On Formal Requirements Modeling Languages: RML Revisited

Invited Plenary Talk

Sol Greenspan[1] John Mylopoulos[2] Alex Borgida[3]

Abstract

Research issues related to requirements modeling are introduced and discussed through a review of the requirements modeling language RML, its peers and its successors from the time it was first proposed at the Sixth International Conference on Software Engineering (ICSE-6) to the present—ten ICSEs later. We note that the central theme of "Capturing More World Knowledge" in the original RML proposal is becoming increasingly important in Requirements Engineering. The paper highlights key ideas and research issues that have driven RML and its peers, evaluates them retrospectively in the context of experience and more recent developments, and points out significant remaining problems and directions for requirements modeling research.

1. Introduction

> "...Requirements definition is a careful assessment of the needs that a system is to fulfill. It must say why a system is needed, based on current and foreseen conditions, which may be internal operations or an external market. It must say what system features will serve and satisfy this context. And it must say how the system is to be constructed..."
>
> — Doug Ross [Ross77a]

Concern for Requirements Engineering is as old as Software Engineering itself. When awareness of a looming software crisis led to the formation of the field of Software Engineering in 1968, requirements analysis and definition practices were immediately under review as a potentially high-leverage but neglected area. By the mid-70s, this review had produced a wealth of empirical data, confirming that "the rumored 'requirements problems' are a reality" [Bell75]. The data suggested that requirements errors were the most numerous and, even more significantly, that they also were the most costly and time-consuming to correct. This recognition of the critical nature of requirements established Requirements Engineering as an important subfield of Software Engineering.

In response to this recognition came a wave of requirements concepts, languages, tools and methodologies (e.g., [TSE77], [Thay90]). However, while the first wave of requirements engineering research still unfolds, another wave is taking shape. This second wave is manifested in terms of a number of "firsts": the first International Symposium on Requirements Engineering [RE93], the first International Conference on Requirements Engineering [ICRE94], and the formation of the first IFIP Working Group on Requirements Engineering (IFIP WG 2.9). Moreover, this new wave is characterized by an expanded scope for and more ambitious demands on Requirements Engineering; we now see attempts to deal with more types of information and to understand, formalize and support more of the requirements definition and analysis tasks. At the core of these concerns are basic issues of representation and reasoning about the knowledge accumulated during the requirements acquisition phase, a subject referred to as *requirements modeling*. It is our contention that such representation and reasoning issues must continue to be addressed and that their resolution is a prerequisite to progress in all aspects of Requirements Engineering research and practice.

This paper reviews the trajectory of our research on requirements modeling from the language RML (first proposed in [Gree82]), through its successor Telos, to the present. Throughout, we highlight key ideas and research issues that have driven us and our colleagues, evaluating them retrospectively in the context of experience and more recent developments, and pointing out significant remaining research issues.

Section 2 of the paper presents a set of premises for requirements modeling, which motivated RML and continue to hold for other related proposals. Section 3 describes the basic features of RML, while Section 4 offers an overview of other languages that might be considered intellectual offsprings and peers of RML, describing how these have dealt with modeling issues left open by the original RML. Section 5 discusses some experiences using requirements modeling languages, and the issues of tools and methodologies that appear to be imperative, if this approach is to succeed. Section 6 draws some conclusions.

This paper might read like an opinionated tutorial to some and an implied research agenda to others. One thing the paper is not, however, is a survey of the field. Generally, the discussion concentrates on the work we are most familiar with, our own, with pointers to widely-known related research. For further work on the state of the art, the reader is encouraged to look at papers in sources such as [RE93, ICRE94, ESEC93, CAiSE93, IWSSD].

[1] Current address: GTE Laboratories Incorporated, 40 Sylvan Road, Waltham, MA, USA 02254.

[2] Current address: Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, Ontario, Canada M5S 1A4.

[3] Current address: Department of Computer Science, Hill Centre, Busch Campus, Rutgers University, New Brunswick, NJ, USA 08903.

2. Requirements Modeling

We propose to review some of the principles underlying the RML framework for requirements modeling, as reported in [Gree82] and later in [Gree84] and [Borg85a]. This gives us the opportunity to acknowledge intellectual debts, and at the same time present hindsights on RML.

• There is more to writing requirements than functional specifications. The stage preceding software design has been frequently called *specification*, and this term is often used as an abbreviation for "functional specifications"—a prescription of the desired functionality of a system to be built. This limited scope for requirements is evident in the use of the term "specification languages," applied to highly developed mathematical formalisms such as Z, VDM, Larch, OBJ (see [Wing90] for an introduction). Another important aspect of software specifications are *non-functional requirements* such as efficiency, security and privacy.

A specification is, by definition, prescriptive: it *specifies* desired properties for a system to be built. However, it has always been self-evident to many practitioners that the process of "systems analysis" preceding the development of software requires the analyst to achieve an understanding of the *application domain*, including the (e.g., organizational) environment within which the proposed system will eventually function. What remained largely unstated was that it is useful (dare we say, imperative) to capture *explicitly* as much of this understanding as possible, in order to support communication between the various "stakeholders" (customers, developers, testers)—which is the chief function of the requirements document. An explicit model is also of use in supporting continuity in face of inevitable staff turnover and other organizational change. With the understanding that *reuse* promises increased productivity, we now also see another purpose for explicitly capturing the understanding of the "environment" of the proposed software system—the requirements model can also be reused.

A model of a social organization or of the natural world is not likely to be prescriptive—"natural kinds" like 'chair' have no mathematical definitions. Hence we see the need to distinguish between *specification* languages, such as those mentioned above, and *modeling* languages, which aspire to offer facilities for the description of settings, or more precisely, *humans' knowledge/beliefs about these worlds*. This is a philosophical and psychological point which has profound implications for requirements language designers and users alike, as well as the requirements discourse.

