Implementing Telos

Bryan M. Kramer, Vinay K. Chaudhri, Manolis Koubarakis,
Thodoros Topaloglou, Huaiqing Wang, John Mylopoulos
Department of Computer Science, University of Toronto,
Toronto, Ontario, Canada. M5S 1A4.
kramer@ai.toronto.edu

1.0 Introduction

This paper describes experiences in implementing Telos [MBJK90, TK89], a knowledge representation scheme developed at the University of Toronto. In a nutshell, Telos is a hybrid of an object-centered representation for representing entities, a logic-based language for representing deductive rules and integrity constraints, and an interval-based language for representing time. The language is unique for its tight integration of temporal knowledge and for its meta-level capabilities which include a class/metaclass hierarchy and a reflective inference engine.

The main motivation for implementing Telos is to discover whether the chosen combination of representations results in an effective and efficient knowledge representation system. Another important goal is to discover how large a knowledge base can be supported using "traditional" AI implementation methodologies such as the use of LISP and memory resident data-structures. A long term goal of this research is to develop knowledge base management systems capable of storing on the order of 10^6 facts. Other goals are to test the language specification, to discover how users react to the language, to discover what features are used and how are they used, and to provide a tool for research into using knowledge representations.

There have been several projects which have used various implementations of Telos. An early version of Telos was used as a world modelling tool to support semantic representation in a multilingual natural language query language for database application in the ESPRIT project LOKI [BDH88]. The features of the language which were proved important in this application were the ability to express structural descriptions of an application domain, the deductive capabilities and the ability to represent and reason with temporal information. A Prolog prototype implementation was developed and demonstrated in the LOKI project. Another dialect of Telos has been used for modeling design histories for information systems as part of the ESPRIT project DAIDA [JR88]. The extensibility of the language through the meta-attributes facilities was extensively used. In addition, the temporal and deductive aspects of the language were found very helpful for defining constraints over the software life cycle. Recently, Telos has been used to express knowledge about software in the logical design of a software information base (SIB) designed to support software reusability as part of the ESPRIT II project ITHACA [CJM89]. All of the above applications are based on Prolog implementations of substantial portions of the language. Of these implementations, Concept Base [JJR88] is perhaps the most complete. Finally, a more complete Lisp-based implementation of Telos has been used to develop an alarm filtering expert system for a nuclear power plant [MWKT90]. This last implementation, which is called *KNOWBEL*, forms the basis of the discussion in this paper.

The paper is organized as follows. First, there is a brief introduction to Telos. This is followed by a description of the latest Lisp-based implementation and examples of interactions with this implementation. Next, work in progress in the areas of concurrency control and query processing is presented. This work is intended to support development of an implementation of Telos capable of supporting very large knowledge bases and multiple users. Finally, the concluding section summarizes what was learned by implementing Telos.

2.0 Overview of Telos

Telos was originally designed as a language for requirements modelling. It combines a language for structuring objects, a first order logic language for expressing integrity constraints, a logic programming language for expressing deductions, and an interval-based representation of time for handling temporal knowledge.

The object representation language describes the world in terms of classes and instances of those classes. An object may have attribute values for attributes defined in the classes of which it is an instance. Classes participate in an IsA (specialization) hierarchy with strict inheritance and multiple parents. Classes, their instances, and attribute relationships are all *Telos propositions*. Propositions are treated uniformly, and every proposition must be an instance of some other proposition. This leads to some interesting properties. First, classes themselves must be instances of other classes (called *metaclasses*). Secondly, attribute propositions are instances of classes which are themselves attribute propositions. This provides the link between the attribute description in a class and the assertion in an instance. For example, the link from `John` to `25` might be an attribute proposition that is an instance of the proposition linking `Person` to `Number` with label `age`. This continues up into the metaclass level thereby enabling the definition of *attribute categories*. Thirdly, attribute propositions may themselves have attribute values.

The second ingredient of Telos is a subset of first order logic that is used for stating integrity constraints. Integrity constraints are logical expressions that must be true at all times. An example might be that a person's birth date must follow her parents' birth dates by at least 10 years.

Next there is a more restricted subset of logic for expressing deductions that can be made in the knowledge base. The deductive inference engine is *reflective* in the sense of [Smi82] and [Kra87]. Reflection provides the ability to control the inference process through the use of objects in the knowledge base. This will be discussed in more detail in the next session.

