Conceptual Modelling and Telos[1]

John Mylopoulos[2]
University of Toronto

Abstract

We review basic premises underlying the application of conceptual modelling to the development of information systems and point out a fundamental problem arising from the broad range of concepts that need to be modelled. We then argue that conventional conceptual models are weak for such broad domains of discourse because they come with built-in collections of primitive notions in terms of which conceptual modelling is to be done. *Telos* is then introduced as a conceptual modelling language designed for capturing knowledge about information systems and it is argued that, unlike its peers, it offers facilities not only for modelling an application but also the notions used to model an application. The presentation of features of the language is eclectic and generally non-technical. Details about Telos can be found in [Mylopoulos90] and [Koubarakis89].

0. A Fable

Imagine! You are an information systems analyst with HyperTech Industries Unlimited ("Hype", for short) and a client comes along wanting a custom-made information system for her large and powerful multinational organization, Home Periodicals Inc. (hereafter "Hope") to manage information flow at the executive level, including memos, policy statements, reports and minutes. Your mission, should you decide to accept it, is to design the system to be delivered to Hope and, more importantly, keep them happy and off your boss' back.

Now, you happen to be a recent graduate from a computer science programme and you are aching to use all these great ideas you learned back in college. So, you dig up your course notes from your home basement and proclaim at the next meeting with your boss and your client that you intend to go about this using the latest from the best minds in the field. Ignoring the anxious looks on their faces -- and the snears behind your back among your fellow analysts the following day -- you begin your task going over all your material, reading up references and taking notes.

At the end of your search, you proudly present your boss with your findings. There seems to be unanimous agreement among experts, you note, that your job should begin with a requirements analysis, where you define the problem at hand. In your case, you need to describe Hope as an organization, how it is structured, what kinds of information flow within and among departments, what are the patterns of flow and how would the proposed system come into the picture. This phase is invariably followed, you announce, by a system design, where the database, applications programs and interfaces constituting the information system are specified. As your boss waits patiently for the punchline, you point out that there seems to be little concrete help in the form of methods and tools for requirements analysis. Some people use diagrammatic notations of one sort or another, but these are mere sketches of the subject matter, be it the organization where a system will be deployed or the system itself. Sketches are fine sometimes, you remark, but can be ambiguous leading to misunderstandings between Hype and its clients. Your boss shakes his head knowingly. Others adopt notations from computer science, you continue, such as flow charts, Petri nets and finite state machines. But these are largely inadequate, since they were invented for an entirely different purpose. The same applies, in your opinion, for others who become born-again mathematicians hoping to find there the modelling tools that are required. Following your carefully laid out script, you draw an analogy between what requirements analysts practice and the proverbial drunk who late one night is looking under a street lamp for his keys, lost elsewhere, because there he can at least see.

As your boss becomes restless, you get to the punchline. There seem to be two approaches that are worth considering. The first is to use the language of your ancestors (nowadays called "natural language") to describe

[1] Appeared in Loucopoulos, P. and Zicari, R. (eds.) *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development*, Wiley, 1992.

[2] Author's full address: Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, Canada M5S 3H5.

the requirements on the new system. The second is to try out something new called "conceptual modelling", which is supposed to allow you to build a description of the subject matter -- including the system, the information it will handle and the environment it will function in -- that is consistent to the way humans (executives included) conceptualize that same subject matter.

Your boss stops looking at the ceiling and is now staring directly into your eyes. He knows all about the language-of-his-ancestors solution and its deficiencies. "Could it be that this guy is on to something?", he wonders. The tools offered for conceptual modelling, you continue, are based on ideas from knowledge representation developed in Artificial Intelligence. The key concern is to structure the representation of knowledge about a subject consistently to the way humans structure that same knowledge and to make sure that the procedures that use these representations draw the same, or at least a subset of the inferences people would draw when confronted with the same facts. Knowledge bases built this way -- you stress the words "knowledge bases" and your boss seems downright impressed -- can be thought as formal repositories of knowledge that serve as formal, unambiguous contracts between Hype and its customers. They can also help end users figure out what the system eventually does, by giving them insights into what the system was intended to do in the first place.

Your boss is ecstatic. "Here is another wacky idea", he thinks "but it can't be any worse than the others. Let's try it!" He gives you his blessing and you give him a good book on conceptual modelling[3] for background reading before embarking on the Hope project. For the rest of the week you're definitely on the good side of your boss' balance sheet. And it's all thanks to conceptual modelling.

1. Introduction

1.1 Motivation

Data models revolutionized data processing in the early '70s by offering data abstractions for the definition of a database thereby hidding implementation details from the database user. However, "classical" data models, grounded on mathematical and computer science concepts, such as relations and records, offered little to aid database designers and users in **interpreting** the contents of a database. Indeed, this criricism first voiced against the database technology of the day shortly after the introduction of data modelling[4] is still with us today.

Semantic data models came about in the mid-seventies in response to this perceived need for better modelling tools to "capture more of the semantics of an application" [Codd79]. Semantic data models, starting with Abrial's *semantic model* [Abrial74] and Chen's *entity-relationship model* [Chen76] combined simple knowledge representation techniques, often borrowed from semantic networks [Findler79] with database technology, leading to systems that promised both modelling power and performance[5].

The research community working on data modelling further broadened its horizons in the early eighties by noting similarities in goals with programming language research focusing on abstract specifications of programs and knowledge representations ideas going beyond semantic networks. *Conceptual modelling* was introduced as a term reflecting this broader perspective[6]. Since the early eighties conceptual modelling has found applications beyond capturing the meaning of a database, including modelling organizational environments -- say, an office -- modelling software development processes or just plain modelling some part of the world for purposes of human communication and understanding.

---

[3] The book turns out to be Brodie, M., Mylopoulos, J. and Schmidt, J. (eds.), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages,* Springer-Verlag, 1984. Naturally!

[4] See, for example, [Schmid74].

[5] See [Mylopoulos88] for a collection of influential papers on this topic.

[6] In fact, the term "conceptual modelling" was used in the seventies as well, either as a synonym to semantic data modelling or in the technical sense of the ANSI /X3/SPARC report [ANSI75] where it referred to a model that allows the definition of schemata lying between external views, defined for different groups of users, and internal schemata defining one or several databases. The term was used more or less, in the sense discussed here at the Pingree Park workshop on *Data Abstraction, Databases and Conceptual Modelling,* held in June 1980 [Brodie81].

The purpose of this paper is to review basic premises underlying the application of conceptual modelling to the development of information systems (section 2), to point to a fundamental problem of conceptual models arising from the breadth of the intended domain of discourse (section 3) and to show how this problem is overcome in the language Telos (section 4). The presentation is generally non-technical. Details about Telos can be found in [Mylopoulos90] and [Koubarakis89].

