PRISM: A KNOWLEDGE BASED SYSTEM FOR SEMANTIC INTEGRITY SPECIFICATION AND ENFORCEMENT IN DATABASE SYSTEMS

*Allan Shepherd * and Larry Kerschberg †*

Depts. of Computer Science and Management Science
University of South Carolina

ABSTRACT

This paper presents a knowledge-based approach to the specification, design, implementation, and evolution of database applications. The knowledge base consists of 1) facts regarding database objects that are organized into a hierarchy of models, and 2) rules that specify the behavior of objects within a model and among models.

The model hierarchy consists of database application data, database schemas, data model definitions, and system-related objects that control the user's interaction with the system. The rules governing the behavior of objects are specified as explicit constraints on those objects. User goals are transformed into conjectures that the inference engine must prove are satisfiable by interpreting all applicable constraints.

The semantic architecture of the PRISM system is described, together with the syntax and semantics of the constraint language. PRISM is implemented in the C programming language and runs under the UNIX ** operating system.

** UNIX is a trademark of AT&T Bell Laboratories.

* Current Address: Hewlett-Packard, 3L, 1501 Page Mill Rd., Palo Alto, CA 94304.

† College of Business Administration, University of South Carolina, Columbia, SC 29208.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1984 ACM 0-89791-128-8/84/006/0307 $00.75

1. INTRODUCTION

This research is motivated by the need for improved tools and techniques for the specification, design, implementation, and evolution of database applications. It draws upon ideas from semantic data models, data dictionaries, specification-based system methodologies, and artificial intelligence.

Semantic data models such as RM/T [8], the Functional Data Model (FDM) [15,18,30,31], and Abrial's Binary Model [1], capture an application's meaning by making extensive use of metadata – data about data – to specify both intra- and inter-entity semantics. In fact, model metadata should be considered part of the model. Data models represent application semantics by means of their data structures, operations, and constraints. Constraints not expressible in terms of the data model are represented as procedures in application programs. Thus, the semantics of an application are expressed in distinct representations, i.e., data structures, operations, metadata, constraints, and application programs.

Conventional Database Management Systems (DBMSs) store metadata in data dictionaries that provide limited data structuring capabilities. However, the Database Directions III Workshop report [13] on Information Resource Dictionary Systems recommends that future dictionary systems provide facilities to 1) make metadata more accessible to users, and 2) to allow metadata to be queried and manipulated in the same manner as application data.

Database applications are not static; rather, they *evolve* over time to meet the changing information requirements of users. Indeed, a database application's

implementation may itself be a powerful force in changing users' information requirements. Balzer [2] argues that *specification-based* systems allow for the creation, verification, execution, and evolution of a conceptual model. In order to implement specification-based database systems, we must make better use of metadata, and provide a uniform formalism to express database objects, operations and constraints.

Brodie [6] has discussed the desirable features of a database specification tool, and many of these are incorporated into the PRISM (constraint-based PRototyping Information System Manager) system [29]. In particular, the PRISM formalism 1) incorporates static as well as dynamic properties of database objects, 2) provides access to system metatypes for system evolution, 3) supports the integration of modeling abstractions such as abstract data types, generalization hierarchies, views, and control abstractions, and 4) allows the specification of modeling methodologies.

To satisfy the above requirements, we have chosen a knowledge-based architecture for PRISM. In expert systems there is a distinct separation between the *knowledge base*, consisting of facts and rules governing those facts, and the *inference engine* that reasons about the knowledge not only to create new facts and rules, but also to provide expert advice to users. A very important aspect of an expert system is the *explanation* facility which explains the system's line of reasoning to the user.

This paper makes several contributions to database management. The knowledge-based approach allows both data and metadata to be treated in a *uniform* fashion. Explicit constraints on objects can be used to specify their behavior at all levels, from application data to system concepts. Finally, *tools* for database design, transaction specification, schema analysis, and data translation can be viewed as *specialized interpreters* that access the knowledge base.

The remainder of the paper has the following organization. Section 2 presents the PRISM knowledge base architecture, its associative net, model hierarchy and constraint language. The PRISM modeling environment and user facilities are presented in Section 3, and our conclusions appear in Section 4.