Finally, Telos has integrated support for reasoning about time. Every relationship has two associated times: the *belief time* and the *history time*. The belief time is the time interval in which the knowledge base took the fact to be true. In general, the belief time of a fact begins when the fact is asserted and ends at the earlier of the present time and the time at which the knowledge base was told that the fact was no longer true. The history time is the time in history when the fact is true. For example, one might state that John is an instance of `Student` before `Mary is an instance of Professor`.

Interaction with a Telos knowledge base is through the five commands **Ask, Retrieve**, `Tell`, `Untell`, and `Retell`. `Ask` is used to pose a query to the knowledge base while `Retrieve` answers a query without recourse to deductive rules. `Tell` is used to add facts to the knowledge base, `Untell` is used to tell the knowledge base that a fact is no longer believed, and `Retell` combines `Untell` and `Tell`.

A complete description of Telos and a formalization is presented in [KMSB89]. The formalization provides a translation of Telos knowledge bases to theories in a many sorted first order language. [Ple90] provides a possible-worlds semantics for the language.

3.0 Description of the Implementation

The implementation consists of three major components: a sorted clausal resolution theorem prover built specifically to support Telos, an object module, and a temporal reasoner. The integrity constraint checking, deduction, and reflective reasoning capabilities of Telos are implemented in the context of the theorem prover. This section describes how the three components are linked and then continues with a short description of integrity constraint checking and reflective reasoning.

The theorem prover is linked to the object module in two ways. First, all variables in logical clauses have a sort which is an object from the object module (e.g. [Fri89]). Secondly, atoms corresponding to Telos objects are resolved through calls to the object module. For example, atoms that query relationships such as InstanceOf that are stored in the object module, for example `(not (instance-of` **`flight1334`** `$x/class...))`, are handled by that module. It is important to note that the linkage between the two modules is designed so that all classes of which `flight1334` is an instance can be successively bound to the variable `$x` if necessary. This is a form of theory resolution [Sti85]. Finally, the theorem prover has indices that are Telos specific; in particular, some atoms are indexed on their first argument as well as the predicate.

The linkage between the theorem prover and the temporal reasoner is similar. First, times are represented in the theorem prover as sorts on variables. When two time variables are unified, the substitution maps these variables to a new time variable whose sort is the intersection of the intervals of the original variables. Of course, if the time reasoner can show that the two intervals do not intersect, the unification fails. Secondly, several predicates such as `before` involving time variables are handled via theory resolution using the temporal reasoner. The use of sorts to distinguish temporal variables has also been described in [BTK89].

The temporal reasoner makes use of a time point representation of intervals as suggested in [VKvB89]. At present, however, the module does not compute and store the consequences of each time relation. Instead, a simple search through the network is done at the time of a query. As pointed out in [GA89], the former makes updates relatively expensive and consumes a great deal of storage, while the latter makes queries relatively expensive. All of the approaches suggested in [VKvB89], [GA89], and [MS88] are complicated by the fact that in Telos the relationships between intervals have individual belief times. Thus heuristics for improving the performance of the time reasoner have yet to be implemented.

Integrity constraint checking is based on ideas proposed in [BDM88]. Since the knowledge base can be assumed to be consistent before an update, integrity constraint checking involves identifying those constraints that might be affected by the current update and testing simplified versions of the constraints constructed from the current update. In addition, any deductive rules that match the current update and that are possibly relevant to some integrity constraint must also be included in the set of clauses to be tested. In order to avoid testing all deductive rules, a dependency checking algorithm is run each time an integrity constraint or a deductive rule is added. A deductive rule is relevant to an integrity constraint if there is some sequence of inferences from the deductive rule that results in a resolvent one of whose atoms unifies with an atom in the integrity constraint.

Reflective reasoning is a form of meta-level reasoning in which a portion of the knowledge base consisting of *control knowledge* is used to influence the course of an inference. This is implemented through a *level shifting* inference engine which under certain circumstances shifts its attention from the inference at hand to the control knowledge in order to make decisions about the lower level inference. Since it is would be very costly to reason about every control decision, level shifting must be restricted to those occasions in which it may be of use. In Telos, a shift is made when the atom which is the theorem prover's focus of attention has a **`meta-rule`** property. When this is the case, a control query is formulated and evaluated in the context of the meta-rule. If the result of this query is true, the bindings of variables in the query are used to change the state of the original inference. Example control actions include setting the priority of a resolvent, declaring a clause unresolvable, and forming a new resolvant. For example, one might have a meta-rule that returns a default value for an attribute if a depth bound is exceeded.