The title of the original RML paper emphasized this distinction by talking about "capturing more world knowledge in the requirements specification". (It might have been better to use the term "requirements document/description" to make clear that existing as well as proposed things are to be discussed.) Among others, Ross [Ross77b] recognized the significance of this point when emphasizing that SADT was a notation for communicating *any* sort of information, not just software specifications; and Balzer *et al* repeatedly pointed to the need for closed models, which include the environment as well as the proposed system (e.g., in [Balz79]).

• Develop and present requirements as models. The appropriate stance to take in describing open-ended, real world phenomena is a "modeling" one: systematically identify *significant/relevant* aspects of the domain, and provide *direct representations* of them in the requirements. Models are the basis of understanding the world as well as for communicating among all involved parties. Requirements engineering activities are defined as model construction, management and analysis tasks.

The case for world modeling is articulated eloquently by Jackson [Jack78, Jack83], who starts with modeling of environmental processes prior to system design, a "model of reality with which [the system] is concerned."

• Object-centered knowledge representation is an appropriate foundation for conceptual modeling. The logical conclusion of the previous two points is that we should be developing *conceptual models.* The idea of world modeling, as offered by RML, is to capture (concrete or abstract) entities, activities, and other phenomena in the world as objects in a model. Moreover, objects are structured and organized according to principles of conceptual organization, such as "classes and instances," "parts and wholes," and "specializations and generalizations." Others, including Bubenko [Bube80] and Solvberg [Solv79], also advocated conceptual modeling for requirements modeling, or built requirements languages on top of knowledge representation substrata (e.g., GIST [Balz82]). The issue of conceptual modeling was considered by a number of participants at the 1980 Pingree Park Workshop [Ping81, Brod84], particularly by researchers working on data modeling for databases. Objects (with intrinsic identity) form the pearl-seeds around which knowledge about the domain is grouped. Of course, the field of Knowledge Representation (e.g., [Find79]) has a long-standing involvement with this subject matter, and has served for us as a rich source of ideas. More generally, the field of Cognitive Science is relevant to this enterprise (e.g., see [Coll88] for a collection of relevant papers).

• Abstraction and refinement, especially involving Is-A hierarchies, are significant in engineering large requirements. The importance of abstraction in dealing with many details has, of course, been a central tenet of Software Engineering all along. Subclass hierarchies were well-known in Simula/Smalltalk, in knowledge representation frameworks such as semantic networks, and in data modeling proposals such as [Smit77]. RML and its peers recognized the significance and utility of having objects in class hierarchies with inheritance (central pillars of what is now known as "object-orientation") and applying them to *all* aspects of software development [Mylo80, Gree83, Borg84]. In fact, we have argued elsewhere [Borg91] that a chief difference between knowledge representation and conceptual modeling is precisely this emphatic concern with abstraction and the engineering of large models.

• **Formal requirements modeling languages are needed.** The above principles defined a foundation for the RML proposal presented in [Gree84]. Implicit in this was the notion that some kind of *formal* language would be used to express requirements models. The advantage of any such formalism is that descriptions which adopt it can be assigned a well-defined semantics, often using methods imported from mathematical logic. The advantages of clear semantics include adjudicating among different interpretations of a given model, and offering a basis for various ways of *reasoning* with models, either through consistency checking (the foundation of useful tools) or by supporting question-answering or even simulation/prototyping.

Of course, the appeal and usability of some techniques may be largely due to their relative simplicity and flexibility derived from informality. We note that the use of a formal requirements modeling language does not preclude the concurrent use of informal notations. In fact, the original RML proposal envisioned early use of an informal notation, such as SADT, and a transformation process from an informal SADT model into a formal RML one.[4]

3. RML: Requirements Modeling Language

[Gree82] presented a framework that formed the basis for a requirements modeling language called RML. The language is further elaborated in [Gree84, Borg85a, Gree86]. This section summarizes its main features.

3.1 An Object-Centered Modeling Framework

RML views a model as consisting of objects of various kinds: individuals, or *tokens*, grouped into *classes*, which are in turn instances of *metaclasses*. Classes and metaclasses can have *definitional properties,* which specify what kinds of information can be associated to their instances through *factual properties*. For example, if the class `Person` has `name` as a definitional property, then each instance of `Person` can have a factual property associating a specific `name` to it. The requirement that every factual property must be induced by a corresponding definitional property is called the Property Induction Constraint, and offers a form of type checking. A subclass relationship between classes (and between metaclasses) asserts that every instance of the subclass is an instance of the superclass, and moreover, every definitional property of a class is a definitional property of its subclasses (i.e., inheritance).

The description of Figure 3.1 for the activity class named `Admit` is intended to convey the idea that the action of 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 first three properties of `Admit` identify properties that must be present for every instance of the class (**`participants`** properties). The next two properties

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

Figure 3.1

(`register` and `canAdmit`) are classified under **`parts`**, and specify sub-activities of `Admit`. 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.

The above framework is object-oriented[5], in line with early object-oriented programming languages or knowledge representation schemes like semantic networks and frames. However, the restriction to three levels and the relatively simple rules for property induction and inheritance were intended to assist in avoiding unnecessary complexity while providing an adequate expressive framework for requirements modeling.

3.2 An Ontology for Requirements Modeling

According to RML's view of the world (what we shall call its *ontology*), there are three types of things to be talked about, represented by instances of three built-in object categories, defined in terms of metaclasses: **Entity, Activity** and **Assertion**. The concepts of entity and activity were chosen because they are ubiquitous in modeling aspects of a real world, and match well corresponding concepts in other requirements modeling languages. Each object category is defined by specifying the property categories (kinds of definitional properties) that can be associated with those kinds of classes. For example, as we saw in Figure 3.1, activity classes have **`participants`**, **`parts`** and **`preconditions`** properties, among others.

Each object category is formalized in the semantics of RML in terms of axioms that capture its essence; for example, activities have axioms which state that their **`start`** time must precede their **`end`** time, or that all precondition properties must be true at the start of a new activity instance, while postconditions will be true at the end. The definition of "instance" is construed so that an activity token's instancehood in an activity class corresponds to the occurrence of the activity according to

[4] For further discussions about formality in Requirements Engineering see [Fick91].