2. KNOWLEDGE BASE ARCHITECTURE IN PRISM

In PRISM, the specifications for all levels of an information system are stored in the Dictionary Subsystem (DS). As shown in Figure 1, the Dictionary Subsystem consists of a Knowledge Base (KB) and an Inference Engine.

The PRISM Knowledge Base contains *facts* (database instances or tokens, database schemas, data model definitions, and system-level concepts) and *rules* (explicitly-specified constraints regarding facts at all system levels). The Modeling Specialist provides a user-interface to the Knowledge Base and Inference Engine.

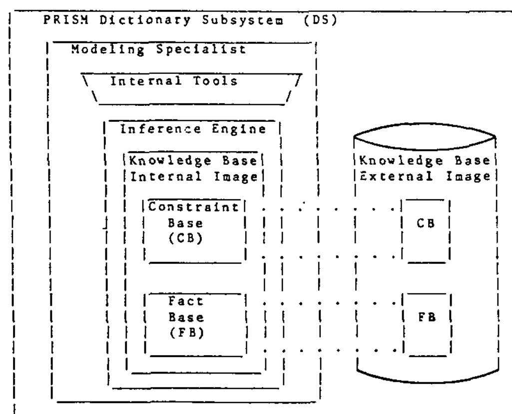

Fig. 1. The PRISM Architecture

The Inference Engine is used to understand and control system behavior, and supports updates to and queries on the Knowledge Base, subject to satisfying the constraints.

A user query is transformed by the Modeling Specialist into a conjecture for the inference engine to prove. The engine, in turn, interprets all applicable constraints on affected database objects, and if these are satisfied then new knowledge (facts and constraints) is added to the knowledge base. Direct interpretation of each explicit constraint, when the object it applies to is referenced, provides maximum flexibility to assist the user in incrementally adding and testing new constraint

specifications. Inherent access and evaluation constraints within the internal tools enforce the implicit and explicit constraints in the KB. The Constraint Base is in a globally consistent state before and after an "update transaction." If at some point in the evaluation, a constraint is violated, then the transaction fails and the update is not committed.

2.1. Knowledge Base Structure

The Knowledge Base consists of two sections, an intension and an extension. The intension, called the Constraint Base (CB), is a dictionary of explicit constraints, which are specified in the PRISM Constraint Language (CL). Constraints and meta-constraints for all information system levels, including the metadata of PRISM itself, are stored in CB, for dynamic update by users. The extension is a Fact Base (FB) of tokens.

The Knowledge Base has both an external and an internal *image*. In the external image, constraints are expressed explicitly in the PRISM constraint language, CL. And the facts, i.e., instances of the abstract objects specified in CL, are stored in a conventional Database Management System. Thus the Constraint Base can be viewed as resting on top of a conventional DBMS, and specifying semantics for the interpretation of the facts in the database. The internal image represents the objects specified in the constraints in an associative net.

2.1.1. The PRISM Associative Net

Object-oriented systems, which attach declarative representations of knowledge to objects, allow dynamic growth of information systems. Objects can in general be entities or relationships between entities. This "duality" between relationships and entities allows for the uniform treatment of both data and metadata, as will be discussed shortly.

Complete typing of objects in an information system can support logical correctness inferencing for *object instantiation*, i.e., testing whether an object, which may be newly created, satisfies the properties of the type of which it is an instance [25]. Modeling tools using such inferencing have been described [3,4,22].

Object-oriented models share the goal of *logical data independence* with database management systems. Object-oriented models can provide logical data independence between applications and the conceptual knowledge base by encapsulating application-oriented behavior into Abstract Operation Types (AOT's) which are themselves objects. For example, an AOT called MakeReservation might model an *association* (aggregation) of component objects such as Person, Hotel, Reservationist, together with the required constraints to allow a person to make a reservation at a hotel. Thus, each instance of MakeReservation would be represented as an object.

The classification of objects in a hierarchy of types has wide acceptance as a strategy to manage the complexity of information system design. A *generalization hierarchy*, which indexes property inheritance on objects, allows effective support of model definition methodologies. The subtype-to-type *generalization* mapping [32] can be represented as links between type objects in an associative net [24,12,16]. Such a knowledge representation provides efficient indexing of type properties on instances for information systems which have a relatively large number of object types and small number of data items, e.g., TAXIS [23].