1.2 What is Conceptual Modelling?

Conceptual modelling is the activity of *formally* describing some aspects of the physical and social world around us *for purposes of understanding and communication*. Such descriptions, often referred to as *conceptual schemata*, require the adoption of a formal notation, a *conceptual model* in our terminology[7]. Conceptual schemata capture relevant aspects of some world, say an office environment and the activities that take place there, and can serve as points of agreement among members of a group, for example the workers in that office, who need to have a common understanding of that world. Conceptual schemata can also be used to communicate that common view to newcomers, through a variety of graphic and linguistic interfaces. Conceptual modelling has an advantage over natural language or diagrammatic notations in that it is based on a formal notation which allows one to "capture the semantics of the application". It also has an advantage over mathematical or other formal notations developed in computer science because unlike them, conceptual modelling supports structuring and inferencial facilities that are psychologically grounded. After all, the descriptions that arise from conceptual modelling activities are intended to be used by *humans*, not machines.

Before proceeding with more technical matters, it is worthwhile to contrast conceptual modelling with *knowledge representation* and *semantic data modelling*, both technical terms as well as research areas in their own right that have attracted much attention over the past 15 years[8]. All three activities involve capturing knowledge about a given subject matter. Knowledge representation, however, has traditionally focused on interesting reasoning patterns and how they can be accounted for semantically and computationally. As pointed out in [Borgida90], knowledge representation assumes that the knowledge bases resulting from the representation activity will be used by some other *system* performing an intelligent task such as planning or design (say, expert systems). Conceptual modelling, on the other hand, has been concerned with life-size models of portions of the world to be made available to human users, for purposes of understanding and communication. Naturally, this leads to an emphasis on efficiency and a focus on simplicity. The adequacy of a knowledge representation is ultimately determined by the "intellectual achievements" of systems that adopt it. The adequacy of a conceptual modelling notation rests on its contribution to the construction of models of reality that promote a common understanding of that reality among their human users.

Semantic data modelling shares purposes with conceptual modelling. However, semantic data modelling introduces assumptions about the way conceptual schemata will be realized on a physical machine (the "data modelling" dimension). Thus semantic data modelling can be seen as a more constrained activity than conceptual modelling, leading to simpler notations, but also ones that are closer to implementation. [Borgida90] presents a thoughtful and thorough contrast of knowledge representation and semantic data modelling.

Is conceptual modelling "informal", "ill-defined", "soft stuff" and the like? Some seem to think so, treating terms such as "knowledge base" with suspicion or even disdain. There are several sources for this suspicion, though none is justified. Firstly, most of us computer scientists were brought up within a culture spanning concepts from machine organization to assembly languages, to (high level) programming languages. Modelling the "real world" never figured much within those confines. Increasingly however, there is a realization among computer scientists that such world modelling can address fundamental problems in areas such as databases and software engineering[9]. Secondly, the foundations of the enterprise of knowledge representation, even in its conceptual/semantic data modelling form, are evolving along with our understanding of ontological, epistemological and semantic issues concerning human knowledge. This evolution dictates experimentation with ideas (even half-baked ones!) but doesn't

[7] These terms are introduced by analogy to data models and database schemata. The reader may want to think of data models as special conceptual models where the intended subject matter consists of data structures and associated operations.

[8] See, for example, the proceedings of the First International Conferences on Principles of Knowledge Representation, [KR89], or surveys of semantic data models such as [Hull87].

[9] In Artificial Intelligence awareness for the need to model aspects of the world goes back to early work on semantic information processing (see, for instance, [Minsky68]).

mean that the methods and/or results of this enterprise are lesser in rigour or in any other way to those used/produced when people were developing the breed of high level languages we now take for granted. Thirdly, much of the inspiration for new ideas for this enterprise has come from cognitive science. Some may think that this is hardly a source of hard, technical (as opposed to soft, mussy) ideas. But cognitive science *is* a perfect source for insights if one is interested not merely in computational systems that are well-defined and formal, but rather in computational systems which are well-defined and formal *and also work for people,* doing what they were intended to. In short, the inspiration from cognitive science puts some science into the enterprise of developing formal notations for modelling of any sort.

1.3 Conceptual Modelling and Information Systems

We are interested in conceptual modelling because it is useful in rationalizing and supporting information system development. Before looking at conceptual modelling notations in general and Telos in particular, we briefly examine the kinds of knowledge that need to be represented during the information system development process. Figure 1 illustrates these kinds of knowledge by classifying them into four "worlds"[10].

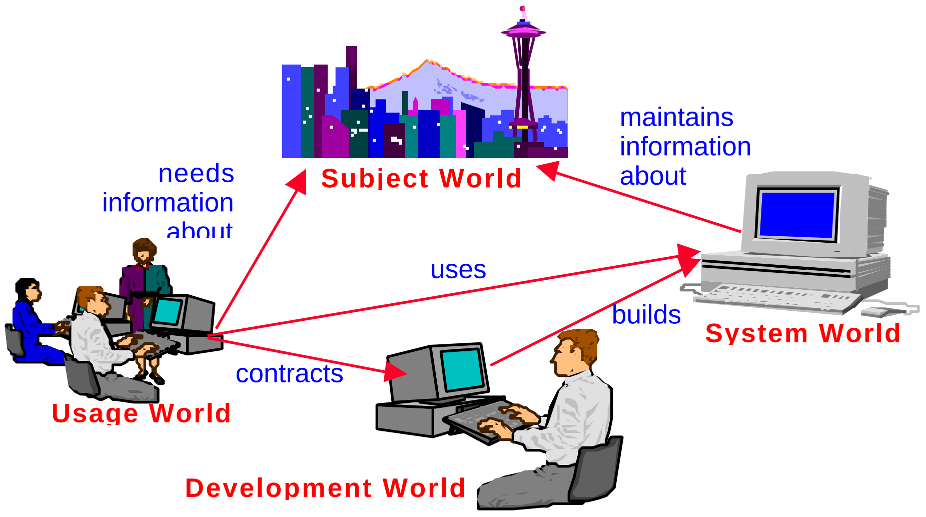

Figure 1

The **subject world** consists of the subject matter for the system, i.e., the world about which information is maintained by the system. For instance, the subject world for a banking system consists of customers, accounts, transactions, balances, interests rates and the like. The **system world**, on the other hand, describes the information system itself at several layers of implementation detail. These layers may range from a specification of functional requirements for the system, to a conceptual design and an implementation. The **usage world** describes the (organizational) environment within which the system is intended to function and consists of agents, activities, tasks, projects, users, user interfaces (with the system) and the like. Finally, the **development world** describes the process that led to the development of the information system, the team of systems analysts and programmers involved, their adopted methodology and schedule, their design decisions along with the justifications behind those. All of this knowledge is relevant during the initial development of the system but also later on during maintenance and use. All of this knowledge needs to be represented, somehow, in any attempt to offer a comprehensive treatment to the software engineering problem of building information systems. Precisely this point of view is adopted by the DAIDA project [DAIDA91], among others. The challenge, from a conceptual modelling perspective, is to provide facilities for this task which are expressively adequate and computationally manageable so that they offer at least the promise of a new knowledge-based paradigm for software development.

