Telos
Manolis Koubarakis
National and Kapodistrian University of Athens, \url{http://www.di.uoa.gr/~koubarak}

SYNONYMS
none

DEFINITION
Telos\textsuperscript{1} is a knowledge representation language designed especially to support the development of information systems. Telos is based on the premise that information system development is knowledge-intensive and that the main design goal of any language intended for the task should be to formally represent the relevant knowledge. Telos is founded on core concepts from data modeling and knowledge representation, and shares ideas with semantic networks and frame systems, semantic and object-oriented data models, logic programming and deductive databases. The main features of Telos include: a structurally object-oriented framework which supports aggregation, generalization and classification; a novel treatment of attributes as first class citizens in the language; a powerful way of defining meta-classes; an explicit representation of time; and facilities for specifying integrity constraints and deductive rules.

HISTORICAL BACKGROUND
The research on Telos follows the paradigm of a number of software engineering projects initiated around the 1990s with the premise that software development is knowledge-intensive and that the primary responsibility of any language intended to support this task is to be able to formally represent the relevant knowledge. Thus, Telos was designed as a knowledge representation language that is intended to support software engineers in the development of information systems throughout the software lifecycle.

Telos has evolved from RML (a requirements modeling language developed in Sol Greenspan’s Ph.D. thesis), and later CML (presented in the Master thesis of Martin Stanley at the University of Toronto). The main difference between RML and CML is that CML adopts a more sophisticated model for representing knowledge, and supports the representation of temporal knowledge and the definition of meta-classes. Telos is essentially a “cleaned-up” and improved version of CML which was originally defined and implemented in the Master thesis of Manolis Koubarakis at the University of Toronto. The original paper on Telos is [7]. Ontological and semantical issues for Telos are discussed in [10]. The history of knowledge representation languages for information systems development related to Telos is surveyed in [3]. An important dialect of Telos is O-Telos defined in the Ph.D. thesis of Manfred Jeusfeld at the University of Passau, and implemented in the ConceptBase system [4]. Since ConceptBase is the most mature implementation of Telos available today, this entry uses the ConceptBase syntax for Telos.

SCIENTIFIC FUNDAMENTALS
The main (and essentially the only) concept of Telos is the proposition. Propositions are used to model any aspect of the application domain. Propositions have unique identities and are distinguished into individuals and attributes. Individuals are intended to represent entities in the application domain (concrete ones such as

\textsuperscript{1}From the Greek word τέλος which means end; the object aimed at in an effort; purpose.
John Doe, or abstract ones such as the class of all persons). Attributes represent binary relationships between entities (concrete or abstract). Two special kinds of attribute propositions exist: instantiation propositions and specialization propositions. The proposition abstraction gives great flexibility to Telos users. Everything in the application domain that is represented by a proposition (e.g., an entity or a relationship) immediately becomes a first-class citizen of the knowledge base.

**Propositions**

Every proposition \( p \) consists of an identifier, a source, a label and a destination, denoted by the functions \( \text{id}(p) \), \( \text{from}(p) \), \( \text{label}(p) \) and \( \text{to}(p) \). For example, the following are propositions:

- \( P_1: [..., \text{Martin}, ...] \)
- \( P_2: [..., "21 Elm Avenue", ...] \)
- \( P_3: [\text{Martin, homeAddress, "21 Elm Avenue"}] \)
- \( P_4: [..., \text{Person}, ...] \)
- \( P_5: [..., \text{GeographicLocation}, ...] \)
- \( P_6: [\text{Person, address, GeographicLocation}] \)

Propositions in Telos are what *objects* are in object-oriented formalisms but also what *statements* are in logic-based formalisms. Thus, an application can use the above propositions to represent the following pieces of knowledge:

- \( P_1: \) There is somebody called Martin.
- \( P_2: \) There is something called “21 Elm Avenue”.
- \( P_3: \) Martin lives in 21, Elm Avenue.
- \( P_4: \) There is an abstract concept, the class of all persons.
- \( P_5: \) There is an abstract concept, the class of all geographic locations.
- \( P_6: \) Persons have addresses that are geographic locations.