4.0 Interaction With Telos

End user interaction with a system built on a Telos knowledge-base will typically be through a graphical user interface. An example of such an interface is the interface to the alarm filtering system (see figure 1). For interactive development of a knowledge base, on the other hand, a knowledge engineer will want to interact through the Lisp-based Ask - Tell interface. A sample interaction with Telos in this mode is presented below.

For the example assume that a knowledge base containing information about airline flights has been loaded. Some attributes of a flight are its origin, destination, and cost. In the first example the user asks Telos for the destination of **flight3402:**

```
<cl> (ask '(exists ($x/city $t@-^+) (attr flight3402 * "destination"
$x * $t)) '(now +))
True
        $x   <new-york>
```

```
      $t    $0@-,1986/1/1-0:0:0
<state>
```

The query language is a version of first order logic that is readable by Lisp. Variables are symbols beginning with a $.A variable may be given a sort at its first occurrence in the sentence by using the notation **$var/sort** where sort is a class name. Temporal variables are identified by the syntax **$var@time-expr**, again at the first occurrence. Note that the value bound to **$t** in the result above is itself a temporal variable. In the example, this is bound to the interval from minus infinity (represented by -) to the beginning of 1986, this being the time associated with the attribute value. The second argument to is an optional belief time against which the query is to be evaluated. Finally, **ask** returns a state which can be used in subsequent calls to ask to generate the next answer. For our example one could continue with:

```
<cl> (ask *)
True
      $x    <tokyo>
      $t    $0@1986/12/31-11:59:59,+
<state>
```

Here, the destination of flight3402 is Tokyo from the end of 1986 until plus infinity.

Now suppose that the user's goal is to add some classes that can be used to find a minimum cost path between two cities. The first class, **proto-Flight-path**, is created as below:

```
<cl> (tell '(proto-Flight-path instance ((M1-Class)) with
      ((attribute
            (from city)
            (flight flight)
            (to city)
            (cost number)
            (subpath proto-flight-path)
            (flight-gen flight
                  :with (meta-rule(flight-path-mr
                        (=> (and (varp $v)(time-of $h $ih)
                              (instance-of $obj/flight flight $ih)
                              (attr $proto * "from" $origin * $ih)
                              (attr $obj * "origin" $origin * $ih)
                              (attr $obj * "price" $price * $ih)
                              (current-clause-value $state $current)
                              (+ $current $price $new-val)
                        (goal (attr $proto/proto-flight-path *
"flight-gen" $v * $h) $state
                              (sigma (val $v $obj)) (:value $new-
val))))))))))
<proto-Flight-path>
```

In addition to the obvious attributes describing a path, the class as an attribute, **flight-gen**, that has an associated meta-rule. The purpose of this meta-rule is to find an instance of **flight** and to assign an evaluation to the resulting resolvent. The predicate **(goal <goal> <state> <substitution> <properties>)** is true when **<goal>** can resolve to **nil** in the theorem prover state **<state>** with the given substitution, and resulting resolvent will have the properties **<properties>**. In the example, the **:value** property is assigned the cost of the partial path ending with the generated flight. When the theorem prover is in **best-first** mode, it chooses the resolvent with the lowest value as the next candidate for resolution.