[5] In the sense that building up a model consists of an iterative description of concepts and individuals with identity rather than an iterative statement of true facts or algorithms.

the formal properties associated with the class.

Just as for RML entities, RML activity classes are also organized into specialization/generalization hierarchies. Organizing activities in this way is a step beyond classical object-oriented software engineering approaches, in which objects (corresponding to RML entities) have attached procedures, but the procedures are themselves not subject to organization by hierarchies of classes. Some benefits and ramifications of this are discussed in [Borg80].

Assertion (formula) "objects" are the most novel part of RML. They provide a formal language for specifying otherwise informal information. Among their roles, they are associated as preconditions and postconditions on activities, and as invariants on entities. Treating assertions as objects makes them subject to the same structuring/organizing principles as other objects, but the meaning is specific to the logical nature of the assertions. For example, a class's **`argument`** properties are taken to be free variables of an open formula, while the induced factual argument properties are taken to be the values bound to those variable to close the formula. Furthermore, the semantics of 'instance' for assertion objects include the that the instance is to be interpreted against the class definition as a true statement while it is an instance. Other types of properties of assertions allow the structuring of assertions in terms of their parts, as for other object categories, but in this case parts are interpreted as logical conjuncts. The resulting representation is somewhat akin to, but semantically richer than, decision tree representations of complex formulas.

In short, RML supports objects of three general kinds (activities, entities and assertions) to be related to each other by binary semantic relationships, grouping these objects into classes, and organizing them according to specialization/generalization. Such a modeling framework lends itself to a methodology for building requirements models according to "*stepwise refinement by specialization,"* [Borg84], which develops class hierarchies in a regular and incremental manner.

3.3 Other Features

A formal semantics is given for RML by defining a mapping from RML descriptions into a set of assertions in First Order Predicate Calculus (hereafter FOPC) [Gree86]. These include all RML framework axioms as well as predicates and axioms associated with the specific classes defined by the modeler. Assertions translate into corresponding expressions in FOPC. However, the notation of FOPC provides no structuring/organization principles or other support for building and maintaining *large* theories (the essence of Software Engineering)—a defect intended to be addressed by RML, and its data modeling cousins.

The representation of time is essential for languages intended to model dynamic applications, if one is to prevent an implementation bias toward imperative programming style**.** RML assumes a linear, dense model of time points and encourages history-oriented modeling of an application, which consists of describing possible histories for an entity or activity (or assertion, for that matter). Accordingly, there is a time argument in every predicate appearing in an RML assertion. Moreover, time "objects" corresponding to time intervals are constructed as specializations of RML classes.

4. Language issues beyond RML

This section reviews extensions of requirements modeling languages over the past decade, discussing some of the research issues that have been raised or remain outstanding. We will do so by summarizing some of our own work over this time period, and pointing (in a regretfully cursory manner) to some significant work by others.

4.1 From RML to Telos

In a nutshell, RML offers a notation for requirements modeling which combines object-orientation and organization, with an assertional sublanguage used to specify constraints and deductive rules. Such a framework is shared by other proposals that tackle part of the requirements modeling problem [Webs87]. Unfortunately, if one is to take seriously the broad application scope of requirements modeling, RML and its peers suffer from a serious weakness. Its view of the world is *fixed***,** in the sense that the notions of entity, activity and assertion are built into the language. Indeed, the attribute categories associated with each one of these three notions are defined formally as part of the RML definition. What is needed to make a requirements modeling language more expressive is the ability to define new notions on par with those of entity, activity and assertion, thereby giving the modeler the ability to tailor the language to a particular class of applications.

On the basis of these observations, a revamping effort for RML was initiated in 1985, within the context of research projects LOKI and DAIDA funded by the European Community under the Esprit program. A language called CML (Conceptual Modeling Language) was an intermediate result of this activity, formalized in [Stan86] and further studied and cleaned up in [Koub88] and [Topa89]. The latest version of the language, obtained after several prototype implementations and some usage, is Telos [Mylo90].

Telos begins to address this problem of ontological extensibility by treating attributes/links in exactly the same way as entities/nodes. In particular, all attribute tokens are instances of attribute classes which, in turn, are instances of attribute metaclasses and so on. In addition, assertions can be associated with any Telos unit (entity or attribute) to declare constraints or deductive rules. These facilities combined make it possible to define *within* Telos (as attribute metaclasses) the property categories that defined the semantics of activities and entities in RML.

Figure 4.1 illustrates how this is accomplished with the definition of the metaclass `ActivityClass`, its instance `Admit` (analogous to the RML definition of `Admit` given in Section 3) and an instance of `Admit`, `AdmitMaria`. The reader should be aware that in many respects the discussion below simplifies the features of Telos, notably

```
CLASS ActivityClass IN MetaClass, Class WITH
  attribute
    participant: EntityClass
    part: ActivityClass
    precondition: AssertionClass
  integrityConstraint
    preCondHoldsBefore:
     ForAll p/Precondition,x/Token,t,t'/Time
      [x in from(p) at t ⇒
           Holds(to(p),t') ∧ t' overlap t]
    partsDuringWhole:
     ForAll p/Part,x/Token,t,t'/Time
      [x in from(p) ⇒Exists q/Attribute
        [q in p ∧ from(q)=x ∧to(p) during x]]

           deductiveRule
        $ (ForAll p/Patient
          [x ∈ p.room.ward ⇒ x ∈ p.loc]
END ActivityClass

CLASS Admit IN ActivityClass, Class WITH
      participant, single
        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

TOKEN AdmitMaria IN Admit WITH
      newPatient
           : Maria
      toWard
           : ChildrensW
      document
           : GetInfoFrom Maria
        ...
END AdmitMaria
```

Figure 4.1

its treatment of time.