Inheritance complexity in the Dictionary Subsystem is limited by allowing only 'asa' links, 'isa' links and 'constraint-on' links for constraint indexing on objects. 'Constraint_on' links simply allow direct retrieval of explicit constraints for an object. An 'asa' link which connects an instance to a type, allows constraints on type objects to be applied to the instances of the type. In order to exist explicitly, every object must be linked to a type by an instance of the 'asa' mapping. An 'isa' link from a type to a supertype, allows constraints on supertype objects to be applied to the instances of the type. The constraints associated with the instances of a type are expressed as "instance of" constraints on the type, and those that refer to the properties of the type itself are "instance of" constraints on the meta-type (at the next higher level) in the type hierarchy.

However, not every type must be a subtype of a type. An 'isa' related type does not necessarily have a single supertype, nor does an 'isa' related subtype have to disjointly partition the instances of the supertype

from other subtypes of the supertype. These properties of subtypes may be specified as constraints on the meta-types of a data model or database schema types.

The 'asa' and 'isa' associations are available directly in the PRISM Constraint Language (CL) as primitive predicates to express property abstraction. For example, the predicate asa(x,man), read "x as a man", specifies that 'x' is an instance of the type 'man'. The 'isa' and 'asa' CL predicates evaluate to TRUE if the equivalent link can be propagated successfully in the KB internal image.

The PRISM Model Hierarchy

In PRISM, an information system is represented as a hierarchy of object-oriented models. The top level consists of PRISM metadata, that is, the ***primitive concepts*** available to users to define successively lower model levels. At the top level are such concepts as built-in types (integer, dollar, date, etc.), dictionary types (dictobj, usrobj, dict_name), and modeling primitives (class, map, isa, asa, domainof, rangeof, etc.) The other levels are the Data Model, Schema, and Application levels.

Objects at each level are defined in terms of the metadata of the level above! For example, metadata for a data model is expressed in terms of PRISM modeling primitives, and defines the concepts and rules for that data model, e.g., the Relational Data Model [7]. Schema metadata consists of the data definition facilities provided by a particular data model. Applications metadata consists of the database types defined in a schema according to a data model, for example, a relational schema for a sales order processing application.

An example of a vertical slice through an information system is shown in Figure 2. Ellipses represent objects. Double-lined arcs represent 'asa' links. Only significant 'asa' links are shown. Single-lined arrows with tail represent 'isa' links. Notice that this information system models, at the two highest levels, PRISM metadata and types associated with the Functional Data Model. The ***schema diagram*** within the dotted region in Figure 2 shows a database schema defined in the graphical notation of the Functional Data Model, which consists of entity_sets 'order' and 'employee' and functions

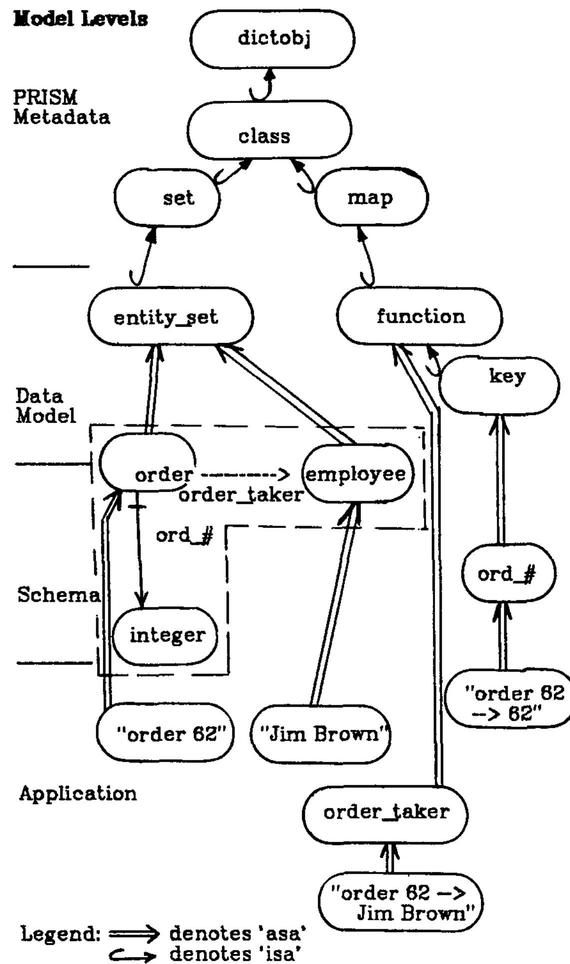

Fig. 2. PRISM Model Hierarchy – Metadata and Data

order_taker: order --> employee and ord_#: order +-> integer.

