Telos: A Knowledge Representation Language for Requirements Modelling

Manolis Koubarakis, John Mylopoulos
Martin Stanley, and Matthias Jarke

December 1988

FORTH/CSI/TR/1988/016
Technical Report Series
Computer Science Institute, FORTH†
P.O.Box 1385, Heraklio, Crete, 711-10 Greece
Tel.: +30.81.229302, Fax: +30.81.229342, Telex: 262389 CCI GR
E-mail – UUCP: mcvax@ariadne/csi

Submitted for publication
† “Foundation of Research and Technology – Hellas,” formerly “Research Center of Crete” (RCC)
1 Introduction

It is generally accepted that before design and implementation of large information systems, such as ones for airline reservations and inventory control, it is useful to describe the environment within which the system will eventually function and to prescribe the interactions between system and environment as well as the correspondences that must exist between the contents of the database managed by the system and the environment. This requirements modelling and analysis phase of software development, intended to describe the purpose of a system to be built, has been traditionally tackled with ad hoc notations and methodologies in the literature (e.g., [Cor85]). We have argued elsewhere [BGM85] that requirements modelling should be treated as a problem of knowledge base construction and analysis. Telos is intended to be a vehicle for testing this thesis.

Requirements models, formal or informal, are generally very large. In many cases, they involve descriptions of entities and activities that play a role in the environment or are part of the system under development. These descriptions are often given in terms of the history of the entity or activity being described. In addition, the language to be used needs to be extensible in a strong sense, to accommodate differences in the nature of entities and activities that play a role in a particular organizational setting. These three observations have shaped Telos as a unique knowledge representation language which focuses on (a) structuring mechanisms for knowledge, (b) integrated facilities for representing and reasoning about temporal knowledge, (c) a powerful mechanism for extending the language through user-defined meta-attributes and (d) embedded inference facilities for query evaluation and integrity checking in the spirit of deductive databases [GMS84]. The design of the language was also shaped by the desire to offer its users a uniform representational framework, and to adopt a functional view of knowledge bases [Lev84].

The language has been under development for more than seven years. An early version, called RML, is described in [Gre84], while a more recent version, called CML, is described in [Sta86] and [Kou88]. Much of the evolution of the language centered around its distinguishing features, namely representation of time and extensibility. The graceful integration of an object-oriented framework, used as a basis for structuring, and a logic-based one, used for deductive query evaluation and integrity checking, was another source of language revision. Finally, experience gained from prototype implementations and early users also led to improvements reflected in Telos.

Section 2 of the paper presents an impressionistic overview of the language, focusing on its main features, while section 3 sketches our approach to its formalization. The next two sections of the paper discuss its implementation and applications. Finally, section 6 discusses its relation to other work and outlines some directions for further research.

2 An Amateur’s Guide to Telos

This section illustrates the main features of Telos by means of examples. It also discusses some aspects of the knowledge representation system for which Telos serves as knowledge definition and knowledge manipulation language. Following a functional approach to knowledge representation [Lev84], Telos provides operations RETRIEVE and ASK for querying the knowledge base and operations TELL, UNTELL, and RETELL to inform the knowledge base. In this section, we firstly explore what a Telos KB can be told about, and then briefly sketch the querying facilities; more details can be found in [KMSB].

1Usually a part of an organization.
2Hence the name Telos from the Greek word telos which means end; the object aimed at in an effort; purpose. The word teleology has telos as one of its roots.
2.1 Propositions as a Structural Knowledge Representation

The representation framework of Telos generalizes graph-theoretic data structures used in semantic networks, semantic data models such as the entity-relationship model, and structurally object-oriented representations, by providing a single modeling unit, named proposition. Formally, propositions are quadruples with components from, label, to, and when denoting the source, label, destination, and duration of the proposition; these components are themselves propositions.

Propositions can be divided into two disjoint kinds: individuals (entities, nodes, objects, etc. in other formalisms) and attributes (relationships, links, etc.). As an example, the elementary statement:

Fikes is the first author of the paper about KEE

can be represented by the attribute proposition

< Fikes, first_author, KEE, t>

where Fikes and KEE are simple individuals, first_author is a label, and t is a time interval during which the above sentence is true.