It is interesting to note that information system development is a particularly challenging software engineering problem and an excellent application area for conceptual modelling ideas. Unlike other kinds of software systems, say operating systems or scientific computing packages, information systems are doubly grounded in the "real

[10] This classification, along with figure 1, is based on discussion in [Mylopoulos90].

world", through the information they maintain about it and because they are embedded, in a very strong sense, in it. These kinds of knowledge are irrelevant for the software engineering problem of building, say, a package of subroutines solving a class of differential equations or an operating system.

2. Modelling an Application: A First Try

One of the early conscious efforts to apply conceptual modelling ideas to software engineering is described in Sol Greenspan's Ph.D. thesis [Greenspan84]. The thesis proposes to use knowledge representation ideas in order to introduce "world modelling" during *requirements definition* [Roman85], the initial phase of software development where the systems analyst attempts to understand the problem at hand before proceeding to devise any sort of a solution. Greenspan's thesis offered the requirements modelling language *RML* as a tool for formally specifying the functional requirements for a given information system. RML adopts much of the structural framework of semantic data models such as Taxis [Mylopoulos80] but substitutes the procedural sublanguage of Taxis, including generalization and attribution for structuring purposes, with an assertional one, used to specify constraints or deductive rules on classes.

RML distinguishes three types of objects: *entities*, *activities* and *assertions*, all of which have attributes that relate them to other objects. Moreover, following in the footsteps of Taxis, every attribute is classified into one of several *attribute categories*. For instance, figure 2 shows the definition for the entity classes `Person` and `Patient` and the activity class `Admit` (a patient to a hospital). These definitions are intended to describe persons and patients, from the point of view of a hospital admistrator perhaps, and to convey the idea that admitting a new patient (to a hospital) involves two sub-activities which respectively obtain information from the patient (`GetInfo`), and assign her to a bed (`AssignBed`). The attributes of `Person` are classified as `necessary`, in which case they must have values for all instances of `Person`, `single`, meaning that they are single-valued, and `associations`, in which case they can have zero or more values at any time. The definition of `Patient` is more elaborate to illustrate some of the intricacies of RML. Its attributes include not only data values (classified under `necessary, single, unique part` and `association`), but also ones that specify what activities can produce patients, "consume" patients (in the sense that thay lead to the removal of an instance from the `Patient` class) and which can change the status of patients. In addition, `Patient` comes with two constaints (`rightPlace?` and `startClean?`) which must be true of any new instance of the `Patient` class. Likewise, the first three attributes of `Admit` identify attributes that are single-valued and must be there for every instance of the class (because that is the RML semantics for `necessary` and `single` attributes). The next two attributes (`document` and `checkIn`) are classified under `part` and specify sub-activities of `Admit`. Clearly, subactivities are to take place before the activity terminates and after it begins. This property is associated with `parts` attributes in RML. The last attribute, `canAdmit?`, defines a precondition which must be true every time `Admit` is instantiated. In the name of uniformity, RML treats assertions such as `HasAuthority(...)` as classes in their own right. A non-technical presentation of the philosophy behind and main features of RML appears in [Borgida85].

Note that `single` constraints attributes to be single-valued while `unique` constraints them to be keys. Thus, according to the definition of `Patient`, there is a one-to-one correspondence between instances of `Patient` and `MedicalRecord`.

Generally then, RML offers a notation for conceptual modelling purposes which combines object-orientation[11], including structuring facilities, with an assertional sublanguage used to specify constraints and deductive rules. Such a framework is shared by many other proposals which claim to tackle all or part of the conceptual modelling problem [Webster87]. Unfortunately, if one is to take seriously the broad application scope of conceptual modelling for purposes of information system development, expounded in the previous section, RML and its peers suffer from a serious weakness. Its view of the world (defined in its notions of entity and activity) is **fixed** in the sense that these notions are built into the language. Indeed, the properties mentioned for the attribute categories used in the example of figure 2 are defined formally as part of the RML definition [Greenspan86]. Let's look again at the four worlds about which knowledge needs to be represented to support the information system development process. The subject world could be anything, from a static world where there are no activities to a world of chemical compounds and liquids where even the notion of object identity is problematic. At best, RML can be said to offer appropriate

[11] In the sense that building up a representation consists of an iterative description of concepts and individuals rather than an iterative statement of true facts [Mylopoulos90b].

modelling tools for a typical application. But a typical application is much like the legendary family with 2.2 kids: it only exists on paper and will never be encountered by the systems analyst in the trenches[12].

[12] The hero of the fable, for instance, frantically working on the Hope project.

```
Entity Class Person with
  necessary single
        name: Name
        gender: {'male, 'female}
  association
        addr: Address
        nextOfKin: Person
end Person

Entity Class Patient isA Person with
  necessary unique single
        record: MedicalRecord
  association
        loc: Ward
        room: Room
        physician: Doctor
        currentBill: $Value
  producer
        register: Admit(person← this, toWard←loc)
  modifier
        assess: Assess(patient←this)
  consumer
        release: Discharge(patient←this)
        decease: Certify(patient←this)
  initially
        rightPlace?: record.place = loc
        startClean?: currentBill = $0.00
end Patient

Activity Class Admit with
  necessary single
        newPatient: Person
        toWard: Ward
        admitter: Doctor
  parts
        document: GetInfo( from ← newPatient)
        checkIn: AssignBed( toWhom ← newPatient,
                          onWard ← toWard)
  precondition
        canAdmit?: HasAuthority( who ← admitter,
                                where ← toWard)
end Admit                       ...
```

Figure 2

What about the adequacy of RML for modelling the usage world alluded to earlier? Usage worlds consist of an organizational environment and demand more specialized notions for their modelling, including interfaces, agents playing roles and having authority and responsibilities, projects involving tasks having deadlines and using resources, messages and communication. Entities and activities as modelling notions seem rather primitive for such a relatively focussed application domain and the modeller may well demand substantially more.

