ADAPTING DATABASE IMPLEMENTATION TECHNIQUES TO MANAGE VERY LARGE KNOWLEDGE BASES

John Mylopoulos, Vinay K. Chaudhri,
Dimitris Plexousakis and Thodoros Topaloglou
Department of Computer Science, University of Toronto
6 King's College Road, Toronto, Canada M5S 1A4

ABSTRACT
The management of very large knowledge bases presupposes efficient and robust implementation techniques, sophisticated user interfaces and tools to support knowledge acquisition, validation and evolution. This paper examines the problem of efficiently implementing a knowledge base management system by adopting database techniques. In particular, the paper describes algorithms for designing logical and physical storage schemes and for processing efficiently queries with respect to a given knowledge base. In addition, the paper offers an overview of a new concurrency control algorithm which exploits knowledge base structure to support efficient multi-user access. Finally, rule and constraint management is discussed and a comprehensive scheme for compiling and processing them is presented. Throughout, the paper sketches algorithms, presents some formally proven properties of these algorithms and discusses performance results. The research presented in this paper was conducted at the University of Toronto for a project titled "The Telos Knowledge Base Management System".[1]

1. INTRODUCTION

The craft of constructing and managing very large knowledge bases is expected to become an ever more important activity to be supported by the computing environments of the future. Such knowledge bases will be used as components of expert systems, but also as knowledge repositories serving large user communities within an organization. Building such knowledge bases requires knowledge base management facilities, ranging from robust and efficient implementations to sophisticated user interfaces and tools for knowledge acquisition, validation and knowledge base evolution. This paper proposes an architecture for a prototype knowledge base management system (hereafter KBMS) under development at the University of Toronto [Mylopoulos92] and offers an overview of some of the technical issues that have been addressed. In particular, the paper discusses (a) physical storage management and query processing, assuming that the knowledge base is stored on secondary storage, (b) concurrency control for efficient multi-user access of knowledge bases, (c) efficient compilation, simplification and evaluation techniques for deductive rules and integrity constraints. Throughout, the paper sketches algorithms, presents some formally proven properties of these algorithms and discusses performance results, where available. A more detailed description of the proposed architecture can be found in [Mylopoulos93]. The KBMS prototype under design adopts the language Telos [Mylopoulos90], an object-centered knowledge representation language supporting a number of structuring mechanisms, a declarative sub-language for representing deductive rules and integrity constraints as well as facilities for representing and reasoning with temporal knowledge.

Section 2 of the paper presents a quick overview of Telos and introduces an example to be used throughout. Section 3 introduces the overall architecture of the KBMS, while sections 4, 5 and 6 describe respectively physical storage management and query processing algorithms, concurrency control policies as well as methods for the compilation and simplification of rules and constraints. Finally, section 7 summarizes the results of this paper and points to directions for further research.

2. A BRIEF OVERVIEW OF TELOS

Telos [Mylopoulos90] is an object-centered knowledge representation language offering structuring mechanisms analogous to those supported by semantic networks and semantic data models as well as an assertional sublanguage for the expression of deductive rules and integrity constraints. Two novel aspects of Telos are its treatment of attributes as first-class citizens and the provision of special representational and inferential capabilities for temporal knowledge. Structural and assertional knowledge is organized along a *classification*, *generalization* and *attribution* dimension. Moreover, the language encourages the modelling of dynamic aspects of an application in terms of two temporal dimensions for representing *historical* knowledge of the application, such as "Maria was born on May 29, 1976" and *belief* history

---

[1] This paper is an extended and updated version of a paper titled "*A Performance-Oriented Approach to Knowledge Base Managemen*t " which was presented by the authors at the 1st International Conference on Information and Knowledge Management in Baltimore, MD. , November 1992.

knowledge such as "The system was told that Maria was born on May 29, 1976 on September 27, 1993". To support the representation of temporal knowledge, Telos adopts Allen's [Allen81] interval-based temporal representation framework. Telos has a well-defined semantics based on a possible-worlds model [Plexousakis93a]. An example knowledge base is shown in figure 1 in the form of a semantic network. Dashed lines represent generalization (is-a) relationships between generic entities (*classes*), shown in bold font, whereas solid lines represent binary relationships among entities (*attributes*).

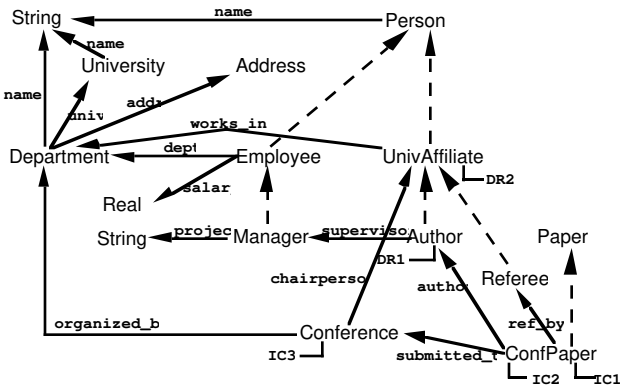

Figure 1: An Example Telos Knowledge Base

Integrity constraints and deductive rules are attached to classes through attributes. For instance, the constraint `IC2` is attached to the class `ConfPaper` and expresses the property that "`no author of a conference paper can serve as its referee`" Likewise, the deductive rule `DR1` is attached to the class `Author` and expresses the rule that "`an author's address is the same as that of his supervisor`". The expression of `IC2` and `DR1` in the typed first-order assertion language of Telos is shown below:

$$\forall p/Confpaper,\ \forall x/Author,\ \forall t_1,t_2/TimeInterval\ [author(p,x,t_1,t_2) \Rightarrow \neg\exists t_3,\ t_4/TimeInterval\ ref_by(p,x,t_3,t_4)) \wedge during(t_3,t_1) \wedge during(t_4,t_2)]\ (at\ 1988..*)$$

$$\forall a/Author, \forall m/Manager, \forall s/String,\ \forall t_1,t_2,t_3,t_4/TimeInterval\ [supervisor(a,m,t_1,t_2) \wedge addr(m,s,t_3,t_4) \Rightarrow addr(a,s,t_1 * t_3, t_2 * t_4)\ (at\ 1988..*)$$

The expression `pred(x,y,`t_1`,`t_2`)` represents the fact that during interval t_2, it was believed by the knowledge base that `pred` held between `x` and `y` for time interval t_1. Literally then, `DR1` states that "*If manager m is the supervisor of author a during t_1 according to beliefs of t_2 and m had address s during t_3 according to beliefs of t_4, then a has address s too during the time interval defined by the intersection of t_1 and t_3 for the belief interval defined by the intersection of t_2 and t_4*". The expression `(at 1988..*)` defines the lifetime of the rule itself (from 1988 onward).

A knowledge base can be queried with respect to both history and belief time. For example, the query shown below retrieves all employees who had a salary increase of more than $5K since the beginning of 1988, according to the system's beliefs for the same time period.

$$ASK\ e/Employee:\ \exists t_1,t_2/TimeInterval\ (e[t_1].salary \leq e[t_2].salary - 5000) \wedge before(t_1,t_2)\ ON\ (1988\ ..\ *)\ AS\ OF\ (1988\ ..\ *)$$

3. SYSTEM ARCHITECTURE

A good architecture delivers good performance at a reasonable cost. The system architecture adopted is shown on figure 2. It consists of three different layers:

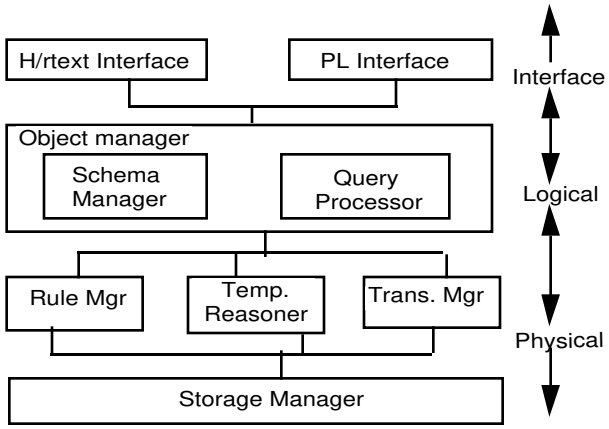

Figure 2: KBMS Architecture

• an interface layer, which offers different types of user interfaces including a hypertext interface, for users who want to browse through the knowledge base, and a programming language interface which supports knowledge base operations for accessing, updating the knowledge and a notion of transaction analogous to that found for databases.

• a logical layer which handles generic term descriptions (classes) as well as logical formulas serving as rules and constraints represented in the knowledge base, and supports syntactic and semantic query optimization

• a physical layer which manages the physical data structures on which classes and formulas are mapped and is responsible for disk I/O, caching etc.

The modules linking the logical and the physical layer manage rules (both their compilation and evaluation), support concurrency control (through the transaction manager) and perform temporal and possibly other types of specialized reasoning.

The proposed architecture is similar to that of the ORION object-oriented database system [Kim90]. The main difference is that the Telos KBMS supports inference mechanisms for deductive rules and integrity constraints as well as temporal knowledge, thereby increasing the complexity of the overall design.

4. STORAGE MANAGEMENT AND QUERY PROCESSING

This section presents a storage facility for Telos-like, large knowledge bases, assuming a storage architecture which includes secondary memory. As with database management systems, the primary factor leading to good performance for a KBMS is the minimization of disk I/O during knowledge base operations. The design of a storage model for a knowledge base is very much influenced by the features of the underlying knowledge representation language. For instance, generalization relationships broaden the extension of classes thereby complicating query processing. Likewise, query processing needs to deal with deductive rules and integrity constraints which define arbitrarily long logical connections between elements of the knowledge base.

Given a knowledge base, such as that of figure 1, the primary goal of storage design is to derive a logical and a physical storage organization which provide efficiency guarantees on the knowledge base operations. The system component which coordinates that is the *storage manager*. Storage design involves three steps. As an initial step, a graph is built reflecting the knowledge base structure and is annotated with quantitative information on the expected cardinalities of classes and attributes. This graph structure is called the *class schema* and constitutes the basis for logical and physical storage schema design. During the same step, multiple inheritance is resolved and generalization hierarchies are decomposed into trees. At the second step, the class schema is mapped onto a logical database schema (defined in terms of a set of relational schemata) through the *controlled decomposition algorithm*. At the third step, a physical storage organization is derived from the logical schema. In this paper we are mainly concerned about the second step.