The following TELL commands illustrate the structuring mechanisms of Telos:

```plaintext
TELL CLASS paper_class
    IN M1_Class
    WITH
        attribute
            paper_dscr: Class
    END

TELL CLASS conf_paper
    IN S_Class, paper_class
    ISA paper
    WITH
        paper_dscr
        referee: Referee;
        reply_address: Address
    END

TELL CLASS paper
    IN S_Class, paper_class
    WITH
        paper_dscr, unique
        author: Author;
        paper_dscr
        title: String
    END

TELL TOKEN KEE
    IN conf_paper
    WITH
        author
        first_author: Fikes
        title
        "The KEE system"n
    END
```

The IN clause gives the list of classes of which a defined object will become an instance. The ISA clause defines inheritance while WITH introduces attributes. We discuss each of these separately.

The instantiation mechanism of Telos defines an infinite dimension along which propositions can be classified into tokens (i.e., propositions having no instances; e.g., KEE), simple classes (i.e., propositions having only tokens as instances; e.g., paper and conf_paper), metaclasses (i.e., propositions having only simple classes as instances; e.g., paper_class), and so on. S_Class and M1_Class above are classes with extensions the set of all simple classes and the set of all metaclasses, respectively. In addition, the w-classes Proposition and Class have been defined which contain all the propositions and all the classes respectively in the knowledge base.

Instantiation relationships are represented by propositions which are themselves instances of the class InstanceOf. For example, the instantiation of paper to paper_class is defined by the

\[\text{class} \quad \text{instances} \quad \text{type} \]

The careful reader will notice that an infinite regression has now been introduced. For practical reasons, we make use of only the first two levels of the dimension.
2.2 Representation of Temporal Knowledge

In many applications, it is desirable to represent the progression of states of the domain over a period of time in a way that enables the system to answer historical queries not only with respect to its current state but also with respect to a previous state. In a nutshell, it should not only represent the history of the domain but also the system’s beliefs about this history. Recently, this feature has received considerable attention in the area of temporal and deductive relational databases [Sno97] [Gri88].

We have modified Allen’s well-known time interval framework [AIl6] to include an infinite set of constants corresponding to conventional dates and times (e.g., 1986/12/7 denoting December 7, 1986), semi-infinite intervals having conventional dates or times as one endpoint (e.g., 1986/10/25...), and the infinite interval AllTime.

Starting from this framework, we now show how temporal information is represented in Telos by means of the following example:

```telos
TELL TOKEN krypton
IN conf_paper
WITH
  author
  first_author: Brachman ( at 1986/10..* );
  : Fikes ( at 1987/1..* );
  : Levesque ( during 1986/10 )
  title
  : 'Krypton: Integrating Terminology and Assertion'
END
```

This command introduces krypton, an instance of the class conf_paper, in the knowledge base. The WITH clause asserts that Brachman is the first author of the paper about Krypton for an infinite time interval starting October 1986. Fikes became an author of the same paper three months later while Levesque was an author only during October 1986.

Assume that this command was processed successfully by the system on December 7th, 1988. As discussed in section 2.1, the first attribute definition above introduces the following propositions:

- p10 := <krypton, first_author, Brachman, t10>
- p11 := <p10, instanceOf, p4, t11>
- p12 := <p11, instanceOf, InstanceOf, t12>

Propositions p10 and p11 give historical information with respect to the attribute relationship between krypton and Brachman i.e., that Brachman is the first author of krypton. Time intervals like t10 and t11 are called history time intervals or history times since they represent temporal knowledge about the history of the application domain:

- t10 at 1986/10...
- t11 at 1986/10...