The function 'ord_#' serves as a ***key*** of the 'order' entity_set. This is denoted by the bar on the function arrow. This figure shows the ***duality*** of representing relationships (functions) as objects. For example, ord_# and order_taker are clearly functions at the Schema Level and also instances of the meta-type 'function' at the Data Model level. Also, the token "order 62 --> 62" represents the mapping under ord_# of order 62 to the integer 62.

2.2. Constraint Specification and Evaluation in PRISM

Constraints can be classified as being inherent, explicit, or implicit [5]. Inherent constraints are those which are an integral part of a data model, e.g., the parent-child relationship in the hierarchical model

inherently supports a functional map from children to parent segments. Explicit constraints are those that can be expressed independently of data model structures in some formalism such as the predicate calculus. Finally, implicit constraints are those derivable from the inherent and explicit constraints.

Several constraint formalisms have been proposed. Brodie has introduced a 'limited generic data model' [5]. The model has a data type algebra and a schema specification language, named BETA, to support a methodology for denotational constraint management. Hammer and McLeod define constraints as invariance assertions on a database [14] and propose a semantic integrity subsystem.

Deutsch describes a constraint formalism to represent both control and data abstractions uniformly [11]. A number of interesting questions are raised by Deutsch, including: 1) how constraints can represent operational abstractions, 2) how constraints can synthesize different supertypes' properties, and 3) whether constraints are a good framework for expressing queries and/or integrity constraints on databases.

In our research we have found that 1) constraints can represent operational abstractions through Abstract Operation Types and their associated constraints, 2) the associative net defines the subtype-supertype relationships with constraints modeling type properties, and 3) constraints can be used to create user views, query a database, and specify semantic integrity.

PRISM employs a generic constraint language which is applicable to all information system levels, as compared to RAISIN (Rules from Artificial Intelligence Specified for Ingres) [34], which handles only one level. Also, a unified constraint manager in PRISM handles all system levels. This provides a basis for correctness inferencing among the different levels of metadata.

Constraints in the Knowledge Base which specify control abstractions are *active* knowledge sources, that is, the control intelligence resides in the constraint specifications. For example, when an object of type 'session' (cf. Example 3 below) is instantiated, the constraint specification for a session may require 1) a login consisting of valid user name, 2) authorized access to an information system, and 3) a collection of transactions.

Deliyanni and Kowalski provide an excellent comparison of various 'extended semantic net' representations as a dual of clausal predicate logic [10]. If constraint semantics are restricted such that constraint instances indexed on objects are well formed formulae (wff) in a logic, e.g., first-order logic, interpreters can use logical operations to perform inferencing to extend the fact base of the system in information retrieval [28, 25] and problem solving.

Propositions in semantic nets have parallels in functional and multivalued dependencies in logical database schemas. The direct equivalence of logical data dependencies in the Relational Model to some predicate calculus expressions has been demonstrated and used in database design [27,35].

In PRISM the enforcement of the relationships specified in constraints is comparable to the propagation of existence dependencies of mappings in database schemas. Using constraint propositions within semantic nets to represent database dependencies allows a natural clustering of related concepts and an associated indexing of those constraints.

2.2.1. The PRISM Constraint Language

The Constraint Language (CL) inherits structural characteristics from specification languages like SPECIAL [26] and Ina Jo [21], artificial intelligence languages such as PLANNER [17], the Functional Data Model specifications [31,15], and from ADT specifications as applied to database specifications [19].

A CL constraint consists of a collection of rules. Each rule consists of a precondition, action and postcondition sequence. Within each precondition and postcondition, predicates are combined with the logical operators AND, OR, NOT and parentheses. Predicates name CL metadata query functions. The optional action statement contains a sequence of simple actions which name CL atomic update procedures.