Next, the subclass **Flight-path** of **proto-Flight-path** is defined. Note that in the assertion language variables are universally quantified if there are no quantifiers. Note also that the construct **#@((proto-flight-path)...)** is a *description* of an object and can appear wherever an object may appear. These structures are important because they may contain variables thereby allowing the inference engine to construct descriptions of objects that are not already in the knowledge base. Note that in the examples of descriptions, the history time value for attributes is made explicit. In contrast, the attributes of proto-Flight-path and Flight-path are taking on default values.

```
<cl> (tell '(Flight-path instance ((M1-Class)) isa ((proto-flight-
path)) with
      ((deductive-rule
            (path-recursive (=>
             (and (instance-of $p/proto-flight-path
                        proto-flight-path $t@-^+)
                  (attr $p * "from" $origin * $t)
                  (attr $p * "to" $to * $t)
                  (attr $p * "flight-gen" $f/flight * $t)
                  (attr $f * "origin" $origin * $t)
                  (attr $f * "destination" $dest * $t)
                  (= $subpath/proto-flight-path
                        #@((proto-flight-path)(("from" (_ $dest $t))
                                    ("to" (_ $to $t))
                                    ("flight" (_ $subflight $t))
                                    ("cost" (_ $subcost $t))
                                    ("subpath" (_ $sp $t)))))
                  (= $path-class flight-path)
                  (instance-of $subpath $path-class $t)
                  (attr $p * "subpath" $subpath * $t)
                  (attr $p * "flight" $f * $t)
                  (attr $f * "price" $price * $t)
                  (+ $price $subcost $cost)
                  (attr $p * "cost" $cost * $t))
            (instance-of $p flight-path $t))))

      (path-base (=> (and (attr $p * "flight-gen" $f/flight * $t@-^+)
                  (attr $f * "origin" $origin * $t)
                  (attr $f * "destination" $dest * $t)
                  (instance-of $p/proto-flight-path
                        proto-flight-path $t)
                  (attr $p * "from" $origin * $t)
                  (attr $p * "to" $dest * $t)
                  (attr $p * "subpath" null-flight-path * $t)
                  (attr $p * "flight" $f * $t)
                  (attr $f * "price" $price * $t)
                  (attr $p * "cost" $price * $t))
            (instance-of $p flight-path $t)))))))
<flight-path>
```

This class contains two deductive rules for determining whether a particular **proto-flight-path, $p,** is in fact a valid flight path. The first rule picks a flight that originates at the from attribute value **$p**, then recursively attempts to find a path from the destination of that flight to the **to** attribute value of **$p**. The second rule succeeds if there is a direct flight and terminates the recursion. What is important here is that both rules generate flights through the **flight-gen** attribute which invokes the meta-rule discussed above. One would ask for a path from Toronto to Chengdu in China using **ask** as follows:

```
<cl> (setq state
      (ask '(and (= $x #@((proto-flight-path)
                        (("from" toronto |$t(-,+)|))
                        ("to" (_ chengdu $t))
                        ("flight" (_ $flight $t))
```

```
                    ("cost" (_ $cost $t))
                    ("subpath" (_ $p $t)))))
              (instance-of $x flight-path $t)) :vars '($x))

True
     $x   @/desc12/(((<proto-flight-path> $0-,+))
               ((from (_ <toronto> $41988/12/31-11:59:59,+))
                (to(_ <chengdu> $41988/12/31-11:59:59,+))
                (flight (_ <flight3502> $41988/12/31-11:59:59,+))
                (cost (_ 1600 $41988/12/31-11:59:59,+))
                (subpath (_ @/desc13/(((<proto-flight-path>
                          $1-,+))
                          ((# #) (# #) (# #) (# #) (# #))))
<state>
```

Please be aware that the printed representation of a Telos structure differs slightly from the input representation. For example, the value of **$x** here is a description and temporal variables are missing the @ symbol. A subsequent call results in:

```
<cl> (Ask state)
True
     $x   @/desc21/(((<proto-flight-path> $0-,+))
               ((from (_ <toronto> $31988/12/31-11:59:59,+))
                (to (_ <chengdu> $31988/12/31-11:59:59,+))
                (flight (_ <flight3502> $31988/12/31-11:59:59,+))
                (cost (_ 1620 $31988/12/31-11:59:59,+))
                (subpath (_ @/desc22/(((<proto-flight-path>
                          $1-,+))
                          ((# #) (# #) (# #) (# #) (# #)))
                          $31988/12/31-11:59:59,+))))
```

<state>

Note the slightly higher value for the cost attribute of the path.

In addition to the interface at the Lisp level, there is also a graphical user interface (see figure 1) having buttons for loading and saving knowledge bases, for displaying the IsA and InstanceOf hierarchies graphically, editing objects, and making queries. Figure 1 shows both a graph and an editor window open on an object. Clicking the query button simply opens a window into which the lisp form of the query can be entered. The result of the query will be displayed in the output window.

5.0 Work In Progress

A major goal of research into implementing Telos is to develop a system that can contain a large amount of knowledge, perhaps on the order of a million objects. Such a system, called a *KBMS*, must maintain its knowledge on secondary storage. In addition, such a knowledge base would be shared by many applications. This section introduces work in progress in three areas that arise in the context of KBMSs: concurrency control, secondary storage management, and efficient access to the knowledge.

