Telos: Representing Knowledge About Information Systems

John Mylopoulos, Alex Borgida,
Matthias Jarke and Manolis Koubarakis

Department of Computer Science
University of Toronto

We describe *Telos*, a language intended to support the development of information systems. The design principles for the language are based on the premise that information system development is *knowledge-intensive* and that *the primary responsibility of any language intended for the task is to be able to formally represent the relevant knowledge.* Accordingly, the proposed language is founded on concepts from knowledge representation. Indeed, the language is appropriate for representing knowledge about a variety of worlds related to an information system, such as the subject world (application domain), usage world (user models, environments), system world (software requirements, design), and development world (teams, methodologies).

The paper introduces the features of the language through examples, focusing on those provided for describing meta-concepts that can then be used to describe knowledge relevant to a particular information system. Telos' features include: an object-centered framework which supports aggregation, generalization and classification; a novel treatment of attributes; an explicit representation of time; and facilities for specifying integrity constraints and deductive rules. Actual applications of the language are reviewed through further examples and a formalization of the language is sketched.

1 Introduction

Language facilities have been a key vehicle for advances in software productivity since the introduction of assembler in the early '50s, the first high level programming languages in the mid-50s, and the languages supporting encapsulation/modularization in the 70's. But programming accounts for only a small fraction of the total effort and cost of producing a software system.

This paper describes a language that is intended to support software engineers in the development of information systems throughout the software lifecycle. This language is *not* a programming language. Following the example of a number of other software engineering projects, our work is based on the premise that information system 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.*[1] Accordingly, the proposed language is founded on concepts from knowledge representation [12]. Indeed, the language is viewed as a knowledge representation language appropriate for representing knowledge about an information system. This viewpoint leads to an eclectic approach on what mathematical concepts are relevant to software development and a rationalization of why some notations are more significant than others.

How is a knowledge representation language different from other types of languages, such as programming or design languages, formal languages or natural languages? According to [12] (pages xiv–xv):

> In order to have an explicit knowledge base, a system must rely on some well-specified language for encoding its beliefs. That role is played by a *knowledge representation language*. Beyond that, in just about all imaginable cases of interest, a system will be concerned with more than just the literal set of sentences (or frames, or production rules, or whatever) representing what it knows. A representation system must also provide access to facts *implicit* in the knowledge base. In other words, a representation component must perform *automatic inferences* for its user.

The ingredients then of a knowledge representation language include *a (formal) notation*, and *a deductive mechanism for drawing inferences* from a body of statements (the knowledge base) represented in that notation[2]. In addition, there is a need to assign some sort of "meaning" to statements – *the semantics of the notation*; this meaning must be respected by the deductive process. Finally, to be effective in large projects, a knowledge representation language must offer *facilities to structure and organize the knowledge base.*

The language presented in this paper is called *Telos*[3]. Like any useful language, Telos has been shaped by its subject matter. But what knowledge needs to be represented about an information system? To begin with, knowledge about the environment within which the system will function and how the system is expected to interact with that environment. Second, the kind of information the system will be expected to store and the meaning of that information with respect to its intended subject matter. Third, knowledge about the design and implementation of the information system, which can be used during initial system development as well as during system maintenance. Fourth, knowledge about design decisions that led to the particular design/implementation, along with appropriate justifications that relate these decisions to performance or other non-functional requirements. Finally, information on the development process itself that led to the system, including the methodology used, the team of developers involved, different system versions and the like. Our work is based on the premise that all this, and other, knowledge about an information system is useful during its initial development, subsequent deployment and use, maintenance and reuse. This, rather ambitious, viewpoint is grounded in our own experiences within the ESPRIT project DAIDA which is concerned with the construction of a complete information system development environment [34].

To deal with these ideas, and to meet some of the goals of a good KR language, Telos provides a number of novel facilities: representing and reasoning about (possibly incomplete) temporal knowledge; particularly general forms of conceptual structuring mechanisms such as *generalization* and *classification*; supporting linguistic extensions through the definition of meta-attributes, in order to cope with the multitude of subject matters. Also, Telos adapts concepts from deductive databases [24] for query processing and integrity enforcement. In keeping with general principles of good language design, attempts were also made to maintain *uniformity* and *simplicity.*

[1] See [47] for a survey of knowledge-based software engineering projects.

[2] There is a tension between the increased "expressive power" of a notation – the ability to express certain facts and make certain kinds of deductions – and the complexity of the computations involved.

[3] From the Greek word τέλος which means *end*; the object aimed at in an effort; purpose.

Telos has evolved from RML (a requirements modeling language developed in a doctoral dissertation by Greenspan [25]), and later CML (described and formalized in [52]). The major difference between RML and CML is that the latter adopts a more sophisticated data structure for representing knowledge, and supports the representation of complex temporal knowledge and the definition of meta-concepts. Telos, on the other hand, is a "cleaned-up" version of CML, both from a language definition and an implementation perspective, which has been implemented and tested with a variety of knowledge representation tasks related to information system development. It has been used both in the LOKI and DAIDA projects and the section on applications of the languages is based on those experiences.

The paper is organized as follows. Section 2 presents and motivates the basic features of the language. In section 3, the nature and applications of metaclasses is investigated through examples. Section 4 surveys some of the applications that have been considered in the context of information system development. The formalization of Telos is reviewed in section 5, while section 6 discusses related work. Finally, section 7 summarizes the contributions of the language and suggests directions for further research.

2 Features of Telos

Telos provides facilities for constructing, querying and updating structured knowledge bases (KBs). The operations `TELL`, `UNTELL`, and `RETELL` are offered to extend or modify a KB while the operations `RETRIEVE` and `ASK` can be used to access it. This section introduces Telos and illustrates its use.

2.1 Structured Knowledge Bases

A Telos knowledge base consists of structured objects built out of two kinds of primitive units: *individuals* and *attributes*. Individuals are intended to represent entities (concrete ones such as `John`, or abstract ones such as `Person`), while attributes represent binary relationships between entities or other relationships. An important and distinctive feature of Telos is that individuals and attributes are treated uniformly by the mechanisms for structuring a KB; they are collectively referred to by the term *"proposition"*.

Every attribute `p` consists of a *source*, a *label* and a *destination*, which can be retrieved through the functions `from(p)`, `label(p)` and `to(p)`. An attribute proposition will be represented for the moment by a 3-tuple, e.g., [`Martin`, `age`, 35].

Propositions (individual or attribute) are organized along three dimensions, referred to in the literature as the *aggregation*, *classification* and *generalization* dimensions [31].

Structured/aggregate objects consist of collections of attributes that have a common proposition as source. For example, the individual `Martin` may aggregate the cluster of propositions

```
{Martin, [Martin, age, 35], [Martin, homeAddr, '21 Elm Avenue'],
         [Martin, workAddr, '10 King's College Road']}
```

This indicates, among others, that `Martin` has two (momentarily unrelated) attributes with labels `homeAddr` and `workAddr`, and values `'21 Elm Avenue'` and `'10 King's College Road'` respectively. Attributes may also represent abstract relationships such as `[Person, address, GeographicLocation]`, intended to represent the concept of address relationships between persons and geographic locations.

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. In this way both `Person` and `[Person, address, GeographicLocation]` are classes, with individual instances which are particular individuals and relationships respectively (e.g., `Martin` and `[Martin, homeAddr, '21 Elm Avenue']`. Generally, propositions are classified into *tokens* – propositions having no instances and intended to represent concrete entities in the domain of discourse, *simple classes* – propositions having only tokens as instances, *metaclasses* – having only simple classes as instances, *metametaclasses*, and so on. This classification defines an unbounded linear hierarchy of planes of ever more abstract propositions.

There are also ω-classes with instances along more than one such plane. For example, the class `Proposition` has all propositions as instances while `Class` has all generic propositions as instances. Figure 1 shows the structure of the classification dimension, with sample propositions at various levels.

Figure 1: A simple semantic net

Instantiation is treated as a form of weak typing mechanism: the classes of which a structured object is an instance determine the kinds of attributes it can have and the properties it must satisfy. For example, by virtue of being an instance of `Person`, `Martin` can have attributes that are instances of `[Person, address, GeographicLocation]`. Such attributes can have arbitrary labels, such as `homeAddr` and `workAddr`, but their values must be instances of `GeographicLocation`.

Classes can be specialized along *generalization* or *ISA hierarchies* For example, `Person` may have subclasses such as `Professor`, `Student`, and `TeachingAssistant`. The classes may form a partial order, rather than a tree. Note that IsA hierarchies are orthogonal to the classification dimension: all these classes could be instances of `PersonClass`. As discussed later, non-token attributes of a class are inherited by more specialized ones, and inheritance is strict rather than default.

2.1.1 Interacting with a Knowledge Base: An Example

Consider the problem of developing an Information System to support organizing international scientific conferences. As part of the requirements model, the designer needs to describe the entities about which information will be maintained, such as papers, authors, conferences and the like. The following `TELL` operation introduces an object to model a paper submitted to, say, an IFIP World Congress [42]:

```
TELL TOKEN martian IN Paper WITH
      author
        firstAuthor: Stanley;
                   : LaSalle;
                   : Wong
      title
        :'The MARTIAN system'
END
```

This operation defines a token with external identifier `paper` and several associated attributes[4]. The `IN` clause specifies the classes of which `martian` is an instance, while the `WITH` clause introduces `martian`'s attributes. For example, the first attribute 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`). The second attribute has no external label and it is an instance of the same attribute class; this attribute is going to acquire a system-generated label.

Continuing with the requirements model, we can use `TELL` to define generic concepts which determine the database schema. Thus, the class `Paper` (which is an instance of the built-in class `SimpleClass`) has associated a number of attribute classes:

```
TELL CLASS Paper IN SimpleClass WITH
      attribute
        author: Person;
        referee: Person;
        title: String;
        pages: 1..100
END
```

As indicated earlier, a class definition prescribes the attributes that can be associated with its instances: `martian` can have `author, referee, title` and `page` attributes *because* it is an instance of some class (i.e., `Paper`) that has attribute classes using these labels. Moreover, `[martian, firstAuthor, Stanley]` is an instance of `[Paper, author, Person]` in exactly the same sense that `martian` is an instance of `Paper`.