The controlled decomposition algorithm is based on the controlled decomposition model (hereafter CDM), a flexible combination of the n-ary (direct) storage model (or NSM) and the decomposition storage model (or DSM) [Copeland85], [Valduriez86]. The NSM stores a complex object in one relation which includes one column for each complex object component or attribute. The DSM, on the other hand, defines one storage relation for each object component or attribute. The former model clearly requires less storage than any alternative models, but is not flexible in dealing with class schema updates nor does it adequately handle generalization hierarchies. Unfortunately, the DSM has its own drawbacks as it requires at least double the space of the NSM and fragments complex object descriptions into many relations, thereby slowing retrieval. The CDM offers a middle-of-the-road approach, enjoying many of the benefits of both models. In the following, we outline the controlled decomposition algorithm, which partitions a class schema into a logical schema.

1. All Telos metaclasses are stored in predefined relations which tend to be large and therefore are secondary memory resident. However, they are accessed in a very predictive ways and thus are optimizable;

2. Decompose the generalization hierarchy of simple classes into a number of generalization trees so that the top class of each generalization tree does not have multiple parents (i.e., we disconnect cases where a class belongs to more than one genaralization hierarchy and also resolve cases of multiple inheritance);

3. Each top class of a generalization tree becomes a relation nucleus; attach to each relation nucleus as fields the class attributes whose domains are built-in classes, for instance String, Integer, (in the spirit of the NSM).

4. Attributes, whose values are instances of other classes, are stored in the form of join relations (in the spirit of the DSM); each join relation represents links between two classes [Valduriez87].

5. Descending the generalization hierarchies, create relations for subclasses in order to store new attributes not inherited from their ancestors.

Tokens are stored in the relation corresponding to their most general class. Subclass instances are partly stored in the ancestor relation and partly locally. In some cases, the clustering policy is likely to place such complementary records physically close. The CDM algorithm also assigns token identifiers for all tokens of the same generalization tree reflecting their position within the tree. These identifiers are utilized by the clustering policy.

Figure 3 illustrates an application of the above method to portions of the knowledge base of figure 1. All internal identifiers for persons are prefixed with 'P' followed by a number which represents a particular class within the person hierarchy. The '*' on the first table of the figure

indicates time invariant values, while the time interval associated with each attribute value indicates the time when that value applies (i.e., its history time). Belief times are not shown on the figure. For the class `Employee`, the salary attribute is attached to its corresponding EMPLOYEE relation while the dept attribute is stored separately.

PERSON

P_ID		PNAME	
P21	1.6	Alex	*
P12	2..10	Elen	*
P03	4..8	Mary	*
P24	8..10	Mike	*
P15	6..9	Nick	*
P06	5..9	John	*
P27	6..10	Lora	*

EMPLOYEE (IsA PERSON)

E_ID		SALARY	
P21	1.6	45000	1..4
		55000	5..6
P12	2..10	50000	2..6
		60000	7..10
P24	8..10	60000	4..8
P15	6..9	60000	8..10
P27	6..10	60000	6..8
		65000	9..10

MANAGER (IsA EMPLOYEE)

M_ID		PROJECT	
P21	1..6	AI1	3..4
		AI2	5..6
P24	8..10	DB1	6..8
		DB2	9..10

DEPARTMENT

D_ID	DNAME

EMP_DEPT

E_ID	D_ID	T

Figure 3: Controlled decomposition example

Another important feature of the KBMS storage manager is the development of a *temporal join index* which is used to optimize sub-object accesses. This is a generalization of the join index proposed in [Valduriez87] which accommodates the temporal dependency on the connection of two tokens.

Consider two classes C_1 and C_2 and a time-dependent attribute *a* of C_1 which takes as values instances of C_2. A temporal join index relation (hereafter TJI) is defined as

$$TJI = \{ \langle i(r), i(s), t \rangle \mid r \text{ in } C1, s \text{ in } C2, s \text{ in } r.a \text{ [at } t] \}$$

where *i(c)* denotes the internal identifier of *c*. A TJI stores a time-dependent relationship for purposes of fast access to the instances of the two joined classes. The proposed TJI possesses a number of useful properties. Firstly, the index is bi-directional (see figure 4). Secondly, the TJI easily generalizes to access paths involving chains of several attributes (e.g., EMPLOYEE . DEPARTMENT . UNIVERSITY) forming a *temporal multi join index* which extends the path multi-index [Bertino89] with time.

The TJI has been implemented as a clustered, secondary storage resident index. As with the join index [Valduriez87], two copies of the TJI are stored (see figure 4): one clustered on source identifiers and one on destination ones, thereby making bi-directional access possible. Each copy of the TJI is implemented in terms of an R-tree-like data structure [Guttman84]. The reason for choosing a spatial data structure, such as R-trees, is that identifiers are not just points on a one dimensional value space, but line segments on a two-dimensional space where the time component specifies the lifetime of the relationship between the joined identifiers. There are two important parameters which influence the performance of the TJI: the degree of sharing among index paths [Bertino89] and the number of versions per attribute occurance [Topaloglou93]. More details and a thorough analysis of the performance of TJI over various data distributions is also presented in [Topaloglou93].