In all fairness to RML, it was never intended for modelling either system or development worlds. However, if one were to add to or replace altogether the notions of entity and activity offered by RML with others deemed appropriate for development or systems world modelling, there would still be the problem that these notions are built into the linguistic framework, and the modeller may find them inappropriate for her modelling task. RML and its peers are much like a programming language with a fixed set of subroutines. Changing the subroutine set doesn't solve the problem. What is needed is the ability to tailor the set of subroutines offered to the application at hand. The solution then to this weakness of conceptual models is to offer the modeller the freedom to define her own notions (or choose the ones most appropriate from a library). In other words, conceptual models, unlike their data model ancestors, *cannot fix the primitive notions offered to the modeller because of the breadth and range of conceptual modelling applications.*

Before embarking on an account of the Telos solution to this problem, we ought to mention a couple of solutions that won't work. The first is to simply abandon the idea of attribute categories altogether. After all, the semantics of attributes (say, being necessary or single-valued) could be defined explicitly in terms of axioms (constraints and/or deductive rules). The problem with this solution is that it forces the user of the notation to say much more than she would otherwise need to. Thus, if a particular class definition includes n attributes each being a member of k attribute categories and each attribute category requires m axioms for its definition, the elimination of attribute categories necessitates the explicit definition of n*k*m axioms. Even if k = 0, i.e., attribute categories don't need associated axioms, there seems to be merit in using them for mnemonic purposes, as in

```
Entity Class Person with
  groupA
        name: Name
        gender: {'male, 'female}
        addr: Address
  groupB
        nextOfKin: Person
        age: { 0::100 }
end Person
```

Grouping of attributes would be particularly useful in applications where the average number of attributes associated to each class is large.

A second non-solution involves associating "canned constraints" directly with each attribute, as in this variation of the definition of `Person` presented earlier:

```
Entity Class Person with
        name: Name  necessary, single, unchangeable
        gender: {'male, 'female} necessary, single
        addr: Address   at-least-one
        nextOfKin: Person single
        age: { 0::100 } initial (it = 0)
end Person
```

This type of solution is used heavily in some semantic data models, such as SDM [Hammer81], but also in some knowledge representation schemes such as KL-ONE [Brachman85] or KEE [Fikes85] where they are referred to as *facets.* The problem here is that we still have to fix the semantics of the modalities in the conceptual model. All that has changed from the original situation is that instead of defining the semantics of attributes by classifying attributes under one or more categories, we do so by associating to attributes one or more modalities or facets.

3. Conceptual Modelling in Telos

Telos begins to address the deficiencies pointed out in the last section by abolishing altogether the distinction between nodes/entities and links/attributes. In Telos jargon, everything in the knowledge base is a *proposition.* Each proposition has four components named respectively `from, label, to` and `when.` The first three of the four components simply specify a labelled edge between propositions. The fourth component specifies a time interval

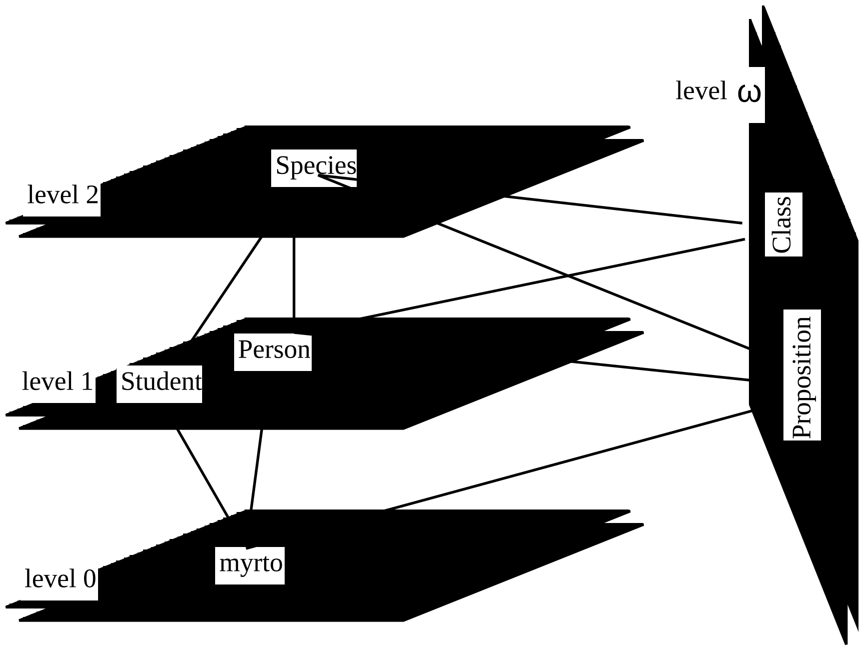

Figure 3

proposition which represents the "lifetime" of the relationship being represented by the proposition. Thus the proposition `[Maria, teacherOf, Myrto, 1990]` might represent the meaning of the statement

"Maria was teacher of Myrto during 1990"

Individuals are represented by self-referencing propositions `p` such that

```
from(p) = to(p) = p .
```

Non-individual propositions will be referred to as *attributes* in the sequel. Rules concerning classification, attribution and generalization apply to all propositions, attributes as much as individuals. In particular, all Telos propositions are categorized along the classification dimension into *tokens*, assumed to represent particular individuals in the domain of discourse, *simple classes*, assumed to represent generic concepts having particular individulas as instances, *metaclasses*, having simple classes as instances, etc. This defines an infinite dimension, shown in figure 3, along which propositions are placed. Propositions at level n can only be instances of ones at level n+1. Ω-classes constitute the only exception to this rule and are allowed to have instances from any level, including the ω-level (figure 3). Note that **all** propositions, individuals and attributes, are placed along this dimension. Figure 3 illustrates the configuration resulting from the classification of a handful of entity propositions along the classifcation dimension. All links in the figure represent instance links. Only some of these links are shown to keep this and subsequent figures relatively unclattered. The individuals classified include a token (`myrto`), two simple classes (`Person` and `Student`) one metaclass (`Species`) and two ω-classes (`Class` and `Proposition`) which are assumed to have respectively as instances all classes (including `Class` and `Proposition`) and all propositions (including attributes and `Proposition` itself).

Figure 4 shows the re-definition of the class `Person` and one of its instances, `myrto`, while figure 5 shows

```
TELL CLASS Person
  IN SimpleClass
  WITH
     attribute
        name: String
        friend: Person
END Person

TELL TOKEN myrto
  IN Person
  WITH
     friend
          bestFriend: marina;
                  : michelle;
                  : sarah;
     name
              : ‘Myrto Cheung’;
END myrto
```

Figure 4

(portions of) the semantic network configuration resulting from these definitions[13].