In PRISM a user request (or goal) is either to ASSERT, DENY or TEST an information system fact. User goals are determined in the precondition clause. Example 1 shows an 'instance' constraint for an object named

'invoice'. The '$' prefix denotes a variable.

Example 1.

```
CONSTRAINT: instance of: invoice;
PARMS:        $invoice_instance;
PRECONDITION:    roleis(ASSERT);
ACTION:        unitasa: $invoice_instance, invoice;
POSTCONDITION:    assert(
 invoice_number($invoice_instance,$integer_instance));
```

An 'instance' constraint specifies the conditions for a user to create or update instances of a type. A CL constraint bundles *functional rules* that specify the semantics of objects. Example 1 has a single rule which says that "To assert the existence of an instance of type 'invoice', the instance must have associated with it an invoice number." The 'invoice_number' object is a mapping between 'invoice' and an integer id-number, that is,

invoice_number: invoice -> integer.

A CL constraint for 'invoice_number' is shown in Example 2. This constraint specifies that two actions will occur: a unitasa and a unitmap only if the system can obtain an integer invoice number from the user, and $inv_num_inst is related to precisely one invoice.

Example 2.

```
CONSTRAINT: instance of: invoice_number;
PARMS:        $inv_inst, $integer_inst;
PRECONDITION:    roleis(ASSERT) AND
          NOT iscurrinstance( $integer_inst);
ACTION:  unitasa: $inv_num_inst, invoice_number;
          unitmap: $inv_num_inst, $inv_inst, $integer_inst;
POSTCONDITION:    pmssg ("Enter invoice number") AND
          assert (integer($integer_inst) ) AND
          mapcardinality ($inv_num_inst,"1:1");
```

2.2.2. Semantics of CL

To determine whether a constraint is satisfied, its logical value is computed to TRUE, FALSE, UNKNOWN, or EXCEPTION as the *conjunction* of the value of each rule in the constraint.

To determine the value of each rule, the logical value of a rule's precondition is first determined. Predicates in preconditions are typically CL query functions on the Fact Base or on environment parameters. If a rule's precondition evaluates to TRUE, the rule fires and its ACTION statement is triggered, then the postcondition is evaluated. The logical value of a fired *rule* is the value to which the postcondition expression evaluates. If a precondition evaluates to FALSE, the rule is not applicable and the value of the rule is *TRUE*, i.e., it does not affect the value of the constraint. Neither the action nor postcondition are evaluated in this case. If any predicate or action evaluates to EXCEPTION, evaluation of the constraint terminates with a value of EXCEPTION. Expressions which evaluate to UNKNOWN combine with logical operators as follows:

```
TRUE    OR  UNKNOWN   ->  TRUE
FALSE   OR  UNKNOWN   ->  UNKNOWN
TRUE    AND UNKNOWN   ->  UNKNOWN
FALSE   AND UNKNOWN   ->  FALSE
```

A user goal is handled as a transaction that is not committed until all subgoals are proved to be correct. Actions, which are derived principally from the update semantics of the Functional Data Model [15], specify atomic update operations on instances in the Knowledge Base. Atomic updates are only committed, i.e., updated instances moved into the Fact Base, if assert, deny, and test subgoals in postconditions are satisfied. In addition to create, delete and update rules, ordering rules may be added to instantiation constraints such that the creation or deletion of instances of the type propagates the ordering. Security constraints and access constraints are evaluated prior to instance constraints.

The syntax and semantics of CL constraints ensure that an interpreter can complete an evaluation of rules sequentially to derive a logical value for a constraint.

3. THE PRISM MODELING ENVIRONMENT

An *initial set* of constraint specifications loaded by PRISM, when accessed via the Modeling Specialist, provides a user view of an information system prototyping and modeling tool. The current implementation of PRISM has over 100 constraints specified. PRISM is implemented in the C Programming Language and runs under the UNIX operating system. The user, with this innate but redefinable *knowledge core*, is able to incrementally define and test specifications of information system objects.

