLLOW
         | P_ADJACENT | F_ADJACENT | ADJACENT
         | P_CROSS | F_CROSS | CROSS | EQUAL
         | CONTAIN | WITHIN | O-CONTAIN | I-WITHIN
         | L-CONTAIN | L-WITHIN | R-CONTAIN | R-WITHIN;
mov_win : mov_func
           | mov_func mov_range;
           | mov_func advance_op;
           | mov_func mov_range advance_op;
mov_func : INTERVAL '(' predicate ')' move_op;
mov_op : ANY scalar time_unit
```

```
            | EVERY scalar time_unit;
mov_range: WITHIN time_int;
advance_op : INCREMENT_BY scalar time_unit;
time_unit : SEC | MIN | HOUR | DAY
              | WEEK | MONTH | YEAR;
inter_cross_reference : set_operators target_class
                                    temp_subdb_definition;
set_operators : INTERSECT
                        | DIFFERENCE
                        | UNION;
target_class : '(' class_specification ')';
class_specification : IDENTIFIER {',' IDENTIFIER};
```

Note: Figures will be available upon request!