According to its definition, `Person` is an instance of `SimpleClass` -- another built-in class with instances all classes having tokens as instances -- also `Class`, the class of all classes simple or not --. Moreover, `Person` has two attributes, `friend` and `name` both of which are instances of the metaclass `[Class,attribute, Class,`

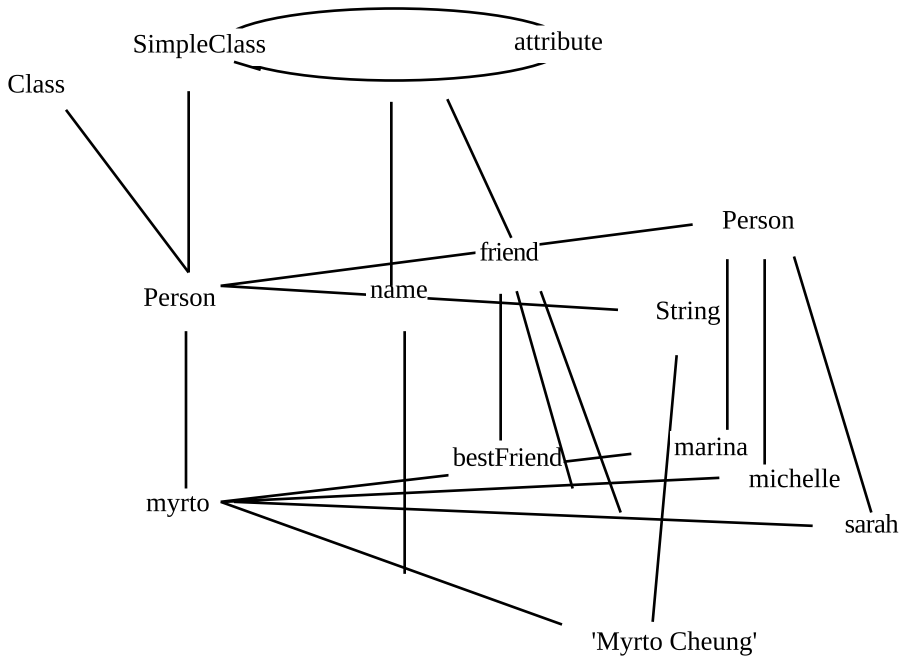

Figure 5

[13] Temporal components of propositions are omitted from the figure. Also, entities are represented graphically by their identifiers.

, ...]. Both friend and name are simple classes, instantiated by attribute tokens in the definition of myrto (figure 4). For example, the friend attribute (simple class) is instantiated three times in the definition of myrto. The first of these receives the label "bestFriend" while the others get system-generated labels not shown on figure 5.

Treating attributes as first class citizens allows us to use attribute metaclasses to represent RML-like attribute categories. For example, figure 6 shows a (partial, at this point) definition of the RML notions of entity and activity in terms of metaclasses (individual and attribute ones).

```
TELL CLASS EntityClass
   IN MetaClass
   WITH
      attribute
         necessary, unchanging, association: EntityClass
         single, unique: EntityClass
         producer, modifier, consumer: ActivityClass
         initially: AssertionClass
END EntityClass

TELL CLASS ActivityClass
   IN MetaClass
   WITH
      attribute
         participant: EntityClass
         part: ActivityClass
         precondition: AssertionClass
END ActivityClass
```

Figure 6

We can now use these metaclasses to define Person, Patient and Admit more or less as they were defined in RML (figure 7). Of course, the user can define here altogether different or additional attributes if she so chooses. A portion of the resulting configuration is shown in figure 8. Here attribute metaclasses such as necessary and single are used in exactly the same way attribute simple classes friend and name were used earlier (see figure 4) in order to classify the attributes associated with the newly defined simple classes Person, Patient and Admit.

```
TELL CLASS Person
  IN EntityClass
  WITH
     necessary unchanging
       name: Name
       gender: {'male, 'female}
     association
       addr: Address
       nextOfKin: Person
END Person

TELL CLASS Patient
  IN EntityClass
  ISA Person
  WITH
     necessary, unique, single
       record: MedicalRecord
     association
       loc: Ward
       room: Room
       physician: Doctor
       currentBill: $Value
     producer, single
       register: Admit(person← this, toWard←loc)
     modifier
       assess: Assess(patient←this)
     consumer, single
       release: Discharge(patient←this)
       decease: Certify(patient←this)
     initially
       rightPlace?: record.place = loc
       startClean?: currentBill = $0.00
END Patient

TELL CLASS Admit
  IN ActivityClass
  WITH
     participant
       newPatient: Person
       toWard: Ward
       admitter: Doctor
     part
       document: GetInfo( from ← newPatient)
       checkIn: AssignBed( toWhom ← newPatient,
                        onWard ← toWard)
     precondition
       canAdmit?: HasAuthority( who ← admitter,
                              where ← toWard)
                    ...
END Admit
```

Figure 7

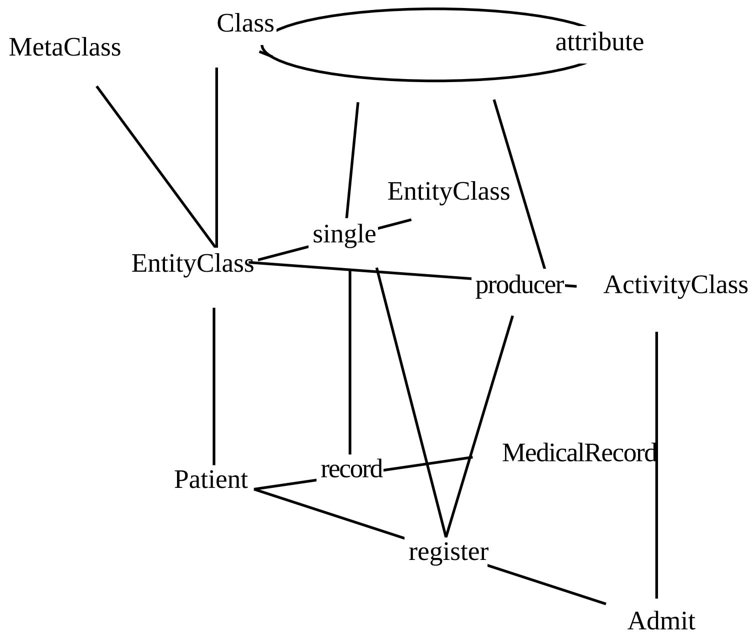

Figure 8

The treatment of attribute categories in terms of metaclasses discussed so far does not deal with axioms associated with these categories. For example, we would want to specify in the definition of the attribute metaclass labelled `single` that a single-valued attribute (represented as an instance of `single`) cannot have two values for the same proposition. Or we may want to state that `producer` attributes take as values activities which are supposed to produce instances of a particular class during their execution. To state such constraints, we need to introduce the assertion sublanguage offered by Telos for the specification of constraints.