The following facilities are provided in the initial modeling image: 1) pragmatic types: session, transaction, login, username, timestamp, etc., 2) Dictionary Subsystem queries to display defined KB object names, the Inference Engine state, etc., 3) trace facilities for goal satisfaction, conjecture evaluation and constraint activation, and 4) built-in types: integer, dollar, date, alpstring, goal, etc.

If a user defines an information model using the Modeling Specialist to update and populate these types, the model will be a *consistent extension* of the initial knowledge core model. For example, in designing the PRISM user-interface, we used the PRISM metatypes to develop the notion of a 'session' which allows the Modeling Specialist to control interactions with the Inference Engine.

Example 3 below shows the CL constraint for 'session.' Evaluation of the 'session' constraint causes propagation of the mappings shown in Figure 3 through the enforcement of constraints on the objects named in the schema. Figure 3 depicts the object relationships in terms of the Functional Model. The double-headed arrows denote set-valued mappings. Recall the duality of functions (relationships) and objects in PRISM, as discussed in Figure 2.

Example 3:

```
CONSTRAINT: instance of: session;
PARMS:        $tempsess;
PRECONDITION:    roleis(ASSERT);
ACTION:       unitasa: $tempsess, session;
        unitasa: $infosys,infosys;
        unitasa: $usr,user;
POSTCONDITION:   assert(login( $tempsess, $usr))     AND
        assert(access($tempsess,$infosys)) AND
        assert(validusr($infosys, $usr))   AND
                 /* validate user */
    REPEAT: assert ( sess_transaction ($tempsess)) AND
        WHEN: NOT ask ("More transactions ?") LEAVE;
    ENDREPEAT;
        asa($tempsess, dictobj);
```

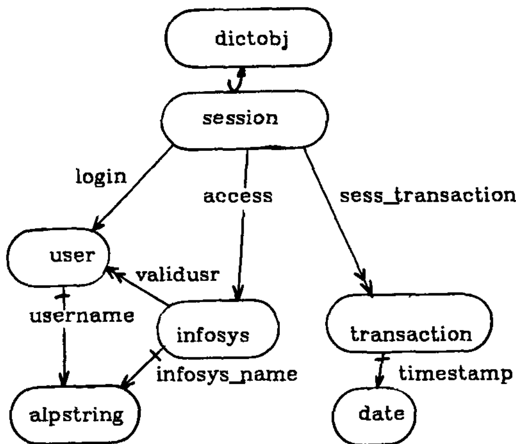

Fig. 3. Functional Model Representation of Session Constraint

A 'session' instance will only be added to the dictionary after a user has logged in, has accessed an information system 'infosys' with unique 'infosys_name' for which he is a 'validusr', and has requested at least one 'transaction' which must be timestamped with the 'date'. Thus the instantiation of a 'session' becomes a generalized transaction involving the objects mentioned in the constraint.

3.1. The Modeling Specialist

The Modeling Specialist enables the user to instantiate objects within a defined information system (or model), to query the facts regarding an information system, and to extend a model level within the PRISM model hierarchy. Transactions that seek to update models are only committed if constraints on objects affected by the transaction have been satisfied by the Inference Engine.

To test a model, the PRISM Modeling Specialist formalizes the object descriptions input by the user into a set of conjectures. If a conjecture has been shown by the Dictionary Subsystem reasoning tools to be consistent with the current Knowledge Base, the definitions of objects referenced in the conjecture are updated in the Knowledge Base, effectively updating the information model.

A model is determined to be inconsistent or incomplete when a reference to an undefined object is found and a definition for it cannot be obtained. A conjecture is not accepted if constraints that apply to an object touched by the conjecture cannot be satisfied.

3.1.1. Starting A User Interaction

To start a simple interaction with PRISM, the user passes the name of a type object to be instantiated to the Modeling Specialist. The default entry point assumes that the user's goal is to *instantiate* the 'session' object. The Modeling Specialist gets a *goal* from the user (or a subgoal from the Inference Engine), and creates a new environment within which parameters that indicate the goal's propagation purpose are set. Each environment inherits the original Knowledge Base, the facts tentatively established in descendent environments, and the constraints from antecedent environments. If references to objects in the environment are satisfied, the antecedent environment can treat the goal as reached.