\( P_1, P_2, P_4 \) and \( P_5 \) are individual propositions while \( P_3 \) and \( P_6 \) are attribute propositions. The source and destination components of an individual proposition are not important, thus they are shown as “...”. Notice that while \( P_1 \) and \( P_2 \) represent concrete individuals, \( P_4 \) represents an abstract one, the class of all persons. Similarly, \( P_3 \) represents a concrete relationship (relating Martin with his address) while \( P_6 \) represents an abstract one (relating the class of all persons with the class of all geographic locations).

Let us continue with some examples of special propositions:

- \( P_7: [P_1, \ast \text{instanceOf}, P_4] \)
- \( P_8: [P_3, \ast \text{instanceOf}, P_6] \)
- \( P_9: [..., \text{Employee}, ...] \)
- \( P_{10}: [P_9, \ast \text{isA}, P_4] \)

\( P_7 \) and \( P_8 \) are *instantiation propositions*. \( P_7 \) represents the fact that Martin is a member of the class of all persons. \( P_8 \) represents the fact that the concrete relationship relating Martin with his address is an instance of the abstract relationship relating the class of all persons with the class of all geographic locations. Finally, \( P_{10} \) is a *specialization proposition* asserting that every employee is a person.

A graphical view of some of the above propositions is given in Figure 1.

**Organizing propositions**

Propositions (individual or attribute ones) can be organized along three dimensions: decomposition/aggregation, instantiation/classification and specialization/generalization.
Figure 1: A graphical view of a set of Telos propositions
The aggregation dimension enables one to see an entity of the application domain as a collection of propositions with a common proposition as source. For example, individual Martin can be seen to be the following aggregation:

\[
\{ \text{Martin}, \\
\text{[Martin, age, 35]}, \\
\text{[Martin, homeAddress, "21 Elm Avenue"]}, \\
\text{[Martin, workAddress, "10 King's College Road"]} \}
\]

The classification dimension calls for each proposition to be an instance of one or more generic propositions or classes. Classes are themselves propositions, and therefore instances of other, more abstract classes. For example, Person is a class and Martin is an instance of this class. Similarly,

\[\text{[Person, address, GeographicLocation]}\]

is a class and

\[\text{[Martin, homeAddress, "21 Elm Avenue"]}\]

is an instance of this class.

With respect to the classification dimension, propositions can be distinguished into:

- **Tokens:** propositions having no instances and intended to represent concrete entities in the application domain.
- **Simple classes:** propositions having only tokens as instances.
- **Meta-classes:** propositions having only simple classes as instances.
- **Meta-meta-classes:** propositions having only meta-classes as instances.
- ...

Thus, classification in Telos defines an unbounded linear order of planes of ever more abstract propositions. Implementations restrict this unbounded hierarchy (e.g., ConceptBase restricts it to 4 levels: tokens to meta-meta-classes). There are also \(\omega\)-classes with instances along more than one plane:

- **Proposition:** Contains all propositions as instances.
- **Class:** Contains all classes as instances.
- **Token:** Contains those individuals that may never have instances themselves.
- **SimpleClass:** Contains individuals that may have instances which are tokens.
- **MetaClass:** Contains individuals that may have simple classes as instances.
- **MetametaClass:** Contains individuals that may have meta-classes as instances.
- ...

Classification in Telos is a form of weak typing: the classes of which a structured object is an instance determine the kinds of attributes it can have optionally, and the properties it must satisfy. For example, by virtue of being an instance of Person, Martin can have attributes that are instances of the attribute class

\[\text{[Person, address, GeographicLocation]}\].

These zero or more attributes can have arbitrary labels, e.g., homeAddress and workAddress, but their values must be instances of GeographicLocation.

Finally, classes in Telos can be specialized along generalization or ISA hierarchies. For example, Person may have subclasses such as Professor, Student, and TeachingAssistant. Classes may form a partial order, rather than a tree (i.e., multiple inheritance is supported). Non-token attributes of a class are inherited by more specialized ones and can be refined. Inheritance in Telos is strict rather than default.
Interacting with Telos knowledge bases
A few examples of Telos are now given. The example application considered is the development of an information system to support organizing international scientific conferences. The knowledge to be represented in this case is about entities such as papers, authors, conferences etc.