5.1 Concurrency Control

When multiple users or applications need access to a common set of data and knowledge such as a knowledge base containing information about a large organization, some sort of simultaneous access is necessary. Providing this access while maintaining the consistency of the knowledge base and providing correct answers to each application is called the *concurrency control problem*. This problem has been extensively studied in the context of databases [BHG87]. However, this new breed of KBMSs poses some additional problems in concurrency control that have not been researched so far. These problems require a re-evaluation of the assumptions on which DB research has been based in order to come up with a design for concurrency control best suited for KBMSs.

One of the these problems is the *object boundary problem*. Conventional database objects such as tuples in a relational databases have obvious boundaries. Interference between updates can be simply avoided by locking either tuples or relations. In contrast, Telos objects are complex and highly interrelated, and have no obvious boundaries. In fact, large portions of the knowledge base might be logically related and need to be locked together. In this case, relational database approaches such as two phase locking are not efficient. Currently, the possibility that the structuring mechanisms in Telos might be exploited to identify smaller boundaries is being explored.

A second important problem in checking for interference between two updates or an update and a query is providing a decision procedure can check whether the operations will access same data items. For relational databases, it has been proven in [Elk90] that this problem is NP-complete for first order logic expressions. This suggests that some sort of heuristic algorithm must be developed.

The concurrency control algorithm proposed for the Telos KBMS project avoids the computational pitfall of checking the equivalence of two logical expressions using an *aggressive* approach. Any **ASK** transaction for which conflict detection is not economical will be scheduled without checking for conflict. However, should any object accessed by the transaction be updated before the transaction is completed, the transaction will be restarted. For **TELL, RETELL,** and **UNTELL** locks must be acquired. Conflicts between updates are detected using an update compatibility matrix derived from the structuring mechanisms of Telos.

Current work is focusing on a performance analysis of the concurrency control algorithms. Performance analysis is important because an algorithm that may be computationally efficient may not perform well in practice (e.g., there is a constant time algorithm, "Lock everything", that is useless for concurrency control).

5.2 Storage Management

Storage management is concerned with how and where Telos assertions are stored. This includes the selection of physical storage structures and the selection of access paths. A storage model for Telos must include support for IsA hierarchies, set-based query operations, associative search, object identity, and direct access. The translation of a set of **TELL** statements into storage structures is called *compilation*.

The storage system for Telos knowledge bases must also be designed in such a way that history time and belief time are handled efficiently. Since these two types of time serve different purposes, the access patterns for the two kinds of temporal knowledge are likely to be quite different. Queries on historical knowledge are expected to be rather uniformly distributed throughout the historical period modeled. On the other hand, queries will probably concentrate on the current snapshot of the knowledge base and access old states of the knowledge base less frequently. Data structures and access methods for these kinds of temporal knowledge have received attention in the database literature, for example [KS89].

Lastly, the intended storage system must be able to evolve as the knowledge base evolves. In a knowledge base environment, changes to generic objects (classes) occur relatively frequently when compared to the case of database systems in which it is likely that the schema remains stable for long time. In addition to augmentation of the knowledge base, recompilation of parts of the knowledge base will be necessary to allow for global reorganization of storage structures as old structures become inefficient.

The proposed compile operation has the following steps. Given a description of a Telos knowledge base as a set of `TELL` statements, first construct a graph that corresponds to the semantic network structure of the knowledge base. Next, supplement the graph with dependencies between nodes and edges which come from an analysis of deductive rules and integrity constraints. Then add the graph control information which is derived from the input statements (e.g. attribute `studentNumber` is single valued and takes values from the primitive domain `Integer`) and from statistics or estimates about the population of classes or values for an attribute. Finally, traverse the graph and decide the storage scheme after evaluating a cost function or perform ad-hoc selections where not enough control information is available.

5.3 Query Optimization

In addition to the development of appropriate storage structures, efficient access to knowledge involves choosing the best way to evaluate a query. This is called *query optimization.* Query optimization requires that a KBMS plan how to evaluate a given query with respect to a given knowledge base. This planning must take into account the nature of the knowledge base (for example, how many students there are or the structure of the generalization hierarchy) as well as its physical implementation. While query optimization problems have been extensively studied in the context of relational databases (e.g., [Fre89]), there is virtually no corresponding work for KBMSs.