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

[4] Each Telos proposition has a unique internal identifier and zero or more external identifiers that can be used in Telos expressions to refer to that proposition.

```
TELL CLASS AcceptedPaper IN SimpleClass ISA Paper WITH
      attribute
         pages: 1..15
         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. It also refines the restriction on page length to indicate that published papers can only be up to 15 pages long.

A token may now be defined which instantiates more than one of these classes. For example, if we also have a class `ReceivedFinalVersion` of papers which have been received in camera-ready form, we can add to `martian` additional information related to these aspects:

```
TELL TOKEN martian  IN AcceptedPaper, ReceivedFinalVersion WITH
      session
        : applicationsI
      dateReceived
        : 1989/2/3
END
```

Note that `martian` can have attributes "induced" by the attribute classes of `AcceptedPaper` and `ReceivedFinalVersion`. In the case of an attribute appearing in both classes, the value must be consistent with *both* class definitions.

Concerning the interaction of subclass and instance-of hierarchies, we have, as usual, that instances of a class are also instances of its superclasses. In other words, if `A ISA B` and `C IN A`, then `C IN B`. For representational structures that fully support classification (and therefore offer the dimension illustrated in Figure 1) the reader may wonder whether `A ISA B` and `B IN C` implies `A IN C`. For Telos, this implication is not supported because there seem to be cases where it is unwarranted. Consider, for example, the binary relations `Spouse` and `Wife`. Obviously `Wife ISA Spouse` and `Spouse IN SymmetricRelation`. However, we do not want to conclude here that `Wife IN SymmetricRelation`.

2.2 Representation of Temporal Knowledge

Most application domains are not static: they exhibit a history of changes through time. A Telos model of a domain captures the full history of its evolution, rather than just the latest snapshot. For this, Telos adapts a framework for representing and reasoning with temporal knowledge proposed in [2]. This representation is based on the notion of an *time interval*, where seven exclusive temporal relations (`equals, meets, before, overlaps, during, starts, ends`) and their inverses are used to characterize all possible relationships of two intervals on a linear time line. Thus, in contrast to temporal databases [50], Telos can represent incomplete information about time, e.g., a paper having been submitted sometime before 1989/2/2. Telos' modifications of this approach include (i) slight changes to the definitions of the thirteen temporal relationships, mostly dictated by language design considerations; (ii) incorporation of temporal constants, such as conventional dates and times (e.g., `1988/12/7` denoting December 7, 1988), semi-infinite intervals having conventional dates or times as one endpoint (e.g., `1986/10/25..*`), the infinite interval `Alltime` and the special interval `Now` denoting the current system time; (iii) restricting the power of temporal assertions that can be told to the system.

With such a framework on hand, it is possible to represent temporal information as shown by the following revised definition of `martian`:

```
TELL TOKEN martian IN Paper ( at 1986/10..* ) WITH
     author
       firstAuthor: Stanley ( at 1986/10..* );
                  : LaSalle ( at 1987/1..* );
                  : Wong ( before 1987/5 )
     title
         : 'The MARTIAN system'
END
```

This operation introduces the token `martian` in the knowledge base (we suppose it was not there already). The `IN` clause makes `martian` an instance of the class `paper` for an unbounded time interval starting October 1986. Similarly, the `WITH` clause asserts that `Stanley` is the first author of `martian` during the interval `1986/10..*`, `LaSalle` is an author during the interval `1987/1..*` while `Wong` was an author for some time before May 1987 (but we do not know the exact time). The corresponding attribute propositions (now 4-tuples) are shown below.

```
[martian, firstAuthor, Stanley, 1986/10..*]
[martian, ..., LaSalle, 1987/1..*]
[martian, ..., Wong, T32]
```

Henceforth, every attribute proposition `p` has a *duration* component which can be accessed with the expression `when(p)`.

The history of the application domain can be modeled by augmenting KB facts with a *history time* i.e., an interval representing the time during which these facts are true in the application domain. History time is useful not only for tokens but also for generic propositions: for example, the *definition* "personal deduction" in an Income Tax Act, may only apply for this year.

A KB 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 KB is only told of it in May. Or we 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 (technically speaking `costart`) at the time when the operation responsible for the creation of the corresponding proposition was committed. All belief time intervals are assumed to be semi-infinite until the system is informed otherwise. So, once the system has been `TELL`ed something, it keeps believing it, until it is explicitly required to revise its beliefs. The operations `UNTELL` or `RETELL` (see section 2.5) will cause precisely such belief revision. Thus, system beliefs "persist" until they are explicitly revised. Similar facilities have been proposed by [50] for temporal databases.

The syntax of the language, illustrated by the above example, is restricted in the sense that it only allows a single temporal relationship to appear in each of the temporal components of a given definition. This is in contrast with Allen's original framework where *sets* of relationships were allowed between time intervals (e.g., `PaulsDateOfBirth (before during) 1975` in order to express further kinds of incomplete knowledge. Thus, after a series of definitions such as that for `martian`, in Telos two intervals `t1, t2` are either unrelated (i.e., any of the thirteen temporal relationships may hold between them) or are related through exactly one relationship. The reasons for this expressive retreat are strictly pragmatic: verifying the consistency of a network of temporal relations for Allen's algebra is NP-hard, as is computing all the consequences of a network. However, reasoning with certain subsets of the framework, including the one adopted here, is tractable [55].

2.3 Rules and Constraints

A typed first order assertion sublanguage is offered as means for specifying *integrity constraints* and *deductive rules*. Well-formed formulas of this language are special objects in the Telos ontology and are allowed to appear quoted as attribute values of propositions. For example, the integrity constraint of the definition below ensures that an author cannot referee her own paper, while the deductive rule states that an author address is also a reply address.

```
martian.author [during 1987/1] = {Stanley, LaSalle}
martian.author [before 1988] = {Stanley, LaSalle, Wong}
```

Figure 2: Examples of the use of the dot function

```
TELL CLASS Paper IN SimpleClass WITH
    integrityConstraint
        :$ (∀y/Person)
            (y ∈ this.author ⟹ ¬(∃t/Time) y ∈ this.referee [at t]) $
    deductiveRule
        :$ (∀x/Paper)(∀z/Address)
            (z ∈ x.author.address ⟹ z ∈ x.replyAddress) $ (at Alltime)
END
```

Note that deductive rules are constrained to be in a simple form to improve efficiency: the antecedent of the rule must be a conjunction of atomic formulas and the consequent must be a single positive atomic formula.

The assertion language is naturally integrated with the existing framework by treating Telos classes as ranges for quantifiers. From a computational point of view, this choice offers some of the advantages associated with sorted logics [22]. The following functions manipulate attributes and their values:

- The dot function `x.l [r1 t1]` evaluates to the set of values of the attributes of proposition `x` which belong to the attribute class labeled `l` during intervals which are in relation `r1` with `t1`.

- The hat function `x^l [r1 t1]` evaluates to the set of values of the attributes of proposition `x` with label `l` during an interval which is in relation `r1` with `t1`.
- The bar function `x|l [r1 t1]` evaluates to the set of attribute propositions with source `x` which are instances of the attribute class labeled `l` during intervals which are in relation `r1` with `t1`.
- The exclamation mark function `x!l [r1 t1]` evaluates to the set of attribute propositions with source `x`, label `l` and duration which is in relation `r1` with `t1`.

The time constraint in the above functions are optional: if they are absent, appropriate defaults are adopted by the system. Figure 2 shows the situation for the example in section 2.2. Finally, the special identifier `this` is used as follows: an assertion $\phi(\texttt{this})$ defined on class `C` is an abbreviation for $(\forall \texttt{x}/\texttt{C})\phi(\texttt{x})$.

Rules and constraints can be given history time intervals like any other attribute values, corresponding to periods during which the assertions hold in the knowledge base. Rules and constraints can also refer to history time explicitly. Integrity constraints must also be given a belief time: the time period during which the beliefs of the system are constrained. For example, if the above operation was processed on December 5, 1988, the integrity constraint included would enforce in every belief state after December 5, 1988 that no instance of the class `Author` can be an author and a referee of the same paper[5]. These features can be exploited for activating and de-activating rules and constraints during certain times. In this way, we get facilities similar to triggers or deamons of active databases.

2.4 Querying the Knowledge Base

Telos offers the operations `RETRIEVE` and `ASK` for querying the knowledge base. `RETRIEVE` uses only temporal and structural information to answer queries, while `ASK` uses all the knowledge available. `RETRIEVE` is a much more efficient operation, since only a few built-in inferences (e.g., inheritance) are performed. Both operations 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 given formula true.

$$\texttt{ASK} \;:\; \texttt{LaSalle} \in \texttt{martian.author}\,[\texttt{over 1988}]\ \texttt{BELIEVED at 1989/1/1}$$

$$\texttt{ASK x/Author} :\; \texttt{x} \in \texttt{martian.author TRUE at 1987 BELIEVED at 1989/1/1}$$

For example, when the above queries are evaluated against the current knowledge base, the system returns `yes` and {`Stanley, LaSalle`} respectively.

Queries can refer to history time explicitly (e.g., first query above). Alternatively, the optional clause `TRUE` can be used to provide default history time for all unqualified atomic formulas in the query. The `BELIEVED` clause identifies the belief times which are of interest to the user. When it is not present, the belief time is assumed to be `Now` (i.e., the currently held beliefs are queried).

2.5 Updating the Knowledge Base

The operations `UNTELL` and `RETELL` allow one to update the system's beliefs about certain historical relationships. As indicated already, updated information is not explicitly deleted from the knowledge base. Instead, the belief time intervals associated with such updated information are terminated.

The `UNTELL` operation can be used to specify that some of the instantiation, specialization, or attribute relationships of a proposition no longer hold. Suppose, for example, that Stanley changed his mind about authoring the Martian system on November 20th, 1986 and no longer wants his name associated with the paper. The following operation effects this update:

```
UNTELL martian  WITH
     author
        : Stanley ( at 1986/11/20..? )
```

[5] Since no belief time is given for this integrity constraint, it is assumed that it constrains every belief state of the system from the time it was processed and on.

As a consequence, if this operation was processed on December 9, 1989, its effect is to make the system believe as of December 9, 1989 that Stanley ceased being an author of `martian` on November 20, 1986 (though it remembers him as an author in earlier belief states). Now the answers to the queries

$$\texttt{ASK x/Author}: \ \texttt{x} \in \texttt{martian.author TRUE at 1986/11/19 BELIEVED at 1989/12/10}$$

$$\texttt{ASK x/Author}: \ \texttt{x} \in \texttt{martian.author TRUE at 1986/11/21 BELIEVED at 1989/12/10}$$

will be $\{\texttt{Stanley}\}$ and the empty set respectively.

The `RETELL` operations amounts to a database update and can be semantically treated as the composition of an `UNTELL` and a `TELL` operation. Through `RETELL`, the user can specify, for instance, that somebody's address changed.

3 Model Extensions Through Metaclasses

Compared to other semantic models or knowledge representation languages, Telos appears to provide few features for capturing the semantics of applications. This section is intended to illustrate that additional structure can be introduced for particular categories of propositions through the mechanisms already offered by Telos. The first-class status of attributes and the ability to define attribute classes and metaclasses plays a particularly important role. Our point is that a relatively sparse framework can be used to accomplish a great deal.

Let us consider again the IFIP Conference example. Conference organization involves many different kinds of documents, including various classes of papers, letters, announcements, memos and the like. To define common properties that various document classes have, we may want to introduce attribute metaclasses which support grouping of document attributes according to their semantic, deductive or other properties. One way to introduce these attribute metaclasses is through the metaclass `DocumentClass`:

```
TELL CLASS DocumentClass  IN MetaClass  WITH
      attribute
        source: AgentClass;
        content: SimpleClass;
        destination: AgentClass;
        typicalTurnaroundTime: TimePeriod IN SimpleClass
END
```

In this example, `source, content` and `destination` are labels of attribute metaclasses which may be instantiated for `DocumentClass` instances[6]. In this case, their effect is to group together semantically similar attributes, such as the attributes that describe the content of a document (e.g., title, abstract, keywords, text and so on) or the destination of a document (zero or more recipients, location/affiliation of the destination(s) and so on). The attribute label `typicalTurnaroundTime` specifies a typical value for the length of time it takes for a document to be prepared and sent to its destination (say, one day for a letter, one month for a paper). Note that instances of `typicalTurnaroundTime` are tokens and represent specific facts about their generic sources. Other `DocumentClass` attributes, on the other hand, indicate the kinds of (attribute) classes that may be associated with a document class. `DocumentClass` may also have arbitrary constraints expressed through assertions on its instances – document classes, such as `Paper` or `Letter`. Thus, metaclasses constitute an important facility in the definition of generic objects, one that cannot be simulated by the use of the generalization hierarchy.[7]

The definition of the class `Paper` might now be refined as

```
TELL CLASS Paper IN DocumentClass WITH
      source
        author: Person
      content
        title: String
      ...
      turnaroundTime
        : 4weeks
END
```

Here, the attribute (class) `[Paper, author, Person]` is an instance of `[DocumentClass, source, SimpleClass]`, in a similar way that `[martian, firstAuthor, Stanley]` is an instance of `[Paper, author, Person]`. Thus attribute metaclasses can be thought of as *categories* for generic attributes associated with a class.

As another example of use of attribute metaclasses, consider constraints on attribute values which are built-in in several semantic data models. One such constraint, let us call it the *Single* constraint, restricts an attribute to (at most) a single value. To define it, we introduce the attribute metaclass `Single` whose instances are singleton attribute classes[8].

```
TELL CLASS Single
    COMPONENTS [Class, single, Class]
    IN AttributeClass, MetaClass WITH
        integrityConstraint
            :$ (∀u/Single)(∀p,q/Proposition)
               (p in u ∧ q in u ∧ from(p) = from(q) ∧
                when(p) overlaps when(q) ⟹ p = q) $
END
```

[6] To help the reader, we use identifiers ending with "-Class" for metaclasses.

[7] The grouping of conceptually related attributes illustrated above could also have been accomplished by the introduction of generalization hierarchies for attributes themselves, as in KL-ONE: `title, abstract`, etc. are more specialized roles than `content`. However, as we shall soon see, attribute classes provide additional facilities — such as abbreviating constraints — that cannot be achieved by attribute hierarchies.

[8] The clause `COMPONENTS` gives the source, label and destination of this attribute class.

Figure 3: The constraint `Single` on attribute `title`

Literally, the above assertion states that for every instance of `Single`, say `u` (that is an attribute class), there are no two distinct instances `p` and `q` (these are attribute instances) with common source and overlapping times. Once this class has been defined, it can be exploited, through instantiation, to constrain the attributes of any other class, such as `Paper`:

```
single
  title: String
```

If now `martian` is an instance of `Paper`, with a title specified by

```
title
    :'The Martian system'
```

then the constraint in `Single` refers to attribute propositions like

```
[martian, ..., 'The MARTIAN system']
```

which are instances of `[Paper, title, String]`, and decrees that there cannot be two such propositions with identical sources. Figure 3 illustrates the situation.

Similarly, one can define an attribute metaclass `Necessary` whose integrity constraint assures that each instance of some class for which the attribute is necessary, has at least one value for that attribute [40].

3.1 Talking about Assertions

To increase the extensibility of the language, we provide a way to talk about assertions. The resulting technique allows meta-level reasoning and is very powerful; we have however exploited this power only in defining attribute metaclasses.

We introduce the predicate `Holds` which is true whenever its argument is an assertion that follows from the KB. Recall that assertions appear in the KB, quoted, as parts of propositions:

```
[Paper, ..., "(∀x/Paper)(∀y/Person)...", Alltime].
```

In the definition of metaclasses, it is often useful to have constraints and rules which refer to formulas of the assertion language. Let us assume, for example, that we want to define an attribute metaclass `PreCondition` and use it for specifying preconditions for certain activities.

```
TELL CLASS PreCondition
    COMPONENTS [Class, preCondition, Assertion]
    IN AttributeClass, MetaClass WITH
        integrityConstraint
            :$ (∀p/PreCondition)(∀Obj/Proposition)(∀t/Time)
               (Obj in from(p)[ at t] ⟹ (∃HTime/Time)(HTime overlaps t) ∧
               Holds(to(p)))) $
END
```

This attribute class constrains its instances to have destination components that are *open assertions* which follow from the knowledge base when their special variable `Obj` is bound to an instance of the class they are associated with and `HTime` is bound to an interval which overlaps the lifetime of this instantiation[9].

Open assertions can have at most two free variables, `Obj` and `HTime`, intended to be bound respectively to a proposition (the subject of the assertion) and a history time[10]. Suppose then that the user specifies:

```
TELL CLASS Conference IN ConfEntityClass WITH
        attribute
            budget:  Money
        preCondition
            :$ Obj.budget ≥ 10000 [at HTime] $
END
```

This operation defines the class `Conference` and asserts the proposition

[`Conference`, ..., "`Obj.budget` ≥ `10000` [`at HTime`]", ...]

in the knowledge base. This proposition becomes an instance of `Precondition`. Now the above integrity constraint will be satisfied if every time an instance of `Conference` is constructed, its `budget` value exceeds $10,000. Analogous definitions can be given for other attribute metaclasses such as `PostCondition`, `ActivationCondition` and `Invariant` with obvious semantics [40].

In general, our language enables us to define metaclasses which represent concepts that are appropriate for a particular application domain. For instance, if it is deemed that the concept of *activity* is useful for modeling conferences, we may want to adopt SADT's[11] notions of *input, output* and *control*, referring to entities that are consumed, produced or used without state change by an activity:

```
TELL CLASS ActivityClass IN MetaClass WITH
        attributes
            agent:  AgentClass;
            input:  EntityClass;
            output:  EntityClass;
            control:  EntityClass;
            part:  ActivityClass
        integrityConstraint
            inputExists:$ (∀p/Proposition)(∀x/Token)(∀t1/Time)
                    (p in this|input ∧ x in from(p)[at t1] ⟹
                    (∃q/Attribute)(∃t2/Time)(q in p ∧
                    to(q) in to(p)[at t2] ∧ t2 overlaps t1)) $;
            outputCreated:  ...
            controlRemains:  ...
            partDuringWhole:$ (∀p/Proposition)(∀x/Token)(∀t1/Time)
                    (p in this|part ∧ x in from(p)[at t1] ⟹ (∃q/Attribute)
```

[9]Note that `t overlaps t'` implies that `t startsbefore t'`, in addition to the implication that the two intervals have a common subinterval.

[10]This binding is achieved through quantification in the constraint or rule which refers to the open assertion.

[11]SADT is the trademark of SofTech Inc.

```
                    (∃t2/Time)(q in p[at t2] ∧ from(q) = x ∧ t2 during t1)) $;
            partInputConsistency: ...
            partOutputConsistency: ...
    END
```

The `inputExists` constraint checks that inputs exist at the start of an activity and cease to be instances of their respective input types before the end of an activity. Likewise, the `partDuringWhole` constraint declares that components of an activity occur during the activity. Note that RML too, [25], offers the notion of activity defined above. However, RML has these notions *built-in* and is therefore less adaptable to applications where these notions need slight or major changes. Suppose that one wishes to base requirements modeling on the notion of *role* rather than that of activity, following [43]. For RML such a change of perspective amounts to a total discard. For Telos it simply means that a different set of individual and attribute metaclasses needs to be defined. Specific activities can now be modeled in the intended application domain.

A final example of a useful attribute metaclass is `Rep`. It can be used to constrain two classes to have isomorphic extensions.