According to its definition, `ActivityClass` is an instance of `MetaClass`, which is a built-in class having as instances all metaclasses, also `Class`, the class of all classes. Moreover, `ActivityClass` has five attribute metaclasses, three of which are instances of `Attribute` (a built-in meta-metaclass associated with `Class`) and two of which are instances of the built-in attribute meta-metaclass `IntegrityConstraint`. The first integrity constraint (`preCondHoldsBefore`) specifies that for every instance (`x`) of the source of a precondition attribute class (i.e., an instance of an activity class) the destination of the precondition attribute (that's the precondition assertion) must hold at the start of the activity instance. The second integrity constraint declares that every activity instance has one subactivity for each part attribute of its class and that the subactivity must take place during the activity instance.

`Admit` is defined as an instance of `ActivityClass`, with three `Participant` attribute classes, two `Part` attributes and a `Precondition` attribute. Its `Participant` attributes are also instances of `Single`, an attribute metaclass whose instances are single-valued attributes. Thus, attributes can be instances of several attribute classes, just like entities. Moreover, commonly-occurring constraints such as having a single-value or at-least-one-value can be defined once and for all in an attribute metaclass, and then used where appropriate through instantiation.

Figure 4.1 also shows an instance, `AdmitMaria`, of `Admit`. This token represents a particular admission to the hospital and has associated attributes which declare the new patient, the assigned ward and its subactivities. Note that the same mechanisms of instantiation and integrity constraint used to endow attribute metaclasses such as `preCondition` with a semantics are also used to endow attribute classes (or even attribute tokens, though not shown here) with an appropriate semantics.

4.2 Ontologies

If one accepts the premise that extensible ontologies are useful for requirements modeling, the obvious next question is: what are examples of such useful ontologies? The answer ranges from concepts that are of universal utility, such as “agents," to more domain-specific notions. We illustrate these with several examples.

Agents: In dealing with most applications, one encounters several interacting entities, processes, etc. that are trying to achieve differing goals. The importance of recognizing the notion of agents, especially for situations involving concurrent actions, has a long tradition in requirements modeling, beginning with the work of Feather [Feat87], and continuing to such recent proposals as [Dard91] and [Hage93].

Goals: A quick review of requirements frameworks, including SADT and data flow diagrams, reveals that they offer no specific help for the analysts to capture “...why a system is needed, based on current and foreseen conditions, which may be internal operations or an external market...,” a task mentioned in Ross' classic account of requirements analysis. To capture the reasons for a system, one needs to understand and model *intentional relationships*, such as organizational goals (e.g., cutting costs, improving quality), dependencies among agents (the manager depends on her boss to provide her with the necessary budget for a project and on her engineers to complete their assigned tasks on time) and non-functional requirements (such as wanting an inexpensive solution) and how they relate to organizational goals (say, cutting expenses).

[Yu93] explores an ontology for capturing intentional relationships within an organization. The ontology includes notions such as *actor, goal-, task-* and *resource-dependency*,

role and *position*. Using it, one can create organizational models which do provide answers to questions such as "why does the manager need the project budget?". Such models can serve as starting points in the analysis of an organizational setting, which precedes the adoption of a solution and the subsequent development of a software system. A different application of the same framework in the design of software processes is detailed in [Yu94].

Similarly, Greenspan et. al. [Gree93a] has proposed a specific ontology for a class of models, those capturing requirements information for service-oriented systems. A service-providing enterprise is modeled from four viewpoints:

- services that meet goals or address the needs of the customers;
- work flows or processes performed by the enterprise to provide the services;
- organizational units that serve as loci of responsibility for the work;
- systems that provide the capabilities and resources for performing the work.

As in Yu's work, this raises modeling and analysis questions in terms of responsibilities, resource dependencies, roles and positions.

Non-functional requirements: To deal with non-functional requirements, [Mylo92] proposes a framework which offers an ontology of *goals*, *methods* and *goal dependencies*. This ontology, based on ideas proposed by [Pott88] and truth maintenance systems before them, can be used to represent non-functional requirements of various types (security, performance etc.) in terms of goals which depend, either positively or negatively, on other goals and particular design decisions during system development. For example, the goal of having a secure database may depend positively on subgoals such as "minimize the number of people who can access the database" and negatively on the goal "offer a user-friendly interface". A goal is *satisficed* if it depends positively on other satisficed goals and does not depend negatively on any other satisficed goals. The proposed framework includes methods for goal decomposition and satisficing. These methods are meant to be domain-specific in the sense that there will be different methods for decomposing security goals as opposed to user-friendliness ones. The framework is explored in detail for performance and security requirements in [Nixo93] and [Chun93a].

Software development domain: More specialized work has also been carried out in the modeling and representation of the process of software development. An implementation of Telos, called ConceptBase, was used to represent requirements, design, implementations along with design rationale, software processes used and other relevant information about an information system development project within the context of the DAIDA Project [Jark92]. In the context of the ITHACA Project [Cons94], initiated in 1989, Telos was used in a *software-reuse information base* to organize descriptions of code, requirements, design specifications, run-time data, bug reports, and the like for software developed using different methodologies, tools and programming languages. Using the features of Telos, the designers of the software information base were able to define a number of associations among software descriptions that serve as basis for structuring the software information base, including new kinds of relationships such as a form of similarity and correspondence.

Other work by Jarke and colleagues has considered the use of ConceptBase extended by appropriate links to model development in-the-large and in-the-many (e.g., [Jark88]) and to organize software repositories.

Time: Requirements modeling languages appear to generally support some form of historical perspective on the world model. Some languages (e.g., GIST) take a state-transition view of history, while others (e.g., RML, Telos) incorporate an explicit notion of *time* in the language; RML and Telos experimented with an interval-based ontology for time, while ERAE [Dubo86] and its successors have chosen to represent temporal information using temporal logic operators, which appear to be much less verbose in some situations than Telos.

The above results, and other proposals for new, more expressive languages for requirements modeling (e.g., [Dard93]} suggest that future requirements modeling frameworks will offer a richer ontology than the basic entity-activity diet of the past.