In general, query processing can be divided into two phases. First, a series of syntactic and semantic transformations are used to find an equivalent query which minimizes cost according to some cost function. This is called the *query modification phase.* Subsequently, the query is transformed into a procedure, taking into account physical data structures, quantitative information and the like, which is then executed. This is the *access planning phase.*

For Telos, query modification involves temporal simplifications, syntactic simplifications, and semantic transformation. Algorithms and a detailed discussion appear in[IST90]. Temporal simplification includes detection of contradictions or implications on temporal constraints appearing in the query and selection of the set of deductive rules and integrity constraints which are most effective for the query. The syntactic simplifier transforms the query using the structural constraints of the Telos knowledge model. Finally, the semantic transformation applies the set of rules selected during temporal simplification to the result of the syntactic transformation.

The access planning phase first determines the order in which classes appearing in the query should be traversed. Then, selection and join criteria are ordered. These steps make use of both information about the access paths for the storage structures and quantitative knowledge about the size of these data structures. Also, if a similar query has been processed in the past, then its plan may be used as a basis for the current plan. Finally, some sort of exhaustive or heuristic search is necessary to discover the plan with the minimum cost. This is one of the crucial and most costly points of the algorithm. Related work in the case of database management systems can be found in [SSD88].

6.0 Conclusions

This paper provides a brief overview of KNOWBEL, an implementation of the Telos knowledge representation scheme. Important features of Telos are 1) the integration of time into the object centered framework, and 2) the inclusions of various meta-level constructs including attribute categories, the InstanceOf hierarchy, and reflective reasoning. In the implementation of Telos an attempt has been made to integrate a number of state-of-the-art special-purpose reasoning modules in order to produce an effective knowledge representation system. This includes a sorted theorem prover, an object representation module, a temporal reasoner, and a graphical interface.

Unfortunately there are as yet few experimental results arising from the implementation of Telos. As mentioned in the introduction, implementations of Telos have been found useful for building a few knowledge bases for requirements modelling. No systematic study of what parts of Telos were used has been reported. However, it is known that the temporal reasoner has usually been disabled because of its negative effect on the performance of the system. For the implementation of Telos described here, however, no example of a knowledge base making much use of symbolic time intervals has yet been tested.

In the case of the alarm filtering application, it was found that moving from the original MRS-based version of KNOWBEL to the current version resulted in an order of magnitude improvement in performance. The improvement can be attributed mainly to the use of the object-module to store InstanceOf, attribute, and IsA relations. This is strong evidence in favor of the use of specialized reasoning modules in a knowledge representation systems.

The second major goal for the implementation was to determine limits to the sizes of knowledge bases in memory resident Lisp-based systems. The alarm filtering example again provides some insight. While the MRS-based version seemed to be reaching the limits of acceptable performance on a knowledge base of about 500 objects, the new version performs quite well. In fact, it appears that swapping due to a large image size will become a problem before the number of objects affects the performance of the inference engine.

The implementation effort revealed some aspects of the language specification that were difficult to implement. In particular, the ubiquitous presence of belief times and history times complicates the caching of inherited values because the number of possible combinations grows rapidly. In the temporal reasoning component, belief times make the straight forward application of various heuristics impossible. Such issues of implementation should be taken into account as the language evolves.

Finally, the long term goal of the research is to develop a KBMS, a system capable of storing a large amount of knowledge that is available to many users. In this paper, some of the issues currently being researched have been presented. It is hoped that some of these ideas

will be implemented using the existing Telos implementation as a basis.

References

1. [BDH88] Jean-Louis Binot, Bart Demoen, Karl-Heinz Hanne, Levy Solomon, Yannis Vassiliou, Walther von Haan, and Tom Wachtel. "LOKI: A Logic Oriented Approach to Data and Knowledge Bases Supporting Natural Language Interaction." In *Proceedings of the ESPRIT Technical Week Conference*, 1988.

2. [BDM88] F.Bry, H.Decker, and R.Manthey. "A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability."·In *Proceedings of the International Conference on Extending Database Technology*, 488-505, 1988.

3. [BHG87] Phillip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Publishing Company, 1987.