```
TELL CLASS Rep
    COMPONENTS [Class, rep, Class]
    IN AttributeClass, MetaClass WITH
        integrityConstraint
            :$ (∀x/Proposition)(x in from(this) ⟹
               (∃!p/Proposition)(p in this ∧ x = from(p)))
               ∧ (∀y/Proposition)(y in to(this) ⟹
               (∃!p/Proposition(p in this ∧ y = to(p))) $
END
```

In the above definition "∃!" stands for "there exists unique". Thus if

```
[ConferenceOrganization, ..., ConfRecord, ...]
```

is an instance of `Rep`, there is a one-to-one correspondence between the instances of the two classes (though corresponding instances need not have identical time intervals). `Rep` is useful for expressing accuracy and completeness requirements on the contents of an information system, as we will see in the following section.

4 Representing Knowledge About Information Systems

After obtaining a basic understanding of Telos, we can now look at its application in developing knowledge bases about software. In particular, we present examples of using Telos in describing and then applying a rather powerful metamodel for knowledge relevant to the development of information systems.

Traditionally, database engineering has made the (tacit) assumption that an information system is supposed to capture some excerpt of world history, and hence has concentrated on modeling (i.e., "capturing information about") the application domain. This practice provides an answer of sorts to the fundamental question "What does the information handled by my system mean?". Unfortunately, it also tends to draw attention away from a number of equally fundamental questions, concerning other types of knowledge about an information system. The section begins with a basic taxonomy of distinct "subworlds" about which knowledge needs to be recorded during the development of an information system. The distinction between the different subworlds is illustrated with definitions drawn from the IFIP Conference example [42].

- The *subject world* is the domain about which the proposed information system is to provide information; this world may be an organizational environment, say a department store, or something completely different, say a world of chemical experiments or geopolitical games. Consequently, the set of appropriate concepts for representing this world may vary considerably. For the purposes of our running example, the proposed information system does maintain information about an organizational setting. The notions of *activity* and *entity* will be assumed to be appropriate for modeling such a world.

- The *system world* includes specifications at different levels of implementation detail on what the information system does. The nature and the number of specification levels depends on the development methodology adopted. For instance, the levels may include functional requirements, conceptual designs and implementations. For each level, appropriate concepts need to be defined and made part of the system world metamodel. To keep our running example (somewhat) manageable, we limit the discussion of system world modeling to functional requirements only. As part of the system world, one may want to prescribe correspondences between the information maintained by the information system and the intended subject world. It might be specified, for instance, that the system's record of conferences is complete (for every conference, there is an entry in the system's records) and accurate (there are no entries in the system's records that do not correspond to an actual conference) through the use of attribute metaclasses such as `Rep`.
- The *usage world* describes the environment within which the system is embedded. Such descriptions often take the form of input/output relationships, but may also include the different classes of (end) users of the intended system or the kinds of interfaces supported by the system (represented, to a first approximation, as activities having the system and one or more users as co-agents). The usage world may also include descriptions of the organizational environment within which the system will function, including office procedures. Advanced usage worlds may even include user modeling, e.g., what does each class of users know about the subject and system worlds, how often do they use the system, and the like.
- The *development world* focuses on the entities and activities which arise as part of the design process itself. These would include the composition of the design team, responsibilities of each team member, design decisions, development tools, supporting documentation, etc. For example, there may be a standard procedure by which an existing version of the conference management system is adapted to the needs of a specific conference, using a specific development team hired by IFIP and a specific software development environment.

One of the obligations of a knowledge representation system is to provide guidance on the organization of the knowledge at hand as well as the process to be used by the knowledge engineer to build up his knowledge base. Figure 4 illustrates the basic "worlds" as well as the kinds of knowledge that might relate them. The rest of the section illustrates features of Telos by suggesting possible ontologies for each of the above worlds, and occasionally instantiating it for the IFIP domain. We emphasize that the discussion in this section is intended to illustrate the flexibility of Telos in modeling drastically different worlds, a flexibility derived primarily from its classification dimension and the treatment of attributes. Nevertheless, the examples presented *are*, in fact, based on actual application of Telos, mostly in the context of ESPRIT project DAIDA, where a complete prototype environment for information systems engineering was constructed, generally following the approach outlined here [34].

4.1 The Subject World

An essential aspect of information system development is the characterization of the domain about which information will be maintained – a characterization which needs to be explicitly recorded. A number of different general approaches could be followed in gathering and recording this information, including Structured Analysis, Entity-Relationship-Activity, etc. These approaches can be defined within Telos. We could even extend them with concepts about a particular application domain, e.g., accounting systems, to cover the use of standard requirements. Some real-world experiences with domain modeling have been gained with a commercial implementation of an early Telos version [29].

For an example, we continue with the definition of a subset of the notions of RML [25] begun in section 3. As indicated earlier, RML's basic structures are loosely based on SADT [49]; this was motivated in part by the idea of using SADT as a graphical road map which sketches the requirements model before filling in the more formal semantic details. RML provides the three basic mechanisms of Activities, Entities, and Assertions. Since the latter are already available in the Telos kernel, we need to concern ourselves only with *activities* and *entities*. The definition of `ActivityClass` presented in the previous section can be used as is. The definition of `EntityClass` follows:

Figure 4: Knowledge about Information Systems

```
TELL CLASS EntityClass IN MetaClass WITH
   attribute
      producer, consumer:  ActivityClass;
      part, association:  EntityClass
   integrityConstraint
      producedByProducer :$ (∀x/Proposition)(x in this ⟹
          (∃y/Proposition)(∃p,q/Attribute)(y in this.producer ∧
          from(p) = y ∧ to(p) = x ∧ p in q ∧ q in y.output)) $
END
```

Every instance of an entity class is produced by a producer activity. Note the (pleasing) duality between entities and activities, initially offered in SADT.

Since entities and activities *per se* can be seen as fundamental concepts in describing human activities, and hence will appear in the other subworlds, we will in fact use the generalization hierarchy to place `EntityClass` and `ActivityClass` in `ISWorldClass`, rather than just `SubjectWorldClass`. Returning to conference organizing, one can then start modeling the domain by considering specific classes of entities and activities, such as the following.

```
TELL CLASS SubmittingAPaper IN ActivityClass WITH
   output
     sentIn: PaperSubmitted;
     submissionLetter: Letter
   input
     prepared: PaperWritten
   control
     sender: Person;
     recipient: Person;
     conference: Conference
   integrityConstraint
     samePaper: $ this.prepared = this.sentIn $;
     submissionOnTime: $ when(this) before this.conference.deadline $;
     rightRecipient: $ this.recipient = this.conference.programChair $
END
```

```
TELL CLASS Paper IN EntityClass WITH
   producer
      paperwriting: Writing
   consumer
      submitting: SubmittingAPaper
   part
      title: String;
      pages: 1..12
   association
      author: Author;
      conference: Conference
   deductiveRule
      rightconference :
         $ c ∈ this.submitting.conference ⟹ c ∈ this.conference $
END
```

In the DAIDA project, Telos-based subject world ontologies have been used to build graphical frontends for requirements engineering which give the developer well-known visualizations such as SADT diagrams but have a precise formal background. This formal background can be further exploited to integrate other views of the same information system (e.g., Entity-Relationship).

4.2 The System World

One possible view of an Information System is as a world with its own specific entity and activity classes – often called *data* and *transactions*. An important specific characteristic of information systems that their data and activity classes are often related to objects and activities in the subject world.

Continuing our example, we use a set of system classes that follows the style of Taxis [39]. System activity classes are called transactions and can have only data classes (a special kind of entity classes) as inputs and outputs. In defining the representation relationship, we leave a lot of room for design decisions; for example, data classes can either represent subject world entities (e.g., data about referees and referees themselves), or traces of subject world activities (e.g., a class `AuthorKitMailReceipts` tracking the activity of mailing out forms to authors), or a mixture.

```
TELL CLASS TransactionClass ISA ActivityClass WITH
   attribute
      input, output: DataClass
   rep
      activitybytransaction: ActivityClass
END

TELL CLASS DataClass ISA EntityClass WITH
   attribute
      producer, consumer: TransactionClass
   rep
      activitybydata: ActivityClass;
      entitybydata: EntityClass
END
```

Given this specification context, one can now start defining the model of the data base at the semantic level, by defining a variety of subclasses of `DataClass` and `TransactionClass`. This process is sufficiently familiar that we will not detail it further here.

4.3 The Usage World

The usage world model is intended to describe the man-machine interactions supported by the information system and the context within which they take place. Depending on the kind of usage environment, such models have been investigated in office research, computer-integrated manufacturing and similar fields of study which involve the integration of information technology with its environment.

A natural description of the usage world can also be given in terms of activities and entities. In fact, subject and usage world have not been traditionally distinguished. Although this is not the place to argue the issue in full, we believe that the distinction is evident in certain domains: e.g., contrast the world of agriculture, involving farming, weather, produce, etc., and the information management activities about farming which occur in the Department of Agriculture. The distinction is crucial in clarifying various, potentially conflicting needs that the information system must fulfil.

For the IFIP example, the subject world contains authors, paper submission, referees and the like, whereas the usage model talks about office staff and office tasks such as writing acknowledgments or selecting referees. Additionally, the usage model can exploit the fact that usage activities (both manual ones and man-machine interactions) may be constrained by plans or bureucratic procedures that involve issues such as precedence, priority and security.

In modeling these aspects, we shall make use of the specialization abstraction in Telos. Usage world activity classes are specialized activity classes for which an agent is known (it can be a person, a system, or a team composed of both persons and systems) and which can be subject to certain precedence constraints; also, usage world activities may have to satisfy goals set by their supervisors, and these goals may influence the representation mapping between subject world and system world.