4.3 Abstraction

One of the intended contributions of RML was the explicit introduction of so-called *abstraction principles* to help organize the considerable mass of details that belong in a requirements model. We briefly consider a variety of abstraction techniques as a way to review relevant research and to suggest possible new directions.

Generalization has been particularly favored by our own research, subclass hierarchies with inheritance playing a central role [Borg84, Borg88] as an organizing principle for the contents of conceptual models. It is interesting to note that in requirements modeling, the placement of classes/concepts into subclass hierarchies is left entirely up to the human developers. In contrast, researchers in knowledge representation, led by Brachman, and more recently information systems (e.g., review in [Borg92a]), have found it useful to allow the computer system to be charged with *self-organizing concept definitions* (e.g., "patients with age under 64" is a subclass of "patients with age under 70" and is disjoint from "persons with age over 72"). It still remains to demonstrate the benefits of such description languages for requirements modeling.

Classification received considerable attention in Telos, where the semantics and extensibility of the language were built into the metaclasses. (See Section 4.1).

Exceptions (special, extraordinary circumstances) abound in any human enterprise, and considerably complicate the understanding of a situations, especially at the beginning. This has led us to advocate [Borg85b] a *normal-case first abstraction,* where only the common/usual states and events in the domain are modeled first, and then

in successive pass(es), the special/exceptional situations and how they are handled are added. This is particularly successful if (i) there is some *systematic* way to find the abnormal cases, and (ii) there is a way to specify the exceptional circumstances as footnotes/annotations that do not interfere with the first reading.

Similarly, it is not uncommon to find generalization leading to over-abstraction (e.g., "all patients are assigned to rooms"), so that a subclass may contradict some aspect of one of its ancestors (e.g., "emergency-room patients may be kept on stretchers in hallways"). In [Borg88], we analyze the conflicting desiderata for subclass hierarchies that allow such 'improper specialization', and then suggest a simple language facility to accommodate them, while maintaining the advantages of inheritance, and even subtyping.

Note however that the above papers deal with the issue of exceptions only at the level of (database) programming languages, albeit ones supporting conceptual modeling. The issue of exceptions in specifications has however been considered in [Fink93] and [Scho93]; it seems interesting to contrast and perhaps combine these approaches.

Parameterization and ***modularization*** are abstraction techniques that have been used with great success in programming and formal specification techniques such as OBJ and Z, beginning with [Burs77]. Among requirements modeling languages, ERAE [Dubo92] supports parameterization to enhance the reusability of requirements. In a different direction, during program specification [Barr82, Borg84], we have found particularly useful the grouping of events and assertions into *scripts* that represent long-term patterns of conditioned actions and dependencies (e.g., "a patient is admitted, repeatedly treated, then discharged"). Scripts, based on the work of Zisman [Zism78] on production systems with Petri nets, appear to be related to "object histories" advocated by some object-oriented approaches.

Viewpoints: A problem inherent in the task of requirements elicitation and modeling is that there are often differences in opinion, approaches, etc. among stakeholders. As in databases, the modeling of views and their relationships has emerged as a significant research issue, with the work of Finkelstein *et al* [Fink92] leading the way in Requirements Engineering. Perhaps not surprisingly, the various techniques above are not independent, and have additional uses: scripts are a natural place to describe the handling of special, exceptional circumstances, while view-points provide a mechanism for dealing with inconsistency in specifications [Fink93, Nuse93].

4.4 Formal Reasoning

Logic provides a paradigmatic case of how one can assign formal semantics to a formal language, and what kinds of tools one can build on top of this. A considerable number of requirements engineering languages in fact have an underlying formal deductive logic. RML itself was given meaning by translation to FOPC [Gree86]. Others build directly on modal logics of actions (e.g., [Kent93]) or temporal logics (e.g., see [Ghez93] for review).

One way to achieve brevity of expression in requirements is by having what are known as *defaults*, deductions that are made in the absence of other information. For example, in specifying an activity, it is widely useful to be able to describe only what has changed, leaving it implicit that "everything else stays the same". These kinds of defaults, known as *frame axioms* in the AI literature, become essential when dealing with object-oriented specifications with inheritance: if we wish to be able to specialize an activity so that the specialized version does additional things, then the more general activity cannot assert that it does all and only what it has been stated to do. In [Borg93] we examine a number of alternative approaches to this, and propose a new technique that appears to have simple yet solid foundations. Defaults can also be used to approach the thorny problem of reasoning in the presence of inconsistencies. Work reported in [Ryan93, Scho93, Fink93] shows how these and other issues may be addressed by including default reasoning in requirements modeling. We remark that defaults may also play a role when trying to "animate" requirements as a way of playing out what-if scenarios where insufficient information has been presented at the beginning.

There appear to be several basic approaches to the formalization of requirements languages; one is to adopt as foundation some advanced logic (e.g., default, modal, deontic logic) or other mathematical formalism (e.g., order-sorted algebras, rewrite systems); the other is to try to stay within the framework of standard First Order Predicate Calculus. Hoping to inherit the benefits of the well-understood semantics and well-studied proof techniques of FOPC, we have tended to follow the second approach. For example, both the exceptional subclasses and the technique for dealing with the frame problem mentioned above are explicated in terms of FOPC (rather than default logic, whose semantics are less settled). The utility of using FOPC is also argued by Zave and Jackson [Zave93], who advocate the use of unadorned FOPC as a lingua-franca for *combining* multiple languages.

The field of Requirements Engineering, like knowledge representation, must eventually come to terms with the computational intractability (even undecidability) of reasoning with most expressive logical formalisms: if we are to have useful tools, we cannot allow them to unexpectedly go off into 'trances'. One approach is to severely stylize and limit the kind of reasoning we are prepared to support, as for example, in the *advanced type checking* of modern programming and specification languages. Another alternative would be to dip into the research on limited and *approximate reasoning* looking for relevant ideas. Or one can aim for more *qualitative* forms of reasoning, as in [Chun93b] on the satisficing of non-functional requirements. In all of these cases, researchers should however be alert to the possibility that the more restrictive circumstances or goals of requirements modeling may considerably simplify the solutions that are being borrowed from fields like knowledge representation.