The query processing cycle of the proposed KBMS consists of two phases. The first phase is the *semantic query optimization* phase which involves generation of an equivalent query expression that can be processed more efficiently than the input query. This phase is discussed in detail in [Topaloglou92]. The second phase is the *physical query processing* phase where optimization is accomplished in terms of bulk operations called *access paths* which are supported directly by the underlying storage model. The selection of optimal access paths is an intriguing problem referred to as *access planning* [Selinger79]. Before planning a query execution scenario by choosing among alternative access plans, the access planner needs to estimate the execution cost of specific access operations, such as the cost of accessing an index, the cost of scanning a relation, etc [Mylopoulos93]. The cost model used for this purpose is a revised version of the cost model of [Kim89] for query processing in object-oriented databases. Selectivities of comparison operators and access costs have been reconsidered by taking into account parameters which depend on the distribution of data / knowledge over time. For instance, the selectivity of a selection operation such as `employee.salary=50000 ON 92..94` will be the selectivity of the operation as in [Selinger79] (i.e., *1/cardinality(s*alary*)*) multiplied by the *temporal selectivity factor* . For the KBMS, this has been defined as:

*= [(query specified time) / (max length of temporal domain's history] * (temporal distribution factor).*

Additional selectivity factors had to be introduced for the likelihood of temporal predicates or operators that appear in Telos queries, e.g., the `before(`t_1`,`t_2`)` predicate in the query shown earlier reduces the selectivity of the query by the following factor:

$$F_{before} = [(e(t1) - s(H)) * (e(H) - s(t2)) / (e(H) - s(H))^2] * d$$

where *s(t), e(t)* denote start and end of an interval, *H* the length of the encoded history and *d* is the factor which depends on the distribution of the temporal domain.

Finally, preliminary performance analysis results [Topaloglou93] indicate that the CDM has lower storage cost than the NSM but higher than those of the DSM.

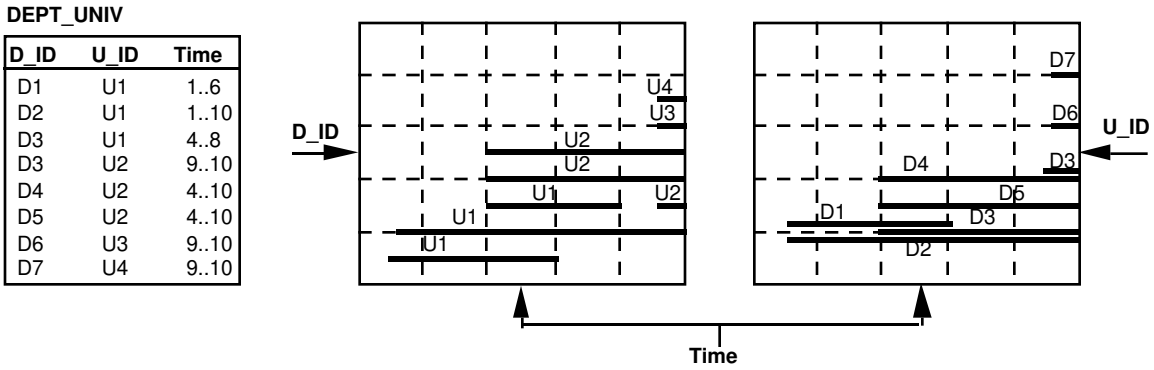

Figure 4: The temporal join index

Moreover, it performs as well as the DSM on attribute-value and schema updates and does better than the DSM on token updates. The CDM's performance is inferior to that of the NSM on token upadates, but does not have the severe drawback of the NSM with respect to schema updates. The query cost of the CDM with TJI for associative queries and attribute retrievals is, as expected, lower than that of the NSM. It should be noted that the TJI could be employed with other storage models, for example, the DSM. The integration of time in a single index dictates changes to the cost model and access-level optimization for structurally object-oriented temporal knowledge bases. Finally, our analysis has shown the correlations between the query processing cost and the temporal selectivity for various temporal distributions.

5. CONCURRENCY CONTROL

As knowledge bases become global resources, rather than individualized aids, it is reasonable to expect that they will be used simultaneously by several users. Simple-minded solutions, such as serving these users by maintaining multiple copies of the same knowledge base, or locking all users out while a single user is accessing and/or updating the knowledge base will soon become impractical. Instead, the Telos KBMS proposes to serve multiple users sharing a single knowledge base by interleaving the execution of operations against the knowledge base, thereby optimizing the deployment of computing resources, both CPU cycles and space. Concurrent processing of user operations (hereafter *transactions*) is a well-established database optimization technique which has been shown to improve system performance up to several orders of magnitude [Gray93], depending on the number of users and the nature of their transactions. Concurrent processing is accomplished through a *concurrency control policy* [Bernstein87,Papadimitriou86].

As the reader might expect, not all interleaved executions of a set of transactions leads to correct results. The classic example of such a situation is the concurrent retrieval by two transactions of an amount, say $50, from a single bank account holding $75. Assume that each transaction first reads the account balance and, if sufficient, deducts the amount withdrawn. If the transaction execution is interleaved so that both transactions first read the account balance ($75, hence sufficient to each transaction individually) then both transactions can proceed to withdraw $50 and deduct $50 from the account balance leading to an unintended result (at least from the bank's point of view).

To avoid such situations, Database research has studied the notion of correctness of interleaved execution through the concept of *serializability* [Bernstein87, Papadimitriou86, Gray93]. Serializability requires that the interleaved execution of transactions should be equivalent to some serial execution of the same collection of transactions (in terms of the final state of the knowledge base and the values returned by each transaction). Although several approaches have been proposed in the literature for achieving serializable concurrency control policies for databases, *locking-based* algorithms have been most successful for commercial systems. The best known locking algorithm, *two phase locking* (affectionately referred to as 2PL) [Eswaran76], works along the following lines. Associated with each data item is a distinct "lock". A transaction must acquire a lock on a data item before accessing it. While a transaction holds a lock on a data item no other transaction may access that item. A transaction cannot acquire any additional lock once it has released some lock (i.e., no locks can be acquired once the transaction has moved into lock-release phase, hence the name two phase locking).