```
TELL CLASS UsageActivityClass ISA ActivityClass WITH
    attribute
       agents:  AgentClass;
       supervisor:  UserClass;
       precedes:  UsageActivityClass;
       goals:  Goals
    integrityConstraint
       precedence:
          $ (∀preclink/Proposition)(preclink in this|precedes ⟹
             when(from(preclink)) before when(to(preclink))) $
END
```

```
TELL CLASS Goals ISA EntityClass END

TELL CLASS AgentClass IN MetaClass WITH
   attribute
     member: AgentClass
END
```

The representational relationships between system world and subject world can be related to the usage world by qualifying these relationships with the goals followed when determining the system requirements. These goals can be both functional (covering the functionality required by the application domain) or non-functional (performance, accuracy, etc.). In Telos, this can be accomplished quite easily by attribution of attributes, one of the unique features of the language:

```
TELL CLASS TransactionClass!repbytrans IN Class!rep WITH
   attribute
      mappingGoal: Goals
END
```

The following is an example of one usage class defined:

```
TELL CLASS WriteAcknowledgment IN UsageActivityClass WITH
    output
       letter:  Letter
    control
       sender:  Person;
       recipient:  Person;
       submission:  SubmittingAPaper
    agents
       letterprogram:  DesignTask1
    supervisor
       programChair:  Person
    precedes
       :  SendToReferees
    integrityConstraint
       chairperson:
          $ this.submission.conference.programChair = this.programChair $;
       rightpersons:
          $ this.sender = this.programChair ∧ this.recipient = this.submission.sender$;
       acknowledgmentWithinAWeek:
          $ when(this) before when(this.submission) + 7 $
END
```

We do not detail the modeling of agents at this point. A more elaborate metamodel could incorporate some basic cognitive constraints on agents which guide the activities they participate in. It could also describe the organizational structure of agents, their access rights, etc. Telos users have actually developed several such models, e.g., for purposes of security specification, for a co-authoring system, for contract negotiation support in public construction projects, even for modeling paradigm shifts in the history of natural science.

4.4 The Development World

The development world sees the information system as a design object to be worked on. Typically, the management of software development is organized in layers. Single-worker tasks involving detailed knowledge about individual languages, methodologies, and tools are called programming-in-the-small. Tasks involving the negotiation and coordination of multiple programmers are called programming-in-the-many. In between, object management tasks such as version and configuration management are the domain of programming-in-the-large [48]. Here, we only sketch an in-the-small and an in-the-many model.

In a Telos model used intensively throughout DAIDA [16] [33] programming-in-the-small is understood as a set of interrelated design decisions which transform design objects into other design objects, supported by design tools. Design tools are modeled by the special kinds of design decisions they support. Design goals are a special kind of design objects which provide a rationale for design decisions.

```
TELL CLASS DesignObject ISA EntityClass END

TELL CLASS DesignGoal ISA DesignObject, Goals END

TELL CLASS DesignDecision ISA UsageActivityClass WITH
   attribute
     input: DesignObject;
```

```
      output: DesignObject;
      agents: DesignTool
      goals: DesignGoal;
END

TELL CLASS DesignTool ISA AgentClass WITH
   attribute
      qualification: DesignDecision
END
```

For the IFIP example, a design decision could concern the choice of a particular standard package for a task defined in the system model. This could be led by the goal of saving development costs, and could have been proposed by an outside consulting firm.

More generally, design decisions may involve the refinement of existing models, the mappings between various representational formalisms, the versioning of existing design objects, and the configuration of complex systems from reusable components. In the DAIDA project, this model has been used to formalize and manage the integration of multiple languages, methods, and tools of the DAIDA environment [33]. The same model has also been used to represent a bootstrapping process by which an implementation of the full Telos language was derived from a small kernel [32].

To organize programming-in-the-many, we start from the concept of Agent introduced in the Usage World. To this concept, we simply add a possibility that these agents can communicate about design decisions according to some protocol.

```
TELL CLASS ConversationClass ISA UsageActivityClass WITH
   attribute
      content: DesignDecision
   integrityConstraint
        :$ size(this.agents) >= 2 $
END
```

The constraint says that any conversation model should foresee at least two kinds of roles for agents; it can be satisfied by any message protocol that knows of senders and receivers, for example.

One possible instance of such a metamodel is Winograd's [56] conversation-for-action protocol. The model below specializes the topics of conversation to be formal tasks in developing software according to our in-the-small methodology. Note also the usage of multiple inheritance.

```
TELL CLASS Message IN ConversationClass WITH
   agents
      sender: ProjectMember;
      recipient: ProjectTeam
END

TELL CLASS Response ISA Message WITH
    attribute, necessary
       reference:  Message
    integrityConstraint
       personsright:$ this.reference.sender = this.recipient ∧
                      this.reference.recipient = this.sender $
END

TELL CLASS MappingRequest ISA Message WITH
   content
      mappingtask: MappingDecision
END
```

Figure 5: A contracting conversation between a designer and a programmer