4.5 Other Languages

RML is not unique in its premises or its features. Many other formal requirements modeling languages, some mentioned above, have been proposed over the same period as RML and its direct descendants.

The Conceptual Information Model, CIM [Bube80] is perhaps the first comprehensive proposal for formal requirements modeling language. Its features include an ontology of entities and events, an assertional sublanguage for specifying constraints, including complex temporal ones.

The GIST specification language [Balz82], developed at ISI over the same period as Taxis/RML, was also based on ideas from knowledge representation and supported modeling the environment; it was influenced by the notion of making the specification executable, and by the desire to support transformational implementation. It has formed the basis of an active research group on the problems of requirements description and elicitation (e.g., [John92]).

ERAE [Dubo86] [Dubo92] was one of the early efforts that explicitly shared with RML the view that requirements modeling is a knowledge representation activity, and had a base in semantic networks and logic.

The KAOS project constitutes another significant research effort which strives to develop a comprehensive framework for requirements modeling and requirements acquisition methodologies [Dard93]. The language offered for requirements modeling provides facilities for modeling goals, agents, alternatives, events, actions, existence modalities, agent responsibility and other concepts. Moreover, like Telos, KAOS relies on a meta-model to provide a self-descriptive and extensible modeling framework.

5. Requirements Modeling Experience

A requirements modeling language alone does not ensure that modelers can create good models. Given the language, one needs at least the following in addition: a framework (metamodel) reflecting an appropriate ontology for the domain; a methodology for elicitation and acquisition of models; analysis methods that help answer questions and find and resolve issues/problems; tool support to help with the above. While it is beyond the scope of this paper to elaborate on these issues in general, we will report here on some experience with RML and its successors, mentioning some relevant other work in passing.

5.1 Implementations

Many of the above-mentioned requirements modeling languages have been implemented, in the sense that there are knowledge-base management systems that can "reason" about the models represented in these languages in order to perform tasks such as consistency checking, question answering, or inferring propositions that had not been explicitly asserted.

For example, the implementation of Telos relies heavily on results from deductive databases, both for query processing (complicated by the presence of Horn clause-like deductive rules) and for constraint enforcement. Temporal reasoning is handled through a special-purpose inference engine based on efficient algorithms for temporal reasoning [Vila89], extended through a number of heuristics.

A subset of Telos was implemented at the University of Crete, using C++, and has been tested as part of the above mentioned ITHACA project to manage software repositories containing hundreds of thousands of software object descriptions.

Three independent Prolog-based implementations of Telos have been developed at SCS (Hamburg) [Gall86], the University of Passau [Jark88] and the University of Crete [Vassiliou90] and are in use at several sites. The Passau implementation, named ConceptBase, has been the most complete of these and the one that has seen most use. On the basis of positive experiences from this work, ConceptBase was adopted in a number of projects, including the NATURE project [Jark93a], which uses it as an integration platform for all components of a comprehensive *requirements engineering environment*..

We believe that part of the success of Telos and its implementations lies in the ability of the language to be extended to new ontologies and domains; among others, Telos has been extended to deal with agents, plans, goals, similarities, etc. through the addition of suitable metaclasses. However, often an unfortunate side-effect of meta-extensions is relatively poor performance, in comparison with special-purpose implementations. It remains an open problem whether there is a way to achieve some extensibility without sacrificing efficiency (as attempted in [Borg92], for example).

ACME [Gree91] constitutes another effort to implement and exploit the RML framework. ACME was originally conceived as an implementation of RML on a commercial knowledge representation system (Intellicorp's KEE™) but evolved over time on the basis of implementation experiences and practice. In order to offer some flexibility in providing different modeling frameworks for different application domains and different analysis tasks, the object-oriented conceptual modeling constructs (objects, properties, classes, metaclasses) have been separated from the RML ontology of entities, activities and assertions. The former appears in ACME as the Conceptual Modeling Platform (CMP), and the framework of choice, e.g., RML or any other, is built separately on top of the CMP. RML property categories are implemented in a manner analogous to their treatment in Telos, i.e., as attribute metaclasses, having definitional properties as instances. Thus, one defines classes of definitional properties by associating properties and behavior to attribute metaclasses. We have implemented some of the SOS framework on top of ACME [Gree93a]; experience is reported below.

5.2 Experience with ACME

We mention here some substantial experience gained using ACME, which has been used to model requirements for the purposes of a business process re-engineering (BPR) effort. Models consist of one to two thousand concepts

(e.g., workflows, process steps, actions, data entities), translating to between ten and twenty thousand ACME objects (where everything, including properties, are considered to be objects). The ACME tool builders interacted with the designers of the BPR methodology in order to acquire their framework (metamodel) into ACME. Then, subject matter experts, namely experts in the business process being re-engineered (which happened to be trouble reporting and repair), used the framework and the tool to acquire and analyze their models. The framework included a simple assertion language for stating initiation conditions on actions.

There were several lessons learned, all of them anecdotal and none of them conclusive. However, we view it as important to share such experiences, because, as stated in [Luba93], little is actually known about how organizations do requirements. We attempt to evaluate our experience against the slogans discussed above in Section 2.

Use explicit world modeling: In this domain, world modeling constituted a major portion of the requirements engineering activities. Modeling the business processes was the most time-consuming and, arguably, the most important activity. Once the models took shape, it was relatively straightforward to state requirements, such as constraints, policies, and non-functional properties. Such requirements are statements *about* the world, and once the world was understood (i.e., adequately modeled), a large portion of the requirements work had been done. The converse is also true: in the absence of an explicit model, we surmise that the statement of requirements would have been very difficult or impossible to create and comprehend. It required considerable commitment to use a modeling approach and to involve people at all levels.