4. [BTK89] F. Bacchus, J. Tenenberg, J. Koomen, "A Non-Reified Temporal Logic." In R. Brachman, H. Levesque, R. Reiter, editors, *Proceedings of 1st International Conference on Principles of Knowledge Representation and Reasoning*, 2-10, Toronto, Ontario, 1989.

5. [CJM89] A.Constantopoulos, M.Jarke, J.Mylopoulos, B.Pernici, E.Petra, M.Theodoridou, and Y.Vassiliou. "Scripting the ITHACA Software Information Base: Requirements Functions, Structuring Concepts," Technical Report Ithaca FORTH 89.E2.1, 1989.

6. [Elk90] Charles Elkan. "Independence of Logic Database Queries and Updates." In *Proceedings of the 1990 Principles of Database Systems Conference*, 154-160, April 1990.

7. [Fre89] J.C. Freytag. "The Basic Principles of Query Optimization in Relational Database Management Systems." In *Proceedings of IFIP '89*, 801-807, 1989.

8. [Fri89] A.Frisch. "A General Framework for Sorted Deduction: Fundamental Results on Hybrid Reasoning." In R.Brachman, H.Levesque, and R.Reiter, editors, *Proceedings of 1st International Conference on Principles of Knowledge Representation and Reasoning*, 126-136, Toronto, Ontario, 1989.

9. [GA89] M.Ghallab and M.A. Alaoui. "Managing Efficiently Temporal Relations through Indexed Spanning Trees." In *Proceedings of IJCAI-89*, 1297-1303, 1989.

10. [IST90] A.Illarramendi, L.Sbattella, and T.Topaloglou. "Query Optimization for KBMSs: Temporal, Syntactic and Semantic Transformation." Technical report, Department of Computer Science, University of Toronto, 1990.

11. [JJR88] M. Jarke, M. Jeusfeld, and T. Rose. "A Global KBMS for Database Software Evolution: Documentation of the First Concept Base Prototype."Technical report MIP-8819, University of Passau, 1988.

12. [JR88] Matthias Jarke and Thomas Rose. "Managing Knowledge about Information System Evolution." In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 303-311, 1988.

13. [KMSB89] Manolis Koubarakis, John Mylopoulos, Martin Stanley, and Alex Borgida. "Telos: Features and Formalization." Technical Report KRR-TR-89-4 on Knowledge Representation and Reasoning, Department of Computer Science, University of Toronto, 1989.

14. [Kra87] B. M. Kramer, "Control of Reasoning in Knowledge-Based Systems," Technical Report CSRI-199, Department of Computer Science, University of Toronto, 1987.

15. [KS89] Curtis Kolovson and Michael Stonebraker. "Indexing Techniques for Historical Databases." In *Proceedings of the International Conference on Data Engineering*, 127-137, 1989.

16. [MBJK90] J.Mylopoulos, A.Borgida, M.Jarke, and M.Koubarakis. "Telos: A Language for Representing Knowledge About Information Systems." *ACM Transactions on Office Information Systems*, to appear 1990.

17. [MWKT90] J. Mylopoulos, H. Wang, A. Kushniruk, Y. Tian, "Using an Expert System Tool for Process Control Applications," in the Proceedings of the Canadian Conference on Electrical and Computer Engineering, Ottawa, ON. 1990, pp 42.4.1-42.4.6.

18. [MS88] S.Miller and L.Schubert. "Using Specialists to Accelerate General Reasoning." In *Proceedings of AAAI-88*, 161-165, 1988.

19. [Plexousakis90] D. Plexousakis. "An Ontology and a Possible-Worlds Semantics for Telos." Technical Report KRR-TR-90-7, Department of Computer Science, University of Toronto, 1990.

20. [Sti85] M. Stickel, "Automated Deduction by Theory Resolution," in *Proceedings of the Ninth IJCAI*, Los Angeles, CA, 1990, pp 1181-1186.

21. [Smi82] "Reflection and Semantics in a Procedural Language." MIT/LCS/TR-272, Massachusetts Institute of Technology, Cambridge, MA, 1982.

22. [SSD88] S.Shekhar, J.Srivastava, and S.Dutta. "A formal model of trade off between optimization and execution costs in semantic query optimization." In *Proceedings 14th VLDB Conf.*, Los Angeles, USA, 1988.

23. [TK89] Thodoros Topaloglou and Manolis Koubarakis. "Implementation of Telos: Problems and Solutions." Technical Report KRR-TR-89-8, Dept. of Computer Science, University of Toronto, 1989.