```
TELL CLASS MappingPromise ISA Response WITH
   attribute, necessary
      reference: MappingRequest
   content, necessary
      mappingtask: MappingDecision
   integrityConstraint
      taskright:
       $ this.reference.mappingtask = this.mappingtask $
END
```

This excerpt covers the first part of a conversation for action. The constraints say that, if one partner promises a request put to him, this partner has (counter it or) to accept it exactly as is. Conversation primitives such as Counter and Accept can be modeled in a similar fashion. In the Telos-based software information system, ConceptBase [21], the full model has been implemented. The implementation has been applied to several software development examples, including management of a large programming class. Figure 5 is a screendump from such a session; the graphical browser documents the status of a subcontracting conversation between a designer and a programmer concerning the redesign of a program.

The design of a sophisticated Telos-specified project management system which also takes into account the proactive organization and reorganization of ill-structured projects is reported in [51].

ConceptBase also offers a graphical "argument editor" which supports the real-time or asynchronous discussion within a distributed software development team [28]. This editor is based on another instance of ConversationClass inspired by Toulmin's model [54], and is similar to the gIBIS tool developed by MCC [17], with the important difference that the topic of discussion is formally known to the system. Initial application experiences indicate that such an editor may be a valuable tool for recording the rationales of design decisions; formal experiments to evaluate this claim are being prepared.

Note how the ConceptBase system associates different shapes and colors with the metaclasses defined above. In Figure 5, the agents are shown in white ovals, and the aggregation of messages in a conversation is made explicit. The flexibility of user-defined metaclasses in Telos requires tools for interactively defining such mappings between knowledge base objects and their graphical representations. Formally, this mapping can be described by Telos deductive rules [32].

4.5 Discussion

In this section, we have proposed a multi-perspective approach to the representation of knowledge about information systems. For space reasons, we could only briefly reference actual experiences with using Telos for each of these perspectives and for their integration, gained by our own research groups as well as by industrial partners and various outside users and students. These experiences lead us to believe that a number of Telos features make it especially suitable for the task of representing knowledge about information system development.

First, the by–now–standard approach of modeling a domain using objects related by attributes, grouped into classes that are organized into subclass hierarchies, provides the usual benefits of organizing and abstracting information. The possibility of viewing classes as objects, themselves grouped into (meta)classes, allows the description itself to be structured in useful ways, as in the case of the different subworlds.

The treatment of attributes as first-class objects has also proven very useful for two reasons. It does not force us to commit ourselves at the point of first definition of an object how it is going to be used since, e.g., new attributes can be associated with any Telos proposition. Further, attributes of attributes can be the most compact and natural representation for structures such as design dependencies (e.g., the dependency `TransactionClass!repbytrans` in the UsageWorld) which have to be represented much more awkwardly in other languages. We have included in the presentation a screendump to illustrate how the structural orientation of Telos, supported by the uniform treatment of individuals and attributes, allows a graphical representation of the knowledge base which can be usefully exploited in providing a nice user interface for developers. In particular, hypertext-like interfaces are natural for semantic network-style representation schemes [7], and allow developers to “navigate” through the knowledge base to explore its contents.

Finally, the explicit distinction between and availability of both domain-time and development-time in Telos allows not just historical reference in the subject world, but the ability to rationalize chronologically the evolution of the software design – an important capability during software maintenance, which is the most cost- and work-intensive part of the software life-cycle.

In summary, our experiments as well as initial experiences by other users have shown that Telos elegantly covers many of the requirements for representing knowledge about information systems.

œ[A

5 Some Formal Aspects of the Telos System

A formal account of a new language concisely expresses the meaning of the various language constructs in a thorough and organized manner. Indeed, with Telos, as with many other languages, ambiguities and inconsistencies were discovered during the process of constructing a formal account of the language. In addition, such an account can serve as point of contact between the language designers, implementors, end-users and critics.

Following [13], we describe a Telos knowledge base in terms of its functional behaviour at the knowledge level for operations such as $TELL$ and ASK. A Telos knowledge base is described in part (though not implemented) as a collection of (historical) first-order theories, indexed by belief times[12]. Intuitively, each one of these theories corresponds to the beliefs of the system during some time period (the index of the theory). Knowledge base operations are then treated as functions defined over indexed theories, and other things. For details omitted in this section, see [37] and [40].

Each historical theory is constructed in the following way: Start with a first-order logic with types augmented with axioms for *isa*, *in*, time entities, etc; each historical theory constructed will be a theory in this logic. All such theories include a group of axioms which in some sense “define” what is a well-formed

[12] Since belief times are constant, they can be represented by semi-open intervals of the form $[a, b)$ or $[a, +\infty)$, where a and b are atomic point-like intervals – days in the discussion of the previous sections.

knowledge base; in addition, every $TELL$ operation introduces additional axioms. Finally, ASK operations invoke certain default assumptions to provide answers to queries.

The formal account of this section offers a proof theory as definition of "consistency" and "question-answering". By translating Telos expressions into a standard first order logic, we get a model theory as a bonus.

5.1 Initial Theory

5.1.1 The Target Language $\mathcal{L}$

The language we use, $\mathcal{L}$, is a first order logic with equality, with the additional restriction that all quantifiers have restricted ranges. The basic types/ranges of $\mathcal{L}$ are *Proposition*, *Time* and *Class*. For any particular knowledge base, all class names appearing in it can also be used as ranges for quantifiers. $\mathcal{L}$ also contains the following predicates and function symbols:

- The 5-place predicate symbol *prop* used to describe the components of every Telos proposition formally. The first four arguments of *prop* are of type *Proposition*; the last argument is of type *Time*.
- The 1-place function symbols $from, label, to$ and $when$ used to map propositions to their components.
- The 3-place predicate symbols *in* and *isa* used to describe instances and subclasses respectively. The type of *in* is $Proposition \times Class \times Time$; the type of *isa* is $Class \times Class \times Time$.

5.1.2 Axioms

Time is axiomatized using Allen and Hayes' proposal [4], modified to reflect the Telos conventions. Other axioms in our theory include: *Axiom for proposition components:*

$$(\forall p, x, y, z/Proposition)(\forall t/Time)(prop(p, x, y, z, t) \implies \\ from(p) = x \ \wedge \ label(p) = y \ \wedge \ to(p) = z \ \wedge \ when(p) = t)$$

Transitivity of IsA[13]:

$$(\forall p_1, p_2, p_3/Class)(\forall t_1, t_2, t_3/Time) \ (isa(p_1, p_2, t_1) \ \wedge \ isa(p_2, p_3, t_2) \ \wedge \\ t_3 = t_1 * t_2 \implies isa(p_1, p_3, t_3))$$

Specialization Postulate: The extension of a class is a superset of the extension of any of its subclasses.

$$(\forall p_1, p_2, p_3/Proposition)(\forall t_1, t_2, t_3/Time) \ (in(p_1, p_2, t_1) \ \wedge \ isa(p_2, p_3, t_2) \ \wedge \\ t_3 = t_1 * t_2 \implies in(p_1, p_3, t_3))$$

The Instantiation Constraint: if proposition p_1 is an instance of proposition p_2 then $from(p_1)$ must be an instance of $from(p_2)$, $to(p_1)$ must be an instance of $to(p_2)$ and $when(p_1)$ must overlap $when(p_2)$.

$$(\forall p_1, p_2/Proposition)(\forall t_1/Time)(instanceOf(p_1, p_2, t_1) \implies \\ (\exists t_2, t_3)(in(from(p_1), from(p_2), t_2) \ \wedge \ in(to(p_1), to(p_2), t_3) \ \wedge \\ during(t_1, t_2) \ \wedge \ during(t_1, t_3) \ \wedge \ overlaps(when(p_1), when(p_2))))$$

5.2 Integrity Constraints and Deductive Rules

Integrity constraints and deductive rules are mapped into closed statements in the language $\mathcal{L}$. This mapping provides the semantics for the terms of the assertion language in Telos. As an example, the following is the $\mathcal{L}$ statement corresponding to the deductive rule from section 2.3

$$(\forall t/Time)(\forall x, y, z, w/Proposition) \\ (in(x, Paper, t) \ \wedge \ in(y, Address, t) \ \wedge \ author(x, w, t) \ \wedge \\ address(w, z, t) \implies replyAddress(x, z, t))$$

[13]The intersection of two time intervals t_1 and t_2 (i.e., the time period common to both) is denoted by $\mathtt{t_1 * t_2}$.

where, in turn, a predicate such as $author(x, y, t)$ corresponds to

$$(\exists p, l, t'/Proposition)(in(p, author, t') \ \wedge \ prop(p, x, l, y, t)).$$

To make their processing tractable, rules are restricted to a "Horn" form: < **antecedent** > $\Longrightarrow$ < **consequent** >. The antecedent must be a conjunction of atomic formulas while the consequent must be a single positive atomic formula. In addition, all the variables in the rule must be universally quantified at the beginning of the formula.

Deductive rules are statements which are added to the appropriate historical theories and are used by ASK for answering queries with respect to these theories. However, integrity constraints are statements which must be *satisfied*: an integrity constraint is satisfied if it is consistent with the completion of the corresponding historical theory[14]. Otherwise, we say that the constraint is *violated*.

5.3 A Functional Specification of the Telos System

A Telos knowledge base is characterized in terms of two sets KB and IC. These sets are sequences of (historical) theories indexed by belief time intervals. As we mentioned earlier, each theory in KB corresponds to the beliefs of the system during the time period which is the index of the theory. The time periods are demarcated by the system clock time, and therefore form a contiguous sequence up to the present time, Now. These beliefs are constrained by a theory in IC (with identical index). Knowledge bases are modified and queried by the functions $TELL, UNTELL, RETELL$ and ASK. Only $TELL$ and ASK are presented here. The $TELL$ operation has the functionality:

$$TELL: \ KB \ \times \ IC \ \times \ O \ \times \ Time \longrightarrow \ KB \ \times \ IC$$

In addition to the objects being defined (O), $TELL$ looks at the last theory in the input knowledge base, say KB_n (where n has the form $[t, +\infty)$), the corresponding set of integrity constraints IC_n, the current system time, and produces a new knowledge base where the index of KB_n and IC_n has been changed to $[t, s)$, and a new theory $KB_{[s,+\infty)}$ and an enhanced set of integrity constraints $IC_{[s,+\infty)}$ have been added. The new theories are produced by unioning KB_n (resp. IC_n) with atomic formulas and deductive rules (resp. integrity constraints) corresponding to the definitions in O.

The $TELL$ operation takes effect only if the theories in $KB_{[s,+\infty)}$ and $IC_{[s,+\infty)}$ are consistent and if all the integrity constraints in $IC_{[s,+\infty)}$ are satisfied. Note that because we are in first order logic, the definition of classes resembling Russel's paradox ("all classes not members of themselves") simply leads to an inconsistent knowledge base. Users of Telos are therefore urged to abide by the convention that only propositions at adjacent levels of the instantiation hierarchy should be related by the predicate in.

ASK can be functionally understood as follows:

$$ASK: \ KB \ \times \ Query \ \times \ Time \longrightarrow \ Answers$$

ASK operates on the historical theories in KB whose index overlaps the time period stated in the `BELIEVED` clause of the query. Assume that there is exactly one such historical theory[15]. We first need to know the *completion* of that historical theory. The completion of a given theory is computed using the following assumptions, inspired by [45]:

1. A "domain closure assumption", which states that the individuals known are all the ones that exist, excluding time constants:

   $$(\forall x/Proposition)\neg Time(x) \implies (x = c_1 \ \vee \ x = c_2 \ \vee \ \ldots \ \vee \ x = c_n).$$

   A separate domain comprehension axiom is given for time intervals.

2. The equality axioms (reflexivity, commutativity, associativity and Leibnitz's principle of substitution of equal terms).

[14] The completion of a knowledge base KB, denoted by $\overline{KB}$, is defined in the next section.