In contrast, we interpret the instantiation links to the class InstanceOf as storing information about the system’s beliefs about the history of the domain. For example, proposition p12 says that the system believes throughout the time interval t12 that p11 holds (i.e., krypton is an instance of paper during t11). Temporal relations about belief times are asserted automatically by the system when a TELL operation commits.
2.4 Extensibility Through User Defined Attribute Classes

So far, we have only demonstrated that the language allows the user to define simple attribute classes as attributes of individual classes. Using the full machinery of propositions, time, and assertions for the definition of attribute classes, makes the language fully extensible, without resorting to external program calls as in systems such as KEE [FK85]. For example, the attribute metaclass Unique introduced by

```
TELL CLASS Unique
    COMPONENTS <Class, unique, Class, Alltime>
    IN Attribute, M1_Class
    WITH
    integrityConstraint:
        $ (forall u/Unique, p,q/proposition)
            (p in u and q in u and to(p)=to(q) and
             when(p) overlaps when(q) => p=q) $ (during Alltime)
END
```

defines uniqueness of an attribute value within a class. This has already been used to induce properties of the attribute author for the class paper defined in section 2.1. More definitions of interesting standard attribute classes can be found in [KMSB].

2.5 The Telos Query Language

Telos offers the functions RETRIEVE and ASK for querying the knowledge base. While RETRIEVE uses only temporal and structural information to answer queries (as a temporal relational database would), ASK can be used either to prove that a closed formula of the assertion language follows from the knowledge base, or to find the propositions in the knowledge base that make a formula true. For example, the commands:

```
ASK: Fikes MemberOf krypton.author [over 1988 - believed over 1989/1/1]

ASK x/author: x MemberOf krypton.author
OF 1987
AS OF 1989/1/1
```

submitted to a KB defined by the TELL commands in this section return yes and (Brachman, Fikes) respectively.

In contrast to many similar predicative query languages (such as Prolog), Telos's definition of ASK must deal with sets introduced by multiple-valued attributes and time. The set expression x,n [0 <= 1 - believed [ <= 2] evaluates to the set of all propositions that are values of attributes of x with attribute category n. These attributes must be valid during time periods that are in relation t with t1 as believed by the system during time periods that are in relation t with t2. The temporal components in brackets are optional. If they are absent, the OF and AS OF classes of the ASK command specify the default history and belief times over which these set expressions are computed.

3 A Knowledge-Level Analysis of the Telos System

Following [BIL85], a Telos knowledge base is viewed as an abstract data type, characterized functionally by a limited set of operations which define what it can be asked or told about the domain.
The TELL operation can be functionally understood as follows:

\[ \text{TELL} : KB \times IC \times \text{Objs} \times \text{Time} \rightarrow KB \times IC \]

It takes place only if the extension \( KB \) of the new knowledge base under the above default rule [Rei80] is consistent and, if after the operation, the integrity constraints in the new set IC logically follow from \( KB \). The ASK operation

\[ \text{ASK} : KB \times IC \times \text{Query} \rightarrow \text{Answers} \]

takes arguments similar to TELL. For a closed query \( W \), it returns yes if \( W \) follows from \( KB \), no if \( \neg W \) follows from \( KB \) and unknown otherwise. For an open query, ASK returns all the possible substitutions for the free variables of the query that make it follow from \( KB \).

4 Implementing Telos: Problems and Solutions

In order to experiment with the full language as early as possible, a Telos prototype was implemented in Prolog [TK]. It offers a window interface with facilities for TELLing, ASKing, loading, and saving a Telos knowledge base. Two other implementations of earlier versions of the language [GS80] [JRRS88] with somewhat different strategies have been used for various applications (see next section).

A Telos knowledge base is structured as a collection of Horn clauses and stored as a Prolog database. Structural information about an object \( x \) is stored as Prolog facts of the following form:

\[
\text{instanceOf}(x, \text{class}, t1, t2). \\
\text{isa}(x, \text{general class}, t3, t4). \\
\text{attribute}(x, \text{attr category}, \text{attr label}, \text{attr class}, \text{value}, t5, t6).
\]

where \( t1 \) through \( t6 \) denote history and belief times. The actual propositions corresponding to the above relationships are computed only when needed. Deductive rules are translated into collections of Horn clauses and stored in the knowledge base. For integrity constraints, a precompiled form is stored in the knowledge base that can later on be used by the integrity constraint checker.