For knowledge base operations such as recursive rule processing, transactions are likely to access large portions of a knowledge base. In such situations, 2PL dictates that each transaction hold locks until it finishes acquiring all the locks that it will ever need. Hence, such global operations will lock much of the knowledge base, thereby reducing the benefits of concurrency control. Therefore, to handle the concurrency control problem for knowledge bases one needs to look for policies that hold only a small number of locks at any time and are serializable.

Interestingly, knowledge bases generally possess a much richer structure than databases (defined in terms of generalization and aggregation hierarchies, deductive rules, temporal or spatial dimensions etc.). This structure can be potentially useful in allowing early release of locks and therefore more efficient concurrency control policies for knowledge bases. One policy that is based on this idea is the *DAG policy* [Silberschatz80,Yannakakis82] and is applicable to situations where knowledge base structure defines a directed acyclic graph (DAG). Under DAG policy, a transaction may begin execution by locking any item. Subsequently, it can lock an item if it has locked all the parents of that item in the past and is currently holding a lock on at least one of them. As long as these assumptions apply, lock releases are allowed throughout transaction execution. For instance, assuming that transactions need to traverse all nodes of a linearly ordered knowledge base, any one transaction need only lock two nodes at a time (the node currently accessed and its parent), whereas 2PL would require locking the whole knowledge base before there were any lock releases.

The DAG policy works (and always leads to serializable executions) because of the assumption that there are no cycles in the underlying structure and the structure does not undergo any change. Unfortunately, the structure of a knowledge base will contain cycles (for instance, the inference graph generated for a collection of recursive rules) and will undergo change (for example, when rules are added or deleted). Accordingly, we have enhanced the DAG policy to *Dynamic Directed Graph policy* (DDG) that works even in the presence of cycles and updates of the knowledge base.

For purposes of concurrency control, a knowledge base is viewed as a directed graph G(V,E) where V is a set of nodes v_i (e.g. Employee in figure 1), and E is a set of edges which are ordered pairs (v_i,v_j) of nodes (e.g. (Employee, Manager) in figure 1). This graph includes as a subgraph the class schema defined in section 4, but also represents structural information about tokens and cross-references among rules.

The DDG policy operates in terms of three types of rules. (i) pre-processing rules specify a simple construction that converts any arbitrary graph to a rooted, connected graph; (ii) locking rules specify how each transaction should acquire locks; (iii) maintenance rules} specify additional operations that must be executed by transactions to keep the knowledge base structure in the desired form. In the sequel, we only present the locking rules of the DDG policy. A detailed description of our model and the algorithm is available in [Chaudhri92].

For the purposes of this section we will assume that the pre-processing rules have been already applied to the knowledge base and the graph G is rooted and connected. In such a graph, a *dominator* of a set of nodes U is defined as a node d such that all paths from the root node to each node v e U pass through d. The root node dominates all nodes in the graph, including itself. Let D be a dominator of all the nodes in R(T), where R(T) is the set of nodes to be accessed by a transaction T. The *entry point* of a strongly connected component G_i of graph G is a node v of G_i, such that there is an edge (w,v) of G so that w is not in G_i.

Given these definitions, we can describe DDG's locking rules for a transaction T to be executed against a knowledge base with structure graph G:

Rule L1. The first node to be locked by T is D, a dominator of R(T) with respect to G.

Rule L2. Before T performs any operation (insertion, deletion or access), on a node v (respectively, an edge (u,v)), T has to lock v (respectively, both u and v).

Rule L3. A node v can be locked if and only if all its predecessors in the present state of G that do not lie on the same non-trivial strongly connected component as v, have been locked by the transaction in the past and the transaction is presently holding a lock on at least one of them. All the nodes on a strongly connected component are locked together in one step, provided all the entry points of that component have been locked. A node that is being inserted can be locked at any time.

Rule L4. Each node can be locked at most once.

It has been shown [Chaudhri92] that the DDG policy only generates schedules which are serializable. and deadlock-free. Moreover, the DDG policy has been proven to be well-structured in the sense that it will handle arbitrary transactions. This overcomes shortcomings of earlier policies that impose artificial restrictions on the set of transactions that can be processed [Yannakakis82]. However, the DDG policy, in general, does not permit concurrency within cycles (see rule L3 above). This suggests that if a knowledge base contains very large cycles which need to be locked in one step, concurrency will be severely hampered. Thus, a possible recommendation for the design of a knowledge base, from the viewpoint of concurrency control, would be to keep the number and size of cycles to a minimum.

We have studied the performance of the DDG policy on a knowledge base that is being developed for process control in a nuclear power plant. The graph corresponding to this knowledge base has cycles and undergoes change. This knowledge base receives two classes of transactions. The first class consists of short transactions, which look up or update an attribute value and occasionally change the semantic relationships in the knowledge base. The second class consists of long transactions which involve traversal of the knowledge base along some of its

semantic relationships. The proportion of the transactions in the first class is 73% with the remaining 27% in the second class.