For example, the goal or subgoal "assert(airport($x))", can be reached if the conjecture

that there exists an airport, $x, can be asserted to be true. A formal conjecture is built for each goal. In this example, the conjecture would be "asa($x, airport)". The user's purpose in this example is to assert that the conjecture is true and to make whatever updates to the Fact Base may be necessary to satisfy the conjecture. However, updates may not satisfy the constraints in the Constraint Base and thus the assertion may fail.

3.1.2. Conjecture Evaluation

Evaluation of a conjecture is done by the Inference Engine. The result of conjecture evaluation is a value in the four-valued logic employed for the Constraint Language. If a conjecture is satisfied by the Inference Engine, the implications are incorporated into the environment to satisfy the goal. For instance, objects marked as deleted are purged from the Knowledge Base. Objects marked by tests are *accounted for*, for instance, are retrieved into query result views, or recorded in an access log. In the case of new properties of pre-existing objects, the new properties are *combined* with the properties defined on the object in the antecedent environment. For example, the specialization of a type, if permitted, adds a new constraint on the type.

As in the evaluation of a goal, each subgoal is transformed into a sub-conjecture within an environment, and the Inference Engine invoked to satisfy the sub-conjecture. The goal and conjecture satisfaction processes are thus recursive, and use a depth-first search strategy. Since constraint propagation requires that most conjecture subtrees be completely traversed, this is reasonably efficient.

4. CONCLUSIONS

This paper has presented a knowledge-based approach to semantic integrity specification and enforcement in database systems. This approach has several distinct advantages over conventional architectures: 1) the knowledge base draws together all constraint specifications in *explicit* form so that they may be examined by the user community and PRISM tools, 2) the associative net organization of type objects provides a uniform treatment of data and metadata – from database instances through system control abstractions – so that users can not only understand the system but also *extend* it to suit their needs, 3) the inference engine may be used to validate system behavior by proving conjectures about the knowledge base in response to user goals, and 4) new knowledge, e.g., facts and constraints, is added incrementally based on the inference engine's success in verifying the conjectures.

User access to and control of metadata implies that user views of the knowledge base could be used to specify methodologies for database design, to control access to database applications, and to incorporate new 'domain' experts, such as a 'Time Expert,' into the system.

Research is needed into *constraint management* tools to allow users to specify constraints, to determine when constraints may lead to conflicts, and to compute a *minimal* set of constraints for an information system.

5. REFERENCES

1. Abrial, J.P., Data Semantics, *Database Management*, eds. J. W. Kimbie & K. L. Kofferman, North Holland, 74.
2. Balzer, R, M., Dynamic System Specification, *Proc. Workshop on Data Abstraction, ACM SIGMOD Record*, Feb. 1981, Pingree Park, Colorado, June 1980, pp 95-97.
3. Borgida, A., Greenspan, S., Data and Activities: Exploiting Hierarchies of Classes, *Proc. Workshop on Data Abstraction, ACM SIGMOD Record*, Feb. 1981, Pingree Park, Colorado, June 1980, pp 98-100.
4. Borgida, A. T., Mylopoulos, J., Wong, H. K. T., Methodological & Computer Aids for Interactive Info. Sys. Design, *IFIP Working Conf. on Automated Tools for Information Systems Design & Dev.*, New Orleans, Jan., 1982.
5. Brodie, M. L., Specification and Verification of Database Semantic Integrity, Ph. D. Dissertation, *Tech. Report* CSRG-91, Univ. Toronto, April 1978.
6. Brodie M. L., Research Issues in Database Specifications, *ACM SIGMOD Record*, 13, 3, April 1983, pp 42-45.
7. Codd, E. F., A Relational Model of Data for Large Shared Data Banks, *Comm. ACM*, 13, 6, June 1970, pp 377-387.
8. Codd, E. F., Extending the Database Relational Model to Capture More Meaning, *ACM TODS* 4, 4, December 1979, pp 397-434.
9. Davis, R., Content Reference: Reasoning About Rules, *Artificial Intelligence*, 15, 3, pp 223-239.
10. Deliyanni, A., Kowalski, R. A., Logic and Semantic Networks, *Comm. ACM*, 22, 3, March 1979, pp 184-192.