Any implementation of Telos must find efficient ways to deal with time. Using a forward-reasoning implementation of the time point algebra proposed by [VK86], the system maintains a network of point relations between the endpoints of every interval. Although these point relations can be used to verify the existence of any interval relation they cannot capture certain indefinite relationships expressible in Allen's framework. We accept this lack of completeness in query processing since the complexity of reasoning with respect to time is reduced from NP-hard to at most cubic [VK86]. A backward chaining version of the same reasoner was found (not to our surprise) to be very inefficient with respect to query processing time.

The query processor employed by ASK is very simple. Using the methods described in [LT85], we transform any query ASKing for the values of \( x1, \ldots, xn \) into a general clause with head \( \text{answer}(x1, \ldots, xn) \) and temporarily store the Horn clauses equivalent to it in the knowledge base. Then the execution of the goal \( ?- \text{answer}(x1, \ldots, xn) \) by a proper backward chaining meta-interpreter returns the answers to the query. This meta-interpreter makes the closed world assumption [Rei78] for every predicate in the knowledge base except for time predicates and

---

5 Objs represents the new object definitions and Time the current system time. The other symbols should be obvious.

6 Disjunctive temporal knowledge might be introduced in the knowledge base thus creating an inconsistency.
Assume that the class paper described in section 2.1 is part of the requirements model for an information system. Since author attributes are unique, it appears natural to choose authorname as the key for the corresponding relation PaperRel, by instantiating the attribute class implements with a specific dependency labeled derived from that relates the authorname with the author attribute. The validity interval for this dependency, and therefore for the dependent key attribute authorname, might represent the initial version of the software system under development (dashed lines in fig. 2). Later, it might be noted that some authors submit more than one paper and, therefore, the author attribute can no longer be considered unique. Telos' time mechanism could then be used to terminate (belief in) the initial dependency and create an alternative, final one, assigning a unique submission no as a key (dotted lines in fig. 2). Suppose, however, some letters of acknowledgment had already been sent based on the initial key; the corresponding objects could still be retrieved by referring to the old version of the knowledge base.

The example has demonstrated the use of attributes for dependency modelling, and the use of time for version modelling in software development processes. Of course, the actual software process model explores much more sophisticated versioning and configuration mechanisms for designing-in-the-large than exhibited in this example; also, a Telos-specific graphics-oriented environment totalling about 30,000 lines of Prolog and interface code has been built for guiding users through the software development process [JFR88].

6 Discussion and Conclusions

We consider the treatment of attributes and the representation of temporal knowledge as key contributions of this work to the state of the art for knowledge representation languages. Below we expand on these points and relate Telos to some other proposals for knowledge representation languages.

As we saw in section 2, the treatment of attributes as first-class "units" in the language, on a par with individuals, leads to several interesting consequences: (a) the (weak!) typing mechanism offered by instantiation applies equally to attributes and individuals, (b) it is possible
tiple concurrent users for a single knowledge base, employing query optimization techniques such as ones developed for databases [Jar].

Acknowledgements

We gratefully acknowledge the contribution of insightful ideas, suggestions and moral support from Sol Greenspan (GTJ Laboratories, Waltham, MA) whose thesis provided a rationale and a springboard for this research, Yannis Vassiliou, Thodoros Topalogi, Manolis Marakis and others (Institute of Computer Science, Crete, Greece) for serving as first users of the language, John Gallagher and Levy Solomon (SCS Technische Automation und Systeme GmbH, Hamburg, Germany), who did the first implementation of CML, Matthias Jarke and his research group (University of Fassau, Germany) who carried out another implementation used in a number of applications, as well as other members of the ESPRIT projects LOKI and DAIDA. Last, but not least, we would like to thank Alex Borgida, Lawrence Chung, Brian Nixon and other members of the Taxis group at the University of Toronto for providing a friendly and stimulating research environment.

The development of the language was financially supported by the University of Toronto, the National Science and Engineering Research Council of Canada, the Computer Science Institute of the Cretan Research Centre and the Commission of European Communities through ESPRIT projects LOKI and DAIDA.

References