Object-oriented conceptual modeling: All in all, the use of the structuring mechanisms of an object-oriented framework were useful to modelers, although, to paraphrase the title of a panel at RE93 last year [Pott93] they might not have known their requirements were object-oriented unless they asked their analyst. From the modeler's viewpoint, the units of description were entities, processes, actions, trigger conditions, and so on. The fact that all units of description were treated uniformly as objects by ACME was not necessarily relevant during modeling. However, from the point of view of ACME the tool, the implementation relies heavily on object-orientedness.

Use abstraction techniques: Many discussions of concepts centered on distinctions that can be explained by abstraction techniques such as specialization, part-of, similarity, exception, and the like. This conceptual modeling vocabulary was obviously useful. However, there was no attempt to use stepwise refinement by decomposition or specialization as an overall process principle or model organization mechanism. The concept of a workflow task did not even have a notion of decomposition into subtasks; elaboration/refinement of the models was done by replacing nodes/links in a "flat" model by other nodes and links. The lack of decomposition might actually have been responsible for some efficiencies in the modeling process, since no one had to present or defend arbitrary groupings of concepts into another, or the invention of concepts that contain others. However, despite the simplicity and comfort of the flat approach, they anticipated a grave disadvantage coming later on in the use of the modeling methodology: there would be little chance for reuse due to the failure to create reusable abstractions in the first place.

Formality: Formal semantics are needed to allow for automated analysis, but the more complex the language gets, the harder it is to use. Our initial attempts at a rich logic in ACME made the language hard to use: users could not easily create or understand assertions in a model. However, a simple/weak language was worse than English, which at least recorded informally the intended meaning of the expression. We compromised by using a relatively simple but formal language (e.g., no quantifiers, and special term constructors only as needed). Novices could specify things and get some inconsistency checking done within ACME. ACME can parse and execute the expressions. In the tradeoff between formality and complexity, we agree with [Zave91] that formality should not be avoided but rather strategies for coping with the concomitant complexity should be pursued, for example, using multiple viewpoints or other means to form simpler projections of the information.

Other reports of experiences with using requirements languages include [Hage93] and [Jarke93b].

5.3 Other Support for Requirements Process

Our experience motivates us not only to improve the language foundations but also the methodological and tool support for the requirements process. The main difficulties of requirements engineering concern negotiating a common understanding of concepts, dealing with ambiguity, and clarifying desires, needs and constraints [Gaus89]. Because these topics have to do mainly with human understanding and communication, they are particularly difficult to make amenable to rigorous or formal treatment. For example, it needs to be recognized that most of the requirements process is spent not in the possession of correct and consistent models, but rather, as indicated by Feather [Feat91], in "Getting Right From Wrong." That is to say, requirements modeling consists of a series of incremental steps that (hopefully) converge in a model with the appropriate content.

In this vein, a useful approach is that offered by Reubenstein [Reub91], who gave a precise meaning to several types of "issues" that arise while developing a requirements model, such as inconsistencies, ambiguities and incompleteness. These issues were detected according to formal rules corresponding to the intuition behind these terms. Given a rather general frame-based representation scheme with an embedded propositional expression language, Reubenstein's Requirements Apprentice recognized the presence of these issues and kept an agenda of issues to be resolved.

Keeping track of assumptions and rationale [Gree93b] is

another aspect of requirements modeling that needs attention and is being addressed by various researchers, e.g., [Rame92].

Another important aspect of requirements modeling is decision-making to achieve requirements models that satisfy design goals, resolve conflicting requirements, predicting failure, and so on. While constructing a requirements model, one is concerned with critiquing, parallel elaboration, etc. This requires combining the representation of specialized domain knowledge with problem-solving techniques, such as planning and searching, as in [Ande93, Robi94, Fick88].

The KAOS methodology [Dard93] exploits the expressiveness of the KAOS modeling language to support all phases of requirements acquisition, starting with initial goals (both functional and non-functional), proceeding with the identification of potential agents who could take responsibility for the satisfaction of these goals, all the way to the assignment of actions to particular agents, including computer systems to be built.

Finally, a number of other methodology and tool issues, such as presentations, transformations, visualization, connection to hypertext are addressed by others, e.g., [John92], [Lali93], [Jark93b].

6. Discussion and Conclusions

The paper has presented a retrospective on RML, its premises, main features, evolution over the past decade, experiences in use and its peers among requirements modeling languages. We have stuck mostly to representation and reasoning issues, with some reports of implementations, and allusion to tools and methodology as important related work.

Like any other researcher in a similar situation, we are pleased to see some of the premises and features of RML and its peers being used in systems analysis practice through the methodology known as Object-Oriented Analysis (OOA)—for example, [Schl88, Coad90, Rumb91, Jaco92, Booc93, Wirf90], to name a few. The basic tenets of OOA include an object-oriented representation framework augmented with a simple ontology and a graphical notation for describing portions of a model. Some OOA proposals go further by adding facilities for representing temporal, cardinality and other constraints, while others attempt to endow their notation with a formal semantics. In so doing, they have to address some of the same issues faced by formal requirements modeling languages since 1980, including issues mentioned in this paper.

Surprisingly, much of the work on OOA has been done independently of earlier work within the Requirements Engineering community[6]. Perhaps this practice can be reviewed. Requirements Engineering has a head start on the development of formal (and object-oriented) requirements modeling languages, tools and methodologies and has much to offer. OOA, on the other hand, clearly has been accumulating a wealth of practical experience that can serve as basis for more directed and directly applicable future research in Requirements Engineering.

Language design has been a central theme in Software Engineering research throughout its history. The emergence of formal requirements modeling languages and OOA techniques is the logical next step in providing linguistic, methodological and tool support for the early phases of the software lifecycle, very much for the reasons first articulated in [Bell75]. However, requirements modeling languages, because of their subject matter, are fundamentally different from programming and specification languages whose subject matter (software systems) is man-made, bounded and objectively known. A corollary of this, argued in the original RML paper as well, is that designers of requirements modeling languages need to turn to research in areas other than core computer systems and programming languages (areas such as knowledge representation), in search of ideas and research results that serve as basis for the design of their languages. To put it another way, it is unwise to try to design requirements modeling languages by merely adopting programming language ideas.