24. [VKvB89] Marc Vilain, Henry Kautz, and Peter van Beek. "Constraint Propagation Algorithms for Temporal Reasoning: a Revised Report." In D.S. Weld and J.deKleer,

editors, *Readings in Qualitative Reasoning about Physical Systems*, 373-381. Morgan Kaufmann, 1989

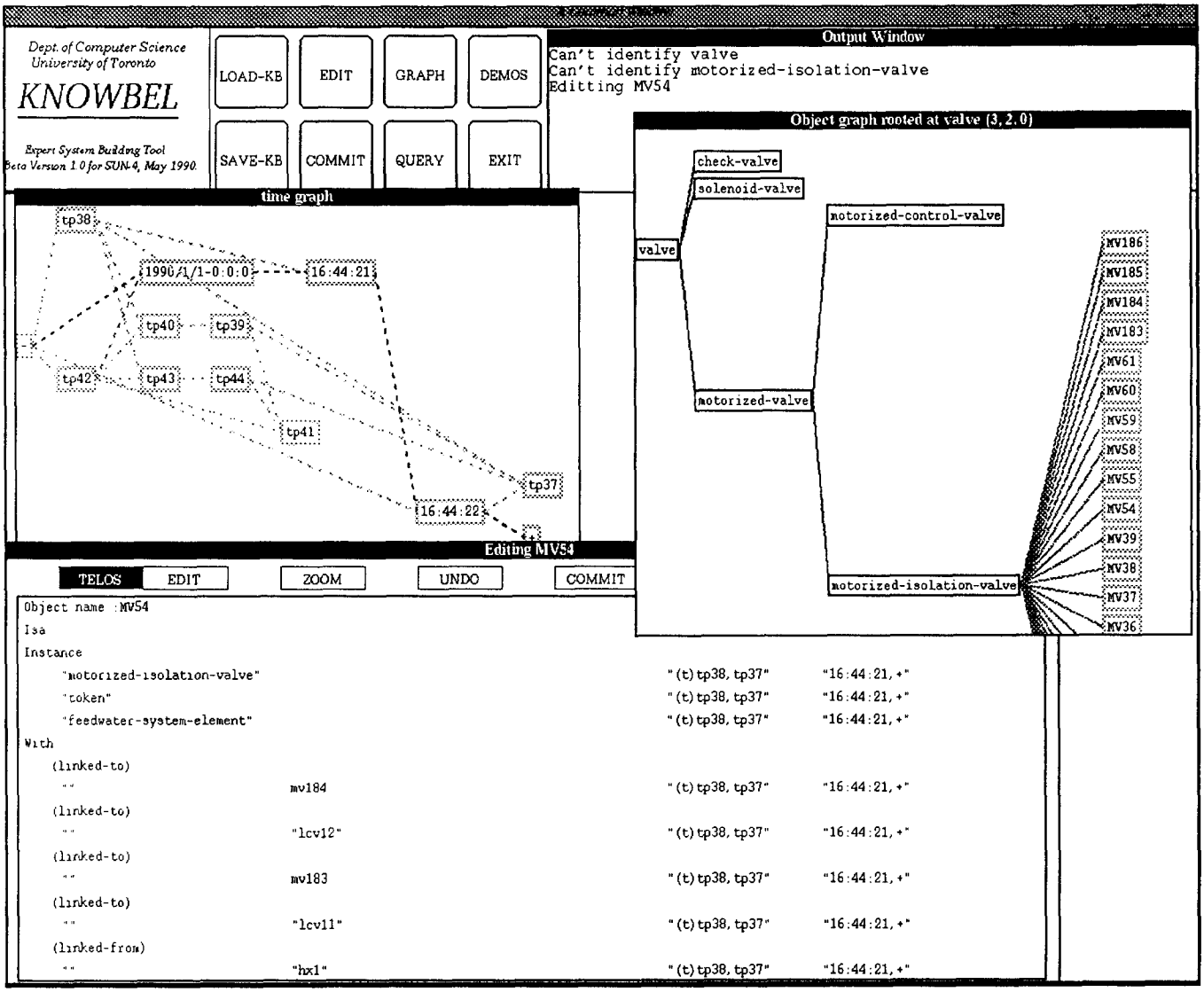

Figure 1: A Graphical Developer's Interface For Telos