[15] In the more general case, intersect the answers to the subqueries to each historical theory.

3. The "unique names assumption", asserting that distinct constants are not equal. Note that this is not asserted for temporal terms other than the ones representing standard intervals, because in general two time intervals a and b are assumed to be equal if and only if $equals(a, b)$ is derivable.

4. The "completed theory assumption": to obtain negative information, we want to assume that any facts about the base predicates $isa, in, prop$ that cannot be derived from told facts or deductive rules are false. For this purpose we use the restricted form of the deductive rules to adopt a "predicate completion" technique such as considered in [45].

Now if $query$ is a closed historical query, i.e., it is of the form $< |W >$ (where W is a first order statement) then

$$ASK(KB_n, query) = \begin{cases} yes & \text{if } \overline{KB_n} \models W \\ no & \text{if } \overline{KB_n} \models \neg W \\ unknown & \text{if } \overline{KB_n} \not\models W \\ & \text{ and } \overline{KB_n} \not\models \neg W \end{cases}$$

If $query$ is an open historical query i.e., it is of the form $< x_1/\tau_1, \cdots, x_n/\tau_n \mid W >$ (where W is a first order formula whose only free variables are $x_1, \ldots, x_n$) then

$$ASK(KB_n, query) = \begin{cases} \theta & \text{if for every substitution } \theta_i \in \theta,\ \overline{KB_n} \models W\theta_i \\ \emptyset & \text{if there is no substitution } \theta \text{ such that } \overline{KB_n} \models W\theta \end{cases}$$

5.4 Formalizing the Holds Relation for $\mathcal{L}$

We provide here a brief formal account of the `Holds` predicate introduced in section 3.1.[16] The predicate was introduced to allow meta-atributes to impose additional constraints on the free variables `Obj` and `HTime`, which occur in formulas that will act as preconditions, activation conditions, etc. These formulas must therefore be objects – they will appear as parts of propositions after all – and some form of "quotation" mechanism is needed for this purpose. However, once values from the actual domain of discourse (propositions, times) are substituted for `Obj` and `HTime`, we want the formulas to be "unquoted" and verified.

For this purpose we encode formulas in the base language $\mathcal{L}$ as abstract syntax trees (Prolog terms or Lisp lists in prefix notation), assuming that for every logical and non-logical symbol of $\mathcal{L}$ we have a corresponding constructor. Thus $(\forall x)(P(x) \wedge Q(x))$ might be encoded as

$$ALL(X, AND(PRED(P, X), PRED(Q, X))).$$

If the former is called formula ϕ, then the latter is named $\ulcorner\phi\urcorner$.

The result is a language $\mathcal{L}'$, in which quoted sentences from $\mathcal{L}$ are also objects. We can give these sentences assertional power by setting up a simple predicate $True$, which represents the standard model theoretic notion of truth (e.g., $True(AND(A, B))$ iff $True(A)$ and $True(B)$, while $True(PRED(P, X))$ iff $P(x)$.). Alternatively, we could define a predicate $Provable$, which fomalizes some proof theory of the logic of $\mathcal{L}$ (as in [10]). Note that the special variables `Obj` and `HTime` will not be quoted inside a formula, so that quantifiers outside `Holds` can keep them in their scope.

As shown in [18], one avoids the paradoxes usually associated with this sort of encoding, by the simple expedient of not allowing *all* formulas from $\mathcal{L}'$ to be encoded in $\mathcal{L}'$, but only those in $\mathcal{L}$. We will want users to write formulas involving `Holds`, as illustrated by the case of `PreCondition`. Such formulas are however not needed as arguments to $True$, so we will simply disallow them in that position, e.g., by defining

$$Holds(\ulcorner\phi\urcorner) \iff NoNesting(\ulcorner\phi\urcorner) \wedge True(\ulcorner\phi\urcorner)$$

Finally, for this construction to work properly it is necessary to have a canonical name (primitive or constructed) for every value in the domain of $\mathcal{L}$ – every value over which one can quantify. This is not an issue for propositions, of which there is only a finite number. But time intervals have been axiomatized according to [3], so we need some naming scheme for a countable space of "canonical intervals". This can be accomplished, for example, by naming intervals with pairs of integers. The actual intervals which appear

[16] We are indebted to Jim des Rivieres for clarifications.

in propositions are then existentially quantified, and can be treated proof-theoretically according to the technique of [45]: as constants for which the "unique name axiom" does not hold. This allows us to learn more information about them, including the fact that some intervals are equal.

6 Related Work

Telos is fundamentally a knowledge representation language, albeit one that has been specially crafted to facilitate the description of concepts related to the development of information systems. Thus, it is reasonable to look for related work in three general areas: Artificial Intelligence, Databases and Software Engineering. In each case we will concentrate on closely related approaches and influences rather than attempt to survey entire subareas.

Telos is a language in the tradition of semantic networks. Its distinguishing marks over other proposals is the treatment of attributes as first-class objects, including the device of attribute instantiation, the *integrated* representation of temporal knowledge, and the parsimonious foundation of the language resting on the single notion of "*proposition*".

The treatment of attributes was influenced by [38] where "slots" are distinguished from "metaslots", analogously to attribute classes and metaclasses. Telos' treatment of attributes can be considered a generalization of this earlier effort. Our choice of a temporal model based on intervals, rather than time points has been influenced by [2] and [55]. Of course, the nature of the temporal component of Telos was considerably complicated by all other features that need to coexist within one linguistic and conceptual framework: notationally (making sure that time does not get in the way as the user `ASK`s and `TELL`s), semantically (the meaning of history and belief times), and structurally (offering time as a fourth dimension of knowledge organization in Telos) and, finally, from an implementation viewpoint. To our knowledge, no full-fledged knowledge representation language provides facilities for time that are as tightly integrated into the overall representational framework.

Finally, the integration of structural/organizational aspects of knowledge representation with assertional/deductive parts, has been advocated and implemented in various forms in precursor systems such as Omega [5], Cake [46] and Krypton [11]. These, and other so-called "hybrid" systems, combine one or more special-purpose but efficient reasoners with a general deductive mechanism in order to achieve better performance. For example, Krypton's terminologic component provides efficient reasoning about concept definitions, which are then used by the assertional component for theorem proving. Telos does not explicitly demarcate subsystems of different kinds; in fact, Telos uses the object-centered nature of the representation mechanism to structure the entire knowledge base, by making individuals and concepts be syntactic anchors for both generic and specific facts, including formulas in the logical assertion language. On the other hand, reasoning with time has been treated in a special way both in the language semantics and implementation, and the distinction between constraints/rules and ask/retrieve point to the existence of distinct ways of using what would otherwise be first-order formulas.

Turning to Databases, the modeling of the application world has been the focus of much work on semantic data models [31]. Telos continues in the spirit of this work by emphasizing the importance of structuring mechanisms, including generalization, aggregation and classification, which were first identified in the context of data modeling. In fact, Telos goes further by applying these ideas uniformly to attributes as well as individuals. The equal treatment of individuals and relationships dates back to the Entity-Relationship model [15], but there are many differences between Telos and ER languages, including the presence of higher order ("meta") classes, and the absence of n-ary relationships with built-in cardinality restrictions. The addition of a temporal dimension is a further step in the evolution of conceptual models. The distinction between the "clock" of the application world and that of the system administrators has also been made in the work on temporal databases [50].

Object-oriented databases [20] are a recent development, which merge the semantic modeling constructs with the notion of objects having an encapsulated internal state. They are particularly useful for CAD applications, including providing database support for Software Engineering. For example, the Cactis system [30] concentrates on the efficient management of derived data — objects can have local attributes defined by functional formulas, and, interestingly, separates the definition of binary relationship classes from that of the object classes being related. This is used to allow increased flexibility in making changes in the schema, and resembles Telos' view of class/individual definitions as being nothing but convenient collections of propositions, rather than monolithic wholes. The Telos implementation has not paid the same careful attention to efficiency of updates, but on the other hand it does support temporal reasoning, a full instance-of hierarchy, and allows attributes as first class citizens.

Interestingly enough, the database logic F-logic [35] also treats individuals and attributes in a uniform way. Like Telos, it can express meta-queries such as "retrieve the set of all objects which represent the labels defined for a certain object". However, it does not support time and does not offer an instantiation dimension (instead, it treats classes and instances as members of the same lattice ordered by the "definedness" ordering of denotational semantics). On the other hand, F-logic offers a more explicit notion of complex objects and a nice way of dealing with inconsistencies.

The inferential aspects of Telos, especially the distinction between integrity rules and deductive rules had been made by researchers in the field of Deductive Databases [23]. However, Telos' assertion language is different from the FOL-based languages usually offered in deductive databases, and the propositional 4-tuple foundations of Telos are fully novel. The functional constructs in our assertion language are reminiscent of similar constructs in the language COL [1] if the temporal arguments are omitted.

Requirements modeling in Software Engineering is a third research area that has influenced the development of Telos. Following the pioneering work of [6], [14] and [25], a consensus seems to have emerged that a requirements model should include a description of the application domain in addition to a functional specification of the system itself, both expressed in an "object-centered" framework. We have already discussed in this paper the close relationship between RML [25] and Telos, while [9] discusses the relationship between RML and the work of [6] and [14].

In addition to RML, the ERAE method [26] also considers a formal embedding of the temporal dimension into the requirements model, albeit using a temporal logic approach. Both ERAE and PLEXSYS [36] consider the embedding of the system in its "usage world" but neither permits the qualification of the "rep(resents)" relationship which is possible in Telos.

More generally, the work on *domain modeling* for software reuse that began with [41] has recently produced a number of languages and systems which are intended to capture a wide range of information about software subdomains. A number of these systems, including the Lassie software information system [19], and the Desire/Rose design recovery/reuse system [7] are built around languages which are explicitly based on AI knowledge representation techniques, such as frame systems and connectionist networks.

There is an entire subfield of formal software specifications, and some approaches are in fact related to Knowledge Representation schemes (e.g., the Plan Calculus [46]). However, these schemes are usually oriented towards the description of software components, and do not attempt to address world modeling, nor the representation of software development knowledge.

7 Summary

The development, use and maintenance of information systems involves a great deal of knowledge. We have classified this knowledge as concerning (at least) four distinct domains of discourse:

- the subject world, about which information will be stored in the information system;
- the usage world, consisting of the environment within which the information system will eventually function;
- the system world, of the various incarnations of the information system itself, ranging from non-procedural requirements specifications to code; and
- the development world, of teams, development processes, design goals, schedules and decisions.

The knowledge involved is both generic – the kind one learns in courses at school – and specific – having to do with a particular system. Moreover, it is essential for the development process to accumulate information concerning the *relationship* between the above four worlds.