The assertion sublanguage is a typed, first order language whose formulas are special propositions in Telos classified under the built-in class `AssertionClass.` Two built-in attribute metaclasses labelled respectively `integrityConstraint` and `deductiveRule` allow the classification of assertion attachment attributes into constraints or deductive rules. For example, the following revised definition of `Patient` includes a constraint that states that every medical record corresponds to a single patient (the `unique`ness constraint) and a deductive rule which states that a patient's ward is the ward to which her room is attached (figure 9). The assertion language includes special predicates for isA and instanceOf relationships (`isa` and `in` respectively) and special selectors which allow navigation through the graph structure. For example, `x.p` returns the set of all destinations of attributes which have `x` as source and are instances of an attribute class with source a class of which `x` is an instance and also have label `p:`

```
x.p = { q | there exists an attribute v and an attribute class A
   such that v is an instance of A, from(v) = x, label(A) = p
         and to(v) = q}
```

This means that `p.record` evaluates to the set of all medical records associated with patient `p` through attributes that are instances of the attribute class `[Patient, record, MedicalRecord,...].` Likewise, `p.room`

```
TELL CLASS Patient
 IN EntityClass
 ISA Person
 WITH
     necessary, single
        record: MedicalRecord
    association
        loc: Ward
        room: Room
           ...
    integrityConstraint
        $ (ForAll p, q/Patient, m/MedicalRecord)
         [(m   p.record  m   q.record   p = q] $
    deductiveRule
        $ (ForAll p/Patient
         [x   p.room.ward    x   p.ward]
           ...
        END Patient
```

Figure 9

evaluates to the set of all rooms associated with `p` through instances of `[Patient, room, Room,...]`. In addition to the "dot" selector, Telos offers three other selectors:

`x^p` evaluates to the set of destinations of attributes having `x` as source and `p` as label, i.e.,

```
 x^p = { q | there exists an attribute v such that from(v) = x,
             label(v) = p and to(v) = q}
```

`x|p` evaluates to the set of attribute propositions with source `x` which are instances of an attribute class with a source that has `x` as instance and has label `p`. i.e.,

```
 x|p = { q | from(q) = x and there exists an attribute class A
           such that label(A) = p and x is an instance of from(A)}
```

`x!p` evaluates to the set of attribute propositions with source `x` and label `p`, i.e.,

```
 x!p = { q | from(q) = x and label(q) = p}
```

```
TELL CLASS EntityClass
 IN MetaClass
 WITH
    attribute
        single: EntityClass
        producer: ActivityClass
              ...
    integrityConstraint
        atMostOne? $ (ForAll u/EntityClass!single, p, q/Proposition)
         [(p in u  q in u   from(p) = from(q)
           when(p) overlaps when(q)   p = q] $
        producedBy?:
          $ (ForAll u/EntityClass!producer, p/Proposition, t/Time)
            [(p in from(u) @ t    (Exists q/u)
              [ from(q) = p   start(t) after start(to(q))
                    start(t) before end(to(q))]]
END EntityClass
```

Figure 10

We are now ready to redefine `EntityClass` in a way that includes appropriate constraints for attribute metaclasses such as `single` and `producer` (figure 10). The `atMostOne?` constraint of `single` specifies, roughly speaking, that for any two instances (`p` and `q`) of an instance `u` of the `single` attribute metaclass (which is the value of the expression `EntityClass!single`), if `p` and `q` have the same source (`from` component) then they have the same destination (figure 11). In other words, `u` is a single-valued attribute (since it is an instance of `EntityClass!single`) therefore no instance of its source should have associated two of its instances as attributes. This is not quite the full Telos story, since it doesn't deal with the temporal components of the propositions involved, but does illustrate the capability of Telos to represent constraints which are associated with metaclasses and apply to instances of their instances.

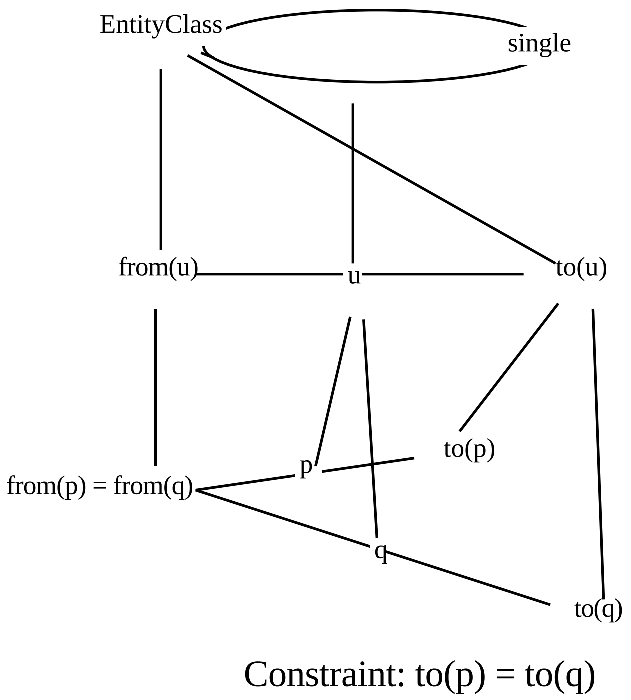

Figure 11

The `producedBy?` constraint is mildly more complicated. Its intent is to make sure that every instance of an entity class is produced by an instance of (one of) its declared `producer` activities. This is accomplished by constraining the time interval specifying when is a proposition an instance of a class (in the formula, `t`) to start during some producer activity of the class. Thus, (`p in from(u) @ t`) is true when `t` is the time interval during which `p` is an instance of `from(u).` The temporal assertions `(start(t) after start(to(q))` and `start(t) before end(to(q)` are not quite Telos expressions but do give the flavour of how temporal constraints can be represented using the `when` component of propositions. The `producedBy?` constraint may be made more realistic by requiring that each class instance starts during **exactly one** producer activity (in its current form, there could be several activities during whose lifetime `t` begins).