11. Deutsch, L. P., Constraints: A Uniform Model for Data and Control, *Proc. Workshop on Data Abstraction, ACM SIGMOD Record*, Feb. 1981, Pingree Park, Colorado, June, 1980.
12. Findler, N. (editor) *Associative Networks: Representation and Use of Knowledge by Computer*, Academic Press, NY, 1979.
13. Goldfine, A. (editor) *Information Resource Management - Strategies and Tools*, Database Directions III Workshop Report, NBS Special Publication 500-92, 1982.
14. Hammer, M., McLeod, D., A Framework for Database Semantic Integrity, *Proc. 2nd Int. Conf. on Software Engineering*, San Francisco, 1976.
15. Hecht, M. S., Kerschberg, L., Update Semantics for the Functional Data Model, *Database Research Report*, No. 4, Bell Labs., Holmdel, New Jersey, 1981.
16. Hendrix, G. G., Encoding Knowledge in Partitioned Networks, in *Associative Networks*, ed. Findler, N., Academic Press, NY, 1979, pp 51-92.
17. Hewitt, C., PLANNER, A Language for Proving Theorems in Robots, *Proc. IJC Artificial Intelligence*, 2, 1971.
18. Kerschberg, L., Pacheco, J. E., A Functional Data Base Model, *Tech. Report*, Pontificia Univ. Catolica, Rio de Janeiro, Jan. 1976.
19. Leveson, N. G., Applying Behavioral Abstraction to Information System Design and Integrity, *Tech. Report #47*, Lab. of Medical Information Science, Univ. Cal., San Francisco, March 1980.
20. Levesque, H. J., Mylopoulos, J., A Procedural Semantics for Semantic Networks, in *Associative Networks*, ed. Findler, N., Academic Press, NY, 1979, pp 93-120.
21. Locasso, R., Scheid, J., Schorre, V., Eggert, P., *The Ina Jo Specification Language Reference Manual*, TM-(L)-6021/001/00, System Development Corp., Santa Monica, CA., Jan. 1980.
22. Lundberg, B., IMT - Information Modeling Tool, *Automated Tools for Information Systems Design*, eds. H-J. Schneider and A. I. Wasserman, North Holland, 1982, pp 21-30.
23. Mylopoulos, J., Bernstein, P. A, Wong, H. K. T., A Language Facility for Designing Database-Intensive Applications, *ACM TODS*, 1, pp 185-207.
24. Quillian, M. R., Semantic Memory, in *Semantic Information Processing*, ed. Minsky, M., MIT Press, pp 216-270.
25. Reiter, R., On the Integrity of Typed First Order Data Bases, in *Advances in Data Base Theory*, Plenum Press, 1981, pp 137-157.
26. Robinson, L., Roubin, O., *SPECIAL - A Specification and Assertion Language*, SRI, Menlo Park, CA., Jan. 1977.
27. Sagiv, Y., et al, An Equivalence Between Relational Database Dependencies, and a Fragment of Propositional Logic, *Journal ACM*, 28, 3, July 1981, pp 435-453.
28. Schubert, L., Extending the Expressive Power of Semantic Nets, *Artificial Intelligence*, 7, pp 163-198.
29. Shepherd, A. W., PRISM: A Constraint-Based Prototyping Information System Manager, *MS Thesis*, Dept. Computer Science, Univ. South Carolina, 1983.
30. Shipman, D., The Functional Data Model and the Data Language DAPLEX, *ACM Trans. Database Syst.* 6, 1 (March 1981) pp 140-173.
31. Sibley, E.H., Kerschberg, L. Data Model and Data Architecture Considerations, *Proc. National Computer Conference*, AFIPS Press (June 1977) pp 85-96.
32. Smith, J. M., and Smith, D. C. P., Database Abstractions: Aggregation and Generalization, *ACM Trans. Database Syst.* 2, pp 105-133.
33. Stonebraker, M., Wong, E., Kreps, P., Held, G., The Design and Implementation of INGRES, *ACM TODS*, 2, 3, Sept. 1976.
34. Stonebraker, M., Application of Artificial Intelligence Techniques to Database Systems, *Mem UCB/ERL* M82/31, Berkeley, May 1982.
35. Zaniolo, C., Melkanoff, M. A., On the Design of Relational Schemata, *ACM TODS*, 6, 1, March 1981, pp 1-47.