The DDG policy has been implemented in the DeNet simulation environment [Livny86]. This implementation has been used to measure the relative running costs of 2PL and DDG policies , which in turn, have been used in our simulation to reflect the overhead of using the DDG policy. Moreover, we used the knowledge base and the transactions derived from the above mentioned knowledge base as an input to our model. Our results indicate that when the long transactions are read only, the DDG policy performs comparably to 2PL. When the long transactions also perform some updates, however, the DDG policy can improve considerably response times for short transactions. This superiority of performance is based on the fact that when there are only shared locks in a transaction, the DDG policy cannot allow any pre-release of locks, and therefore, performs comparably to 2PL. On the other hand, if the transactions are update intensive, the extra overhead is more than offset by the increased concurrency obtained due to pre-release of locks. [Chaudhri94] describes a comprehensive set of experiments, intended to evaluate the DDG policy.

6. RULE AND CONSTRAINT MANAGEMENT

Within the KBMS architecture, the *Rule Manager* is responsible for the evaluation of deductive rules and integrity constraints. This section describes this KBMS component, focusing on extensions to existng techniques for rule and constraint management and enforcement. In particular, we describe a novel approach to the efficient enforcement of declarative temporal integrity constraints [Plexousakis93b] based on a compile-time simplification method initially proposed in [Bry88] and later adapted to an object-oriented setting in [Jeusfeld90]. Our approach extends this work by offering special treatment for temporal knowledge also by generalizing the compilation phase.

In general, the problem of constraint enforcement in the presence of deductive rules, historical and belief time is decomposed in two phases. During the *compilation* phase, integrity constraints and relevant deductive rules are compiled into parameterized forms organized in terms of a dependence graph and simplified by the application of a sequence of truth-preserving transformations on temporal and non-temporal subformulae. The application of these simplification steps exploits the assumption that the knowledge base satisfies its constraints prior to any knowledge base operation [Nicolas82]. The second phase is purely an *evaluation* phase during which the constraints and rules relevant to an update - both from a logical and a temporal viewpoint - are selected and evaluated against the knowledge base.

The two-fold presence of time in Telos (history/belief time) allows for the expression of constraints that can't be handled by other formalisms that support a single dimension of time, e.g. [Bry88], [Hulsmann90]. For purposes of enforcement, three types of constraints are distinguished by the rule manager: (i) *state* constraints, specifying properties that must hold in any one domain state (i.e., history time), (ii) *dynamic* constraints, referring to two or more domain states, and (iii) *dynamic epistemic* constraints, referring to two or more belief states, in addition to possibly multiple domain states. These are expressed as range-restricted formulae of the assertion language.

The compilation process yields a *parameterized simplified form* for each literal in a constraint with the components of the literal as parameters. To ensure that the integrity of the knowledge base is preserved it suffices to evaluate this form when *affecting updates* occur: an insertion (deletion) to (from) the extension of a literal of a constraint *affects* the constraint if a literal unifiable with the update occurs negatively (positively) in the constraint and the literal's temporal components intersect with those of the constraint. In other words, unifiability of updates with literals in the constraint refers to their temporal components as well. For each literal *L*, a *concerned class* *C* is defined for the purpose of restricting the search space for constraints affected by updates on the literal. *C* is the most specialized class such that, insertion or deletion of an instance of *C* affects the truth of *L* and the respective time intervals of *L* and *C* are overlapping. The concerned class and the history and belief time intervals of the constraint also form part of each of its parameterized simplified forms. Finally, the simplified form is derived by applying the following steps:

1. The quantifiers binding variables occurring in the literal are dropped.

2. The temporal variables of the formula are constrained with respect to the temporal intervals of the constraint. The resulting during relationships between temporal variables of the formula and the known constant intervals of the constraint are conjoined with the constraint.

3. The atom into (from) whose extension a tuple is inserted (deleted) can be substituted by the constant True (False) and absorption rules of first-order predicate calculus are applied.

4. Temporal simplification is applied if possible without introducing incomplete knowledge. Expressions of the form $\texttt{during(t,i}_1\texttt{)} \wedge \texttt{r}_1\texttt{(t,i}_2\texttt{)} \wedge \texttt{r}_2\texttt{(i}_1\texttt{,i}_2\texttt{)}$ are replaced by equivalent forms $\texttt{r(t,i)}$, where $\texttt{i}$ is an interval depending on $\texttt{i}_1$ and $\texttt{i}_2$. A table lookup is merely required for any of the 169 possible cases of relationships $\texttt{r}_1\texttt{,r}_2$. Table 1 shows only a portion of the table of simplifications, with * denoting the interval

r1(t,i2) / r2(i1,i2)	before	during	overlap	meet	start	finish	equal
before	before i1	⊥	⊥	⊥	⊥	⊥	⊥
during	⊥	during i1	⊥	⊥	⊥	⊥	⊥
overlap	dur i1-i1*i2	dur i1*i2	ove i1-i1*i2	fin i1-i1*i2	sta i1-i1*i2	⊥	⊥
meet	during i1	⊥	⊥	⊥	⊥	⊥	⊥
start	⊥	during i1	⊥	⊥	⊥	⊥	⊥
finish	⊥	during i1	⊥	⊥	⊥	fin i1	⊥
equal	⊥	during i1	⊥	⊥	⊥	⊥	⊥

Table 1: Temporal Simplification

intersection operation. Relation r_2 is derived by merely examining the constant intervals i_1 and i_2 and can be determined in constant time.