Note that in its present definition, the attribute metaclass labelled `single` can only be used for instances of `EntityClass`. As a final variation (twist?) of the simple running example, suppose we want to be able to use the `single` attribute metaclass anywhere in the knowledge base, for example, with activity classes or even metaclasses. After all, `single, necessary` and the like embody general constraints on attributes rather than ones specific to entities or activities. To accomplish this, we redefine `single`, this time as an attribute metaclass with `Class` as source (figure 12). This definition, defines the four components of the attribute metaclass `Single` to be respectively `Class, single, Class, AllTime` and still uses the `atMostOne?` constraint. However, because of its different source and destination it can be used to constraint attributes associated with any class within the knowledge base.

```
TELL CLASS Single
  COMPONENTS [Class, single, Class, AllTime]
  IN AttributeClass, MetaClass
  WITH
     integrityConstraint
  atMostOne? $ (ForAll u/EntityClass!single, p, q/Proposition)
        [(p in u  q in u  from(p) = from(q)
                     when(p) overlaps when(q)  p = q] $

END Single

TELL CLASS Patient
  IN Person
  WITH
       ...
       single
            record: MedicalRecord
  ...
END Patient
```

Figure 12

5. Concluding Remarks ... and a Moral

As indicated in the introduction, this paper is only intended to demonstrate how the classification dimension of Telos can be deployed to define appropriate concepts (in terms of individual and attribute metaclasses) for modelling any "world". We won't even attempt to discuss some of the other novel features of Telos, such as the tight integration of facilities for representing and reasoning with time into the semantic network framework of which the reader only got a glimpse; or the model-theoretic semantics of Telos, described in [Plexousakis90], where an attempt is made to account for different modes of existence, e.g., physical existence (characteristic of my car) vs past existence (characteristic of, say, Alexander the Great) vs abstract existence (characteristic of the number 7) vs non-existence (my cancelled trip to Japan).

A few words on the history of Telos. The language was initially conceived as a revamped RML, integrating improvements in a number of areas. The revamping effort was initiated in '85, as part of a research project funded by the European Community under the Esprit programme [LOKI88], with Alex Borgida, Yannis Vassiliou and the author as main contributors. A language called CML (Conceptual Modelling Language) was the result of this activity. The language is formalized in [Stanley86] and further studied -- and cleaned up -- in [Koubarakis88] and [Topaloglou89]. The latest version, obtained after several prototype implementations and some usage, has been named Telos [Mylopoulos90].

The implementation of Telos relies heavily on results from deductive databases [Hulin89], both for query processing -- complicated by the presence of Horn-like deductive rules -- and for constraint enforcement. Temporal reasoning is handled through a special-purpose inference engine based on recent efficient algorithms [Vilain89] and a number of heuristics. Three independent, Prolog-based implementations of Telos have been developed at SCS (Hamburg) [Gallagher86], the University of Passau [Jarke88] and the University of Crete [Vassiliou90] and are in use at several sites. Another, LISP-based implementation [Mylopoulos91] has been produced with expert system applications in mind. Moreover, [Sobiesiak91] describes a Smalltalk-based implementation of the structuring facilities of Telos intended to be used to structure hypertext databases. All these implementations include a window-based interface with graphical as well as textual forms of input and output. We are currently exploring the application of query optimization and concurrency control techniques adopted from DBMSs in an attempt to develop a truly efficient and robust implementation for Telos.

The adoption of Telos for the ITHACA project constitutes perhaps the most serious test todate for its claimed modelling advantages [Constantopoulos91]. ITHACA is a large ESPRIT project initiated in 1989 whose aim is to construct an object-oriented application development environment. An important component of the ITHACA environment is a *software information base* intended to facilitate software reuse. This information base is assumed to contain descriptions of software code, requirements and design specifications, run-time data, bug reports and the like for software developed using different methodologies, tools and programming languages. Using the structural part of Telos, the designers of the software information base have been able to define a number of associations among software descriptions that will serve as basis for structuring the software information base. These associations include, of course, generalization, classification and attribution supported by Telos, but also a form of similarity and correspondence relationships. Moreover, metaclasses have been introduced for languages or data models used by software to be included in the information base. This system, developed at the University of Crete, is based on a C++ implementation of a subset of Telos and is exceptionally fast.

Telos is one of several projects that aspire to advance the state-of-the-art in conceptual modelling. Terminological languages, in the tradition of KL-ONE and KRYPTON, such as CLASSIC, KANDOR and BACK provide another set of modelling features, focusing primarily on generalization as a structuring mechanism and the provision of a subsumption operation through which one can determine if the definition of one class constitutes a special case of that of another. Extensions of the Entity-Relation notation have also been offered for conceptual modelling.

Modelling aspects of the world, past, present or future, real or imaginary, for purposes of self-preservation, advancement or pleasure, has been a human endeavor since prehistoric times. *Conceptual* modelling offers the promise of a novel perspective and a new set of tools for advancing the state-of-the-art for this all-important human activity. Computer scientists have been working on conceptual modelling for a decade or more, depending on where one places its origins, developing notations, such as Telos or those mentioned in the previous paragraph, and applying them to "real world" problems. And yet, despite all this effort and the experimental tools, one can't point yet to a mature or even maturing technology for conceptual modelling, i.e., a set of tools along with an accompanying methodology that can be deployed to *systematize* conceptual modelling practice. Indeed, conceptual modelling today is roughly where you were left off at the end of the fable: somewhere between Hype and Hope.

Acknowledgements

Special thanks are due to all those who contributed to the design of Telos, particularly Alex Borgida (Rutgers University), Sol Greenspan (GTE laboratories), Matthias Jarke (University of Aachen), Manolis Koubarakis and Yannis Vassiliou (University of Crete).

Funding sources for the development of Telos include the National Science and Engineering Research Council of Canada, the Canadian Institute for Advanced Research, the Information Technology Research Centre funded by the Government of Ontario through its Centres of Excellence programme and the European Community through ESPRIT programme projects such as LOKI , DAIDA and ITHACA which adopted and applied different versions of Telos.

Bibliography

[Abrial74] Abrial, J-R., "Data Semantics", in Klimbie and Koffeman (eds.) *Data Management Systems*, North-Holland, 1974.

[ANSI75] ANSI/X3/SPARC Study Group on Database Management Systems, "Interim Report", *FDT 7(2)*, 1975.

[Borgida85] Borgida, A., Greenspan, S. and Mylopoulos, J., "Knowledge Representation as a Basis for Requirements Specification", IEEE Computer 18(4), April 1985. Reprinted in Rich, C. and Waters, R., *Readings in Artificial Intelligence and Software Engineering*, Morgan-Kaufmann, 1987.

[Borgida85b] Borgida, A., "Features of Languages for the Development of Information Systems at the Conceptual Level", *IEEE Software 2(1)*, January 1985.

[Borgida90] Borgida, A., "Knowledge Representation and Semantic Data Modelling: Similarities and Differences", Proceedings Entity-Relationship Conference, Geneva, 1990.

[Brodie81] Brodie, M. and Zilles, S., (eds.) Proceedings of Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park Colorado, Joint SIGART, SIGMOD, SIGPLAN newsletter, January 1981.