The original definition of Telos in [7] defines the operations TELL, UNTELL, RETELL and ASK for interacting with a Telos knowledge base. These operations can be used to add new knowledge, discard existing knowledge, update existing knowledge and query a knowledge base, respectively. Implementations such as ConceptBase have followed the original definition and offer these (and other) operations. The above operations have Telos statements such as the following as their means of interaction with a knowledge base:

\[
\text{Individual p133 in Token, Paper with}
\]

\[
\text{author}
\]

\[
\text{firstAuthor: Stanley;}
\]

\[
\text{secondAuthor: LaSalle;}
\]

\[
\text{thirdAuthor: Wong}
\]

\[
\text{title}
\]

\[
\text{called: "The language Telos"}
\]

The above statement introduces an individual with name p133. The in clause specifies the classes of which p133 is an instance (in this case, the predefined class Token and the application class Paper). The with clause introduces p133's attributes. The first attribute of p133 has label firstAuthor and is an instance of an attribute class which has source Paper and label author (the latter is denoted by the attribute category author). Before introducing individual paper p133, one might have defined the class of all papers using the following statement:

\[
\text{Individual Paper in SimpleClass with}
\]

\[
\text{attribute}
\]

\[
\text{author: Person;}
\]

\[
\text{referee: Person;}
\]

\[
\text{title: String;}
\]

\[
\text{pages: Integer}
\]

end

A class definition prescribes the attributes that can be associated with its instances: p133 can have author, referee, title and page attributes as we saw previously, because it is an instance of class Paper that has these attribute classes. Moreover, \([p133, \text{firstAuthor}, \text{Stanley}]\) is an instance of attribute class \([\text{Paper, author, Person}]\) in exactly the same sense that p133 is an instance of Paper.

Once Paper has been defined, one can introduce specializations such as InvitedPaper using the isA clause of class definitions:

\[
\text{Individual AcceptedPaper in SimpleClass isA Paper with}
\]

\[
\text{attribute}
\]

\[
\text{session: ConfProgrammeSession}
\]

end

AcceptedPaper inherits all attributes from Paper and adds a session attribute, to indicate the programme session during which the accepted paper will be presented.
**Metaclasses**

Metaclasses are a very powerful concept for modeling power and extensibility in Telos. It is the metaclass mechanism combined with its other features that makes Telos a powerful modeling language (one might wonder about this, since Telos offers only very simple primitives). From a modeling point of view, one can use Telos metaclasses in the following situations:

- To define concrete attributes of classes e.g., cardinality of a class. This is exactly the same to what a simple class does for its instances (tokens).
- To group together semantically similar classes of a domain in a generic way. For example, in the conference organization example, the classes `Paper, Announcement, Letter, Memo` could be grouped under the metaclass `DocumentClass`.
- To define concepts that are built-in in other frameworks e.g., necessary attributes, single-valued attributes etc.
- To do other forms of meta-level logical reasoning (again, for language expressibility).

Let us revisit the conference organization example and define:

```telos
DocumentClass in MetaClass with
    attribute
        source: AgentClass;
        content: SimpleClass;
        destination: AgentClass;
        cardinality: Integer
end
```

We can now define the class `Paper` as follows:

```telos
Paper in DocumentClass with
    source
        author: Person;
        content
        title: String;
        abstract: String
        cardinality
            how_many: 120
end
```

Note that attribute categories such as `source` introduced in metaclass `DocumentClass` are then used to define attributes for the instance class `Paper` (this mechanism is the same along the instantiation hierarchy).

**Integrity constraints and deductive rules**

Telos borrows the notions of integrity constraints and deductive rules from logic-based formalisms such as deductive databases. **Integrity constraints** are formulas that express conditions that knowledge bases should satisfy. They are used to express rich language or application semantics that cannot possibly be expressed only by the structural framework of Telos. **Deductive rules** are formulas that can be used to derive new knowledge. Integrity constraints and deductive rules in Telos are expressed in appropriately defined assertional languages that are subsets of first-order logic [7, 4].