It seems intuitively clear that the ability to explicitly capture and manipulate these kinds of knowledge can be helpful for many software engineering tasks (including requirements acquisition, expert support for development, maintenance and staff training), and provides the basis for a variety of computer tools to support these activities. These expectations have been confirmed by our experience in the DAIDA project.

To support the above paradigm of knowledge-intensive software development, we need knowledge bases. This paper has shown how Telos can be used to build *meta-models* of the various subworlds and software engineering activities involved in developing an information system, and to populate these metamodels with specifications of particular software environments and development projects.

Telos is an object-centered language which has a number of special features that have enhanced its utility for maintaining a software knowledge base:

- it supports the organization of knowledge built up from "atomic facts" through the use of classification and generalization hierarchies;
- it is relatively easy to extend and customize with abbreviations by providing higher-order classes, including attribute meta-classes; the syntax of "attribute categories" and the underlying simple framework of "propositions" are particularly useful;
- it supports evolving views of objects and a hypertext-style browsing interface, by treating attributions as individuals;
- the consistency of the knowledge entered can be verified through constraint rules, and new values can be inferred by triggering the evaluation of rules;
- a complete "longitudinal" view of various domains can be captured because of the powerful yet tractable model of time which has been tightly integrated into the language;
- the evolution of the knowledge base can be recorded through the use of "belief" times associated with facts, and this can be used to support software maintenance;

There have been three prototype implementations of the language, all in PROLOG, carried out at the University of Crete [53], the University of Passau [21] and SCS Hamburg [29]. In addition to the DAIDA project, the language was/is used in a number of research projects, including ESPRIT projects LOKI [8], ITHACA [44] and MULTIWORKS [27], with generally positive feedback.

In a nutshell, the contribution of Telos lies in its adaptation of ideas from knowledge representation, deductive databases and requirements modeling languages in order to offer a language that can be used to tackle a broader class of modeling tasks – arising from information system development tasks – than those attempted by other proposals.

Design records for large information systems, formally or informally represented, are bound to contain millions of facts. If the reader were to accept the thesis that we can develop language facilities that are expressively adequate for the task at hand, it would still not be possible to manage effectively such records due to the lack of suitable knowledge base management systems, e.g., ones that can manage knowledge bases with $O(1M)$ facts, offering implementation techniques for query optimization, concurrency control and recovery. Even though unavailable yet, we believe that such facilities can and will be built in the next few years and will open the way towards more systematic, effective and productive software development technologies. We also believe that the key to such new technologies is and will continue to be the availability of *all relevant knowledge* to *human* designers.

Acknowledgements

We gratefully acknowledge the contribution of insightful ideas, suggestions and moral support from Sol Greenspan (GTE Laboratories, Waltham, MA) whose thesis provided a rationale and a springboard for this research; Yannis Vassiliou, Thodoros Topaloglou, Manolis Marakakis and others (Institute of Computer Science, Iraklion, 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; Thomas Rose, Manfred Jeusfeld and others (University of Passau, Germany) who carried out a second implementation; as well as other members of the ESPRIT projects LOKI and DAIDA. Last, but not least, we would like to thank Lawrence Chung, Brian Nixon, Martin Stanley and other members of the Taxis group at the University of Toronto for providing a friendly and stimulating research environment.

References

[1] Abiteboul, S., and Grumbach, S. COL: a Logic-based Language for Complex Objects. In *Proceedings of the International Conference on Extending Data Base Technology* (Venice, Italy, March 1988).

[2] Allen, J. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM 26*, 11 (November 1983), 832–843.

[3] Allen, J., and Hayes, P. A Common-Sense Theory of Time. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (Los Angeles, CA, 1985), pp. 528–531.

[4] Allen, J., and Hayes, P. Moments and Points In an Interval-Based Temporal Logic. *Computational Intelligence 5* (November 1989), 225–238.

[5] Attardi, G., and Simi, M. Consistency and Completeness of OMEGA, a Logic For Knowledge Representation. In *Proceedings of IJCAI-81* (Vancouver, B.C, 1981), pp. 504–510.

[6] Balzer, R., and Goldman, N. Principles of Good Software Specification and Their Implications For Specification Languages. In *Proceedings of the Conference on Specifications for Reliable Software* (Boston, MA, 1979), pp. 58–67.

[7] Biggerstaff, T., and Perlis, A., Eds. *Software Reusability (Vol. 1 and 2).* ACM Press, 1989.

[8] Binot, J.-L., Demoen, B., Hanne, K.-H., Solomon, L., Vassiliou, Y., von Haan, W., and Wachtel, T. LOKI: A Logic Oriented Approach to Data and Knowledge Bases Supporting Natural Language Interaction. In *Proceedings of the ESPRIT '88 Conference* (1988), North-Holland, pp. 562–577.

[9] Borgida, A., Greenspan, S. J., and Mylopoulos, J. Knowledge Representation as the Basis to Requirements Specification. *IEEE Computer 18*, 4 (1985), 82–91.

[10] Bowen, K., and Kowalski, R. Amalgamating Language and Meta-Language. In *Logic Programming*, K. Clark and S. Tarnlund, Eds. Academic Press, 1982, pp. 153–172.

[11] Brachman, R., Fikes, R., and Levesque, H. KRYPTON: Integrating Terminology and Assertion. In *Proceedings of AAAI-83* (Washington, DC, 1983), pp. 31–35.

[12] Brachman, R. J., and Levesque, H. J., Eds. *Readings in Knowledge Representation.* Morgan Kaufmann, 1985.

[13] Brachman, R. J., and Levesque, H. J. The Knowledge Level of a KBMS. In *On Knowledge Base Management Systems*, M. Brodie and J. Mylopoulos, Eds. Springer-Verlag, 1986, pp. 9–12.

[14] Bubenko, J. On Concepts and Strategies for Requirements and Information Analysis. In *Prodeedings of IFIP-80* (1980).

[15] Chen, P. The entity-relationship model-Towards a unified view of data. *ACM Transcactions on Database Systems 1*, 1 (March 1976), 9–36.

[16] Chung, K., Katalagarianos, P., Marakakis, M., Mertikas, M., Mylopoulos, J., and Vassiliou, Y. From Information System Requirements to Designs: A Mapping Framework. Technical Note 53, Computer Systems Research Institute, University of Toronto, November 1989.

[17] Conklin, J., and Begeman, M. A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems 6*, 4 (1988), 303–331.

[18] des Rivières, J., and Levesque, H. J. The Consistency of Syntactical Treatments of Knowledge. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning About Knowledge* (Los Altos, California, 1986), J. Y. Halpern, Ed., Morgan Kaufmann Publishers, Inc., pp. 115–130.

[19] Devanbu, P., P., S., B., B., and R., B. A Knoweldge-Based Software Information System. In *Proceedings of IJCAI-89* (1989), pp. 500–501.

[20] Dittrich, K., Ed. *Advances in Object-Oriented Database Systems (Proceedings of the 2nd International Workshop on Object-Oriented Database Systems).* Lecture Notes in Computer Science, 334, Springer Verlag, 1988.

[21] Eherer, S., Jarke, M., Jeusfeld, M., Miethsam, A., and Rose, T. A Global KBMS for Database Software Evolution: ConceptBase 2.0 User Manual. Tech. Rep. MIP-8936, University of Passau, 1989.

[22] Frisch, A. A General Framework for Sorted Deduction: Fundamental Results on Hybrid Reasoning. In *Proceedings of 1st International Conference on Principles of Knowledge Representation and Reasoning* (Toronto, Ontario, 1989), R. Brachman, H. Levesque, and R. Reiter, Eds., pp. 126–136.

[23] Gallaire, H., Minker, J., and Nicolas, J. Logic and Databases: A Deductive Approach. *ACM Computing Surveys 15*, 2 (1984), 52–57.

[24] Gallaire, H., and Nicolas, J.-M. How to Look at Deductive Databases. In *Foundations of Knowledge Base Management*, J. Schmidt and C. Thanos, Eds. Springer Verlag, 1989, pp. 119–127.

[25] Greenspan, S. J. *Requirements Modelling: A Knowledge Representation Approach to Software Requirements Definition.* PhD thesis, Dept. of Computer Science, University of Toronto, 1984.

[26] Hagelstein, J. Declarative approach to information systems requirements. *Knowledge-Based Systems 1*, 4 (1988), 211–220.

[27] Hahn, U., Jarke, M., Kreplin, K., Farusi, M., and Pimpinelli, F. Co-AUTHOR: a hypermedia group authoring environment. In *Proceedings of the European Conference on Computer-Supported Cooperative Work* (Gatwick, United Kingdom, 1989).

[28] Hahn, U., Jarke, M., and T., R. Group Work in Software Projects. In *Proc. IFIP WG 8.4 Conference on Multi-User Applications and Interfaces* (Iraklion, Crete, Greece, September 1990).

[29] Haidan, R., and Meyer, R. Requirements Modeling and System Specification In a Logic-Based Knowledge Representation Framework. Technical Report, ESPRIT project 892 (DAIDA), SCS Informationstechnik, Hamburg, Germany, 1990.

[30] Hudson, S., and King, R. Cactis: A Self-adaptive, Concurrent Implementation of an Object-Oriented Database Management System. *ACM Transactions on Database Systems 14*, 3 (1989), 291–321.

[31] Hull, R., and King, R. Semantic Database Modelling: Survey, Applications and Research Issues. *ACM Computing Surveys 19*, 3 (1987), 201–260.

[32] Jarke, M., Jeusfeld, M., and Rose, T. Software Process Modeling as a Strategy for KBMS Implementation. In *Proceedings of the First Internatinal Conference on Deductive and Object-Oriented Databases* (Kyoto, Japan, 1989), pp. 496–512.

[33] Jarke, M., Jeusfeld, M., and Rose, T. A software process data model for knowledge engineering in information systems. *Information Systems 15*, 1 (1990), 86–115.

[34] Jarke, M., Mylopoulos, J., Schmidt, J., and Vassiliou, Y. Information Systems Development as Knowledge Engineering: the DAIDA Project. Technical Report, ESPRIT project 892 (DAIDA), Forthcoming.

[35] Kifer, M., and Lausen, G. F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (1989), pp. 134–146.

[36] Konsysnki, B., and Kotteman, J. Dynamic Metasystems for Information Systems Development. In *Prodeedings of 5th International Conference on Information Systems* (1984), pp. 187–204.

[37] Koubarakis, M., Mylopoulos, J., Stanley, M., and Borgida, A. Telos: Features and Formalization. Tech. Rep. KRR-TR-89-4, Dept. of Computer Science, University of Toronto, 1989.

[38] Kramer, B. The representation of programs in the procedural semantic network formalism. Master's thesis, Dept. of Computer Science, University of Toronto, 1980.

[39] Mylopoulos, J., Bernstein, P. A., and Wong, H. K. A Language Facility for Designing Interactive Data-intensive Applications. *ACM Transactions on Database Systems 5*, 2 (1980), 185–207.

[40] Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M. Telos: A Language for Representing Knowledge About Information Systems. Tech. Rep. KRR-TR-89-1 (Revised), Dept. of Computer Science, University of Toronto, August 1990.

[41] Neighbors, J. Draco: A Method for Engineering Reusable Software Systems. In *Software Reusability*, T. Biggerstaff and A. Perlis, Eds., vol. 1. ACM Press, 1989, pp. 295–319.

[42] Olle, T., Sol, H., and A.A., V.-S., Eds. *Information Systems Design Methodologies: A Comparative Review*. North Holland, 1982.

[43] Pernici, B. Objects with Roles. In *Object-Oriented Development*, D. Tsichritzis, Ed. Centre Universitaire d'Informatique, Universite de Geneve, Switzerland, 1989, pp. 75–100.

[44] Profrock, A.-K., Ader, M., Muller, G., and Tsichritzis, D. ITHACA: An Overview. Tech. rep., Nixdorf Software Engineering GmbH, Berlin, 1989.

[45] Reiter, R. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, M. Brodie, J. Mylopoulos, and J. Schmidt, Eds. Springer Verlag, 1984, pp. 191–233.

[46] Rich, C. Knowledge Representation Languages and Predicate Calculus: How to Have Your Cake and Eat it Too. In *Proceedings of AAAI-82* (Pittsburgh, 1982).

[47] Rich, C., and Waters, R., Eds. *Artificial Intelligence and Software Engineering*. Morgan Kaufmann, 1986.

[48] Rose, T., and Jarke, M. A Decision-Based Configuration Process Model. In *Proceedings of 12th International Conference on Software Engineering* (Nice, France, 1990).

[49] Ross, D., and Schoman, K. Structured Analysis for Requirements Definition. *IEEE Transactions on Software Engineering* (1977), 49–60.

[50] Snodgrass, R. The Temporal Query Language TQuel. *ACM Transcactions on Database Systems 12*, 2 (June 1987), 247–298.

[51] Srinkath, R., and Jarke, M. The design of knowledge-based systems for managing ill-structured software projects. *Decision Support Systems 5*, 4 (1989), 425–447.

[52] Stanley, M. CML: A knowledge representation language with application to requirements modelling. Master's thesis, Dept. of Computer Science, University of Toronto, 1986.

[53] Topaloglou, T., and Koubarakis, M. Implementation of Telos: Problems and Solutions. Tech. Rep. KRR-TR-89-8, Dept. of Computer Science, University of Toronto, 1989.

[54] Toulmin, S. *The Uses of Argument*. Cambridge University Press, 1958.

[55] Vilain, M., Kautz, H., and van Beek, P. Constraint Propagation Algorithms for Temporal Reasoning: a Revised Report. In *Readings in Qualitative Reasoning about Physical Systems*, D. Weld and J. de Kleer, Eds. Morgan Kaufmann, 1989, pp. 373–381.

[56] Winograd, T., and Flores, F. *Understanding Computers and Cognition*. Ablex Corporation, 1987.