[Brodie84] Brodie, M., Mylopoulos, J. and Schmidt, J., (eds.) *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, Springer-Verlag, 1984.

[Brachman85] Brachman, R. and Schmolze, J., "An Overview of the KL-ONE Knowledge Representation System", *Cognitive Science 9*, 1985.

[Chen76] Chen, P. "The Entity-Relationship Model: Towards a Unified View of Data", *ACM Transactions on Database Systems 1(1)*, 1976.

[Codd70] Codd, E.F., "A Relational Model for Large Shared Data Banks," *Communications of the ACM 13,* No. 6, June 1970, 377-387.

[Codd79] Codd, E.F., "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems 4*, No. 4, December 1979.

[DAIDA91] Jarke, M., Mylopoulos, J., Schmidt, J. and Vassiliou, Y., "DAIDA: An Environment for Evolving Information Systems", to appear.

[Fikes85] Fikes, R. and Kehler, "The Role of Frame-Based Representations in Reasoning", *Communications of the ACM 28(9),* 1985.

[Findler79] Findler, N. V., (ed.), *Associative Networks: Representation and Use of Knowledge by Computers,* Academic Press, New York, 1979.

[Gallagher86] Gallagher, J. and Solomon, L., "CML Support System", SCS Technische Automation und Systeme GmbH, Hamburg, June 1986.

[Greenspan82] Greenspan, S., Mylopoulos, J. and Borgida, A., "Capturing More World Knowledge in the Requirements Specification", Proceedings International Conference on Software Engineering, Tokyo, 1982. Reprinted in Freeman, P. and Wasserman, A. (eds.) *Tutorial on Software Design techniques*, IEEE Computer Society Press, 1984.

[Greenspan84] Greenspan, S., *Requirements Modelling: A Knowledge Representation Approach to Requirements Definition,* Ph.D. thesis, Department of Computer Science, University of Toronto, 1984.

[Greenspan86] Greenspan, S., Borgida, A. and Mylopoulos, J., "A Requirements Modelling Language and Its Logic", in Brodie, M. and Mylopoulos, J., (eds.) *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, Springer-Verlag, 1986.

[Hammer81] Hammer, M. and McLeod, D., "Database Description with SDM: A Semantic Data Model", *ACM Transactions on Database Systems*, September 1981.

[Hulin89] Hulin, G., Pirotte, A., Roelants, D., and M. Vauclair, "Logic and Databases," in A. Thayse (ed.), *From Modal Logic to Deductive Databases - Introducing a Logic-Based Approach to Artificial Intelligence,* John Wiley & Sons Ltd, 1989.

[Hull87] Hull, R. and King, R., "Semantic Database Modelling: Survey, Applications and Research Issues", *ACM Computing Surveys 19(3),* September 1987.

[Constantopoulos91] Constantopoulos, P., Jarke, M., Mylopoulos, J. and Vassiliou, Y., "The Software Information Base: A Server for Reuse", (submitted for publication).

[Jarke88] Jarke, M. and Rose, T., "Managing Knowledge About Information System Evolution", Proceedings *ACM SIGMOD International Conference on Management of Data*, 1988.

[Koubarakis88] Koubarakis, M., *An Implementation of CML*, M.Sc. thesis, Department of Computer Science, University of Toronto, 1988.

[Koubarakis89] Koubarakis, M., Mylopoulos, J., Stanley, M. and Jarke, M., "Telos: Features and Formalization", KRR-TR-89-4, Department of Computer Science, University of Toronto, 1989.

[KR89] Proceedings of *First International Conference on Knowledge Representation*, Toronto, May 1989.

[Kramer80] Kramer, B., *The Representation of Procedures in the Procedural Semantic Network Formalism,* M.Sc. thesis, Department of Computer Science, University of Toronto, 1980.

[LOKI88] Binot, B., Demoen, B., Hanne, K-H., Solomon, L., Vassiliou, Y., "LOKI: A Logic-Oriented Approach to Data and Knowledge Bases Supporting Natural Language Interaction" Proceedings ESPRIT Technical Conference, Brussels, November 1988.

[Minsky68] Minsky, M., (ed.), *Semantic Information Processing,* MIT Press, Cambridge, MA, 1968.

[Mylopoulos80] Mylopoulos, J., Bernstein, P. and Wong, H., "A Language Facility for Designing Interactive Database-Intensive Applications", *ACM Transactions on Database Systems 5(2)*, June 1980.

[Mylopoulos88] Mylopoulos, J. and Brodie, M., (eds.) *Readings in Artificial Intelligence and Databases,* Morgan-Kaufmann, 1988.

[Mylopoulos90] Mylopoulos, J., Borgida, A., Jarke, M. and Koubarakis, M., "Telos: Representing Knowledge About Information Systems", *ACM Transactions on Information Systems*, October 1990.

[Mylopoulos90b] Mylopoulos, J., "Object-Orientation and Knowledge Representation" in Meersman, R. and Kent, W., (eds.) *Object-Oriented Databases: Analysis, Design and Construction,* North-Holand, 1991.

[Plexousakis90] Plexousakis, D., *The Semantics of Telos: A Language for Knowledge Representation, M.Sc. thesis*, Department of Computer Science, University of Toronto, 1990.

[Roman85] Roman, G-C., "A Taxonomy of Current Issues in Requirements Engineering" *IEEE Computer 18(4),* April 1985.

[Schmid74] Schmid, J. and Swenson, R., "On the Semantics of the Relational Data Model", Proceedings *ACM SIGMOD International Conference on Management of Data*, 1974.

[Stanley86] Stanley, M., *CML: A Knowledge Representation Language with Applications to Requirements Modelling,* M.Sc. thesis, Department of Computer Science, University of Toronto, 1986.

[Topaloglou89] Topaloglou, T. and Koubarakis, M., "An Implementation of Telos", TR-KRR-89-8, Department of Computer Science, University of Toronto.

[Vassiliou90] Vassiliou, Y., Marakakis, M., Katalagarianos, P., Chung, L., Mertikas, M.and Mylopoulos, J., "A Mapping Assistant for Generating Designs from Requirements", Proceedings the SEcond Nordic Conference on Advanced Information Systems Engineering, CAiSE'90, Stockholm, May 1990.

[Vilain89] Vilain, M., Kautz, H. and van Beek, P., "Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report", in Weld, D. and De Kleer, J., (eds.) *Readings in Qualitative Reasoning About Physical Systems,* Morgan Kaufmann, 1989.

[Webster87] Webster, D.E., "Mapping the Design Representation Terrain: A Survey", TR-STP-093-87, Microelectronics and Computer Corporation, Austin, 1987.