Integrity constraints and rules are defined as attributes of Telos classes that are instances of the built-in object `Class`. For example, the following Telos statement defines well-understood constraints and rules regarding employees, their managers and their respective salaries.
Class Employee with
rule
   BossRule: $ \forall e / \text{Employee} \ m / \text{Manager} \\
   \ (\exists d / \text{Department} \\
   \ (e \ \text{dept} \ d) \ \text{and} \ (d \ \text{head} \ m)) \implies (e \ \text{boss} \ m) \$ 
constraint
   SalaryBound: $ \forall e / \text{Employee} \ b / \text{Manager} \ x, y / \text{Integer} \\
   \ (e \ \text{boss} \ b) \ \text{and} \ (e \ \text{salary} \ x) \ \text{and} \ (b \ \text{salary} \ y) \implies x \leq y \$
end

Language extensibility through metaclasses and integrity constraints
In Telos, one can use integrity constraints together with the metaclass mechanism to define concepts that are
built-in in other representational frameworks. For example, in many object-oriented models one can constrain an
attribute to be single-valued using some built-in construct of the model. In Telos, one can do this by using only
the primitive mechanisms of the language as follows. First, one defines the class Single:\footnote{The syntax of this statement is from the original paper of Telos [7] (O-Telos allows one to specify the same thing in a slightly more complex way).}

\begin{verbatim}
Class Single
   components [Class, single, Class]
   in AttributeClass, MetaClass with
   integrityConstraint
      : $ \forall u / \text{Single} \ p, q / \text{Proposition} \\
      \ (p \ \text{in} \ u) \ \text{and} \ (q \ \text{in} \ u) \ \text{and} \ \text{from}(p) = \text{from}(q) \implies p = q \$
end
\end{verbatim}

Then, one uses attribute class Single in the definition of class Paper:

Individual Paper in SimpleClass with
   attribute
      author: Person;
      referee: Person
   single
      title: String;
      pages: Integer
end

Now in every instance of Paper, a title attribute is constrained to be single-valued due to the integrity constraint
and the instantiation relationships introduced by the above Telos statements.

Query languages for Telos
The papers [7, 11, 4] give various query languages for Telos knowledge bases ranging from ones based purely on
first-order logic [7] to ones exploiting the structurally object-oriented features of the language as well [11, 4]. The
paper [8] presents work on a knowledge base management system based on Telos.

Temporal knowledge in Telos
The original paper of Telos [7] presents a powerful framework for representing and reasoning about temporal
knowledge. In Telos, the history of an application domain is modeled by augmenting propositions with a history
time i.e., an interval representing the time during which these facts are true in the application domain. Historical
knowledge in Telos is allowed to be incomplete and a modification of Allen’s interval algebra [1] is used to capture
the relevant knowledge.

A knowledge base records essentially the beliefs of the system, which may be distinct from the actual state
of the world at that time. So, for example, the title of a paper might have been changed in March, but the knowledge base is only told of it in May. Or one may make a correction to some previously told fact. Just like it represents the full history of an application domain, Telos also records the full history of its beliefs. For this reason, Telos represents belief times; these are intervals associated with every proposition in the knowledge base, which commence at the time when the operation responsible for the creation of the corresponding proposition was committed.

For efficiency reasons, implementations of Telos such as ConceptBase [4] have restricted the kinds of temporal knowledge that can be represented.

Telos and RDF
Telos is probably the pre-Web knowledge representation language most closely related to the Resource Description Framework (RDF) and the RDF Vocabulary Description Language or RDF Schema proposed by the W3C for representing knowledge about Web resources (see e.g., http://www.w3.org/TR/rdf-primer/). This relationship has been exploited by the prominent RDF query language RQL defined by ICS-FORTH [6] but also in the O-Telos-RDF proposal [9].

KEY APPLICATIONS*
Telos was designed as a knowledge representation language that is intended to support software engineers in the development of information systems throughout the software lifecycle [7]. The strengths of the language made it the choice of many prominent research projects in Europe and North America including DAIDA [5], ITHACA [2] and others.

URL TO CODE*
The most mature implementation of Telos is the ConceptBase system available at http://conceptbase.cc/.

CROSS REFERENCE*
Semantic Data Models, Object Data Models, Meta Model, RDF Schema

RECOMMENDED READING
Between 5 and 15 citations to important literature, e.g., in journals, conference proceedings, and websites.