Requirements engineering is just one of several tasks a computer professional may be called on to perform that requires modeling aspects of the real world. Data modeling for database design, process modeling for software and business process engineering, and knowledge engineering for expert system development are others. Moreover, the demand for such world modeling skills for the computer professional is growing, as we find that software systems need to be conceived right from the start as embedded systems in a complex, evolving organizational setting. Unfortunately, this skill is being given very little attention in the standard undergraduate computer science curriculum. How well do our graduating computer science students know FOPC and its use in representing facts about an application? Is the relationship between

`AirCanadaFlight#23#from#NYC#to#Toronto`

and `AirlineFlights` like the relationship between an instance and a concept, or like the relationship between a concept and a superconcept? What about the relationship between `AirCanadaFlight#23` and the flight leaving tomorrow morning from NYC to Toronto? And what is the effect of making one choice vs. the other? How much practice have students had in building requirements or process models with formal tools, using different notations (compared with, say, how much training they receive in using different programming languages)? Where in their program of study do they learn about world modeling as a professional skill (worth learning in its own right and on par with system modeling) which comes with a theory and an engineering practice? We believe the time has come to think of conceptual modeling as essential to our

[6] For example, [Embley92] presents an Entity-Relationship-like model for OOA, extended with notions for modeling actions, triggers and states, with a formal semantics defined in terms of a translation into FOPC, very much along the lines of RML. Throughout, there is no reference to the formal requirements modeling languages mentioned here.

undergraduate curriculum, to be taught to computer science undergraduates as a subject whose manifestations they will encounter in varied fields such as requirements analysis, database design, knowledge engineering and process modeling.

Acknowledgments.

We offer our heart-felt thanks to all our collaborators and co-authors over the years, as well as the members of the Requirements Engineering community. We also appreciate very much the comments, given often on extremely short notice, by the following people: Anthony Finkelstein, Axel van Lamsweerde, Matthias Jarke, Pamela Zave and Eric Yu.

Mark Feblowitz (GTE Labs) has done remarkable work on the design and implementation of ACME, from which valuable experience and insights have been gained.

John Mylopoulos has been supported in part by the National Science and Engineering Research Council of Canada, the Canadian Institute of Advanced Research, the Information Technology Research Centre of Ontario, also by the Institute of Robotics and Intelligent Systems.

Alex Borgida has been supported in part by NSF Grant IRI 91-19310.

References

[Ande93] Anderson, J. R., and B. Durney, B., "Using Scenarios in Deficiency-Driven Requirements Engineering," in [RE93].

[Balz79] R. Balzer, N. Goldman, "Principles of good software specifications and their implications for specification languages," *Proc. IEEE Conf. Spec. of Reliable Software,* pp. 58-67.

[Balz82] Balzer, R., N. Goldman, D. Wile, "Operational specifications as a basis for rapid prototyping," *Proc. Symp. Rapid Prototyping, ACM Soft. Eng. Notes, 7(5),* Dec. 1982, pp. 3-16.

[Barr82] Barron, J., "Dialogue and Process Design for Interactive Information Systems Using Taxis," *Proc. SIGOA*, June 1982, *ACM SIGOA Newsletter*, 3(1,2), pp. 12-20.

[Bell75] Bell, T. E., and Thayer, T. A., "Software Requirements: are they really a problem," Proc. 2nd Int. Conf. on Software Engineering, 1976, pp. 61-68.

[Booc93] Booch, G., Object-Oriented Design with Applications, Benjamin/Cummings, 2nd edition, 1993.

[Borg80] Borgida, A., and S. J. Greenspan, "Data and Activities: Exploiting Hierarchies of Classes," in [Ping81].

[Borg84] Borgida, A., Mylopoulos, J., and Wong, H. K. T., "Generalization/Specialization as the Basis for Software Specification," in [Brod84].

[Borg85a] 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.

[Borg85b] A. Borgida, "Language features for the flexible handling of exceptions in Information Systems," *ACM TODS 10(4)*, pp.565-603, December 1985.

[Borg88] A. Borgida, "Modeling class hierarchies with contradictions," *Proc. ACM SIGMOD Conf.* , pp.434-443, 1988.

[Borg91] A. Borgida, "Knowledge Representation, Semantic Modeling: Similarities and Differences", *E-R Approach'91,* Elsevier Publ., 1991.

[Borg92a] A. Borgida, "A New Look at Description Logics, and Their Utility in the Management of Information," Dept. of Computer Science, Rutgers University, June 1993. (revised version of DCS-TR-295, June 1992).

[Borg92b] Borgida, A. and R. Brachman, "Customizable Classification Inference in the ProtoDL Description Management System", *Proc. 1st Conf. Information and Knowledge Management*, Baltimore, MD, November 1992

[Borg93] Borgida, A., Mylopoulos, J. and Reiter, R., "...And nothing else changes: The frame problem in procedure specifications," *Proc. 15th Int. Conf. on Software Engineering*, Baltimore, 1993.

[Brod84] Brodie, M., Mylopoulos, J. and Schmidt, J., (eds.) *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, Springer-Verlag, 1984.

[Bube80] Bubenko, J., "Information Modeling in the Context of System Development," in *Proc IFIP 80*, pp. 395-411, 1980.

[Burs77] Burstall, R., J. Goguen, "Putting theories together to make specifications," *Proc. IJCAI'77*, pp.1045-1052.

[CAiSE93] *Proc. 5th Int. Conf. Advanced Information Systems Engineering,* Paris, June 1993, LNCS 685, Springer Verlag.

[Chun93a] Chung, L., "Dealing with security requirements during the development of information systems," in [CAiSE93].

[Chun93b] Chung, L. *Representing and Using Non-Functional Requirements: A Process-Oriented Approach,* Ph.D. thesis, Dept. of Comp. Sci,., U. of