As an example of the temporal simplification process, consider the conjunction of temporal relationships :

during(t,01/88..09/88) ∧ before(t,05/88..12/88)
∧ overlaps(01/88..09/88,05/88..12/88)

According to table 1, the above expression can be simplified - at compile time - into the simpler equivalent expression during(t,01/88..05/88). Returning to the example of section 2, we rewrite constraint IC2 in the following form that lends itself more readily to the application of the simplification steps.

$\forall$p/Confpaper, $\forall$x/Author, $\forall$r/Referee,
$\forall t_1,t_2,t_3,t_4$/TimeInterval
[author(p,x,t_1,t_2) ∧ ref_by(p,x,t_3,t_4)
∧ during(t_3,t_1) ∧ during(t_4,t_2)] ⇒
[r ≠ x (at t_1*t_3 - believed at t_2*t_4)]
(at 1988..*)

Then, assuming the update is an insertion author(P,X,T,T') and that the constraint was defined on 02/01/88, the compilation process yields the simplified form:

$\forall$r/Referee,$\forall t_3,t_4$/TimeInterval
[ref_by(p,x,t_3,t_4)∧during(t_3,T)∧during(t_4,T')]
⇒[r ≠ x (at t_3 - believed at t_4)]

The above simplification phase has been shown in [Plexousakis93b] to be sound and complete. The evaluation phase then has to replace the parameters with the actual values of the update and evaluate the considerably simpler form by means of a hybrid reasoner (e.g. [Miller90]. The same process applies to the bodies of deductive rules. The method also treats updates of deductive rules and integrity constraints. During compilation, the parameterized forms of rules and constraints are organized in a dependence graph that reflects their logical and temporal interdependence. The graph is modified incrementally and in a minimal manner when rules or constraints are added or deleted. Paths in the graph show how updates on literals in bodies of rules can affect the constraints and other deductive rules. Hence, *potential implicit updates* can be derived at compile-time. These have to be refined to the *actual implicit updates* at run-time. Implicit updates are treated as normal updates. For instance, rule DR2 expresses the property that ``*the department a university affiliate works in is derived by checking which department has the same address as he does''*. Hence, an update on the supervisor literal of DR1 causes the derivation of an addr literal (by DR1) and, in turn, that of a works_in literal (by DR2) which affects IC1.

An analysis of the performance of the simplification method has been undertaken in order to assess the gains from the compile time organization of rules and constraints in a dependence graph and the implicit update precomputation. The proposed algorithms for the computation and on-line maintenance of the dependence graph transitive closure were implemented and a number of experiments was performed on randomly generated graphs. The results indicate that compilation and precomputation of implicit updates yield an average gain of up to 95% for the run-time maintenance of rule and constraint dependencies. More details about the performance assessment of the proposed method can be found in [Plexousakis93b] and [Mylopoulos93].

7. CONCLUDING REMARKS

This paper presents an overview of a novel knowledge base management system which adopts database

techniques to support efficient knowledge base storage management, query processing, concurrency control, rule evaluation and constraint enforcement. The proposed implementation techniques have been evaluated under different assumptions of knowledge base size, structure and usage and have been shown to perform generally better than alternative techniques.

Clearly, the proposed design is neither complete, nor has it seen a prototype implementation that integrates all proposed implementation techniques. Nevertheless, we believe that the results obtained so far suggest that an efficient KBMS that advances the state-of-the-art in the management of large knowledge bases *is* feasible and can be obtained by extending database techniques.

ACKNOWLEDGEMENTS

This work has been supported by the Information Technology Research Centre, funded through Ontario's Centres of Excellence programme, and the Department of Computer Science of the University of Toronto. The authors wish to thank professors Vassos Hadzilacos (University of Toronto), Ken Sevcik (Universtiy of Toronto), Yannis Ioannidis (University of Wisconsin), Miron Livny (University of Wisconsin), Mike Carey (University of Wisconsin) and Yannis Vassiliou (Technical University of Athens) for valuable technical advice and encouragement.

REFERENCES

[**Allen83**] J. Allen, "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM*, 26(11):832--843, 1983.

[**Bry88**] F. Bry, H. Decker and R. Manthey, "A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases", Proceedings of *EDBT-88* , pages 488--505, 1988.

[**Bernstein87**] P. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems.*, Addison-Wesley, 1987.

[**Bertino89**] E. Bertino and W. Kim, "Indexing Techniques for Queries on Nested Objects", *IEEE Transactions on Knowledge and Data Engineering,* 1(2):196--214, 1989.

[**Chaudhri92**] V. Chaudhri, V. Hadzilacos and J. Mylopoulos, "*Concurrency Control for Knowledge Bases*", Proceedings *3rd International Conference on Principles of Knowledge Representation and Reasoning,* 1992.

[**Chaudhri94**] V. K. Chaudhri, "T*ransaction Synchronization in Knowledge Bases: Concepts, Realization and Quantitative Evaluation*, PhD Thesis, Universtiy of Toronto, Forthcoming.

[**Copeland85**] G.P. Copeland and S.N. Khoshafian, "A Decomposition Storage Model", Proceedings *ACM SIGMOD International Conference on Management of Data*, pages 268--279, 1985.

[**Eswaran76**] K.P. Eswaran, J. N. Gray, R. A. Lorie and I. L. Traiger, "The Notions of Consistency and Predicate Locks in Database Systems", *Communications of the ACM ,* 19(9):624--633, 1976.

[**Gray93**] J. N. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, San Mateo, 1993.

[**Guttman84**] A. Guttman, "R-Trees: A Dynamic Index Structure For Spatial Searching", Proceedings , *ACM SIGMOD International Conference on Management of Data*, pages 47--57, 1984.

[**Hulsmann90**] K. Hulsmann and G. Saake, "Representation of the Historical Information Necessary for Temporal Integrity Monitoring, Proceedings *EDBT-90*, pages 378--392, 1990.

[**Jeusfeld90**] M. Jeusfeld and E. Kruger, "Deductive Integrity Maintenance in an Object-Oriented Setting", Technical Report MIP-9013 , Universitat Passau , 1990

[**Kim89**]W. Kim, K.-C. Kim and A. Dale, "Indexing Techniques for Object-Oriented Databases", In *Object-Oriented Concepts, Databases and Applications* , ACM Press, 1989.

[**Kim90**] W. Kim, J.F. Garza, N. Ballou and D. Woelk, "Architecture of the ORION Next-Generation Database System", *IEEE Transactions on Knowledge and Data Engineering*, 2(1):109--124, March 1990.

[**Kuchenhoff91**] V. Kuchenhoff, "On the Efficient Computation of the Difference Between Consecutive Database States", Proceedings *Int. Conference on Deductive and Object-Oriented Databases* , pages 478--502, 1991.

[**Livny86**] M. Livny, "DeNeT User's Guide (Version 1.5)", Technical Report, University of Wisconsin, 1986.

[**Lockemann91**] P. Lockemann, H. Nagel and I. Walter, "Databases for Knowledge Bases: Empirical Study of a Knowledge Base Management System for a Semantic Network", *Data and Knowledge Engineering,* 7(2):115--154, 1991.

[**Manthey90**] R. Manthey. "Satisfiability of Integrity Constraints: Reflections on a Neglected Problem", Proceedings *2nd Int.Workshop on Foundations of Models*

and Languages for Data and Objects, 1990.

[**Mylopoulos90**] J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis, "Telos: Representing Knowledge about Information Systems", *ACM Transactions on Information Systems*, 8(4):325--362, 1990.

[**Mylopoulos92**] J. Mylopoulos, V. Chaudhri, D. Plexousakis and T. Topaloglou, "A Performance -Oriented Approach to Knowledge Base Management", Proceedings *1st International Conference on Information and Knowledge Management*, 1992.

[**Mylopoulos93**] J. Mylopoulos, V. Chaudhri, D. Plexousakis and T. Topaloglou, "The Design of a Prototype Knowledge Base Management System", forthcoming , 1993.

[**Miller90**] S. Miller and L. Schubert, "Time Revisited", *Computational Intelligence*, 6:108--118, 1990.

[**Neches91**] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator and W. Swartout, "Enabling Technology for Knowledge Sharing", *AI Magazine,* 12(3):36--56, 1991.

[**Nicolas82**] J.-M. Nicolas, "Logic for Improving Integrity Checking in Relational Databases, *Acta Informatica*, 18:227--253, 1982.

[**Papadimitriou86**] C. Papadimitriou, *The Theory of Database Concurrency Control.*, Computer Science Press, Rockville, MD, 1986.

[**Plexousakis93a**] D. Plexousakis, "Semantical and Ontological Considerations in Telos: a Language for Knowledge Representation ", *Computational Intelligence, 9 (1)*, 41-72, 1993.

[**Plexousakis93b**] D. Plexousakis, "Integrity Constraint and Rule Maintenance in Temporal Deductive Knowledge Bases", Proceedings *19th International Conference on Very Large Data Bases*, pp 146-155, 1993

[**Selinger79**] G. Selinger, M. Astrahan, D. Chamberlin, R. Lorie and T. Price, "Access Path Selection in a Relational Database Management System", Proceedings *ACM SIGMOD International Conference on Management of Data*, pages 23--34, 1979.

[**Silberschatz80**] A. Silberschatz and Z. M. Kedem, "Consistency in Hierarchical Database Systems", *Journal of the Association for Computing Machinery, 27(1):72-80*, 1980.

[**Topaloglou92**] T. Topaloglou, A. Illarramendi and L. Sbattella "Query Optimization for KBMSs: Temporal, Syntactic and Semantic Transformation", Proceedings *8th Int. Conference on Data Engineering*, pages 310--319, 1992.

[**Topaloglou93**]T. Topaloglou, "Storage Management for Knowledge Bases", Proceedings *2nd International Conference on Information and Knowledge Management,* 1993.

[**Valduriez86**]P. Valduriez, S. Khoshafian and G. Copeland, "Implementation Techniques of Complex Objects", Proceedings *12th International Conference on Very Large Data Bases,* pages 101--109, 1986.

[**Valduriez87**] P. Valduriez, "Join Indices", *ACM Transactions on Database Systems,* 12(2):218--246, June 1987.

[**Yannakakis82a**] M. Yannakakis, "A Theory of Safe Locking Policies in Database Systems", *Journal of the Association for Computing Machinery,* 29(3):718--740, July 1982.

[**Yannakakis82b**] M. Yannakakis, "Freedom from Deadlock of Safe Locking Policies", *SIAM Journal of Computing*, 11(2):391--408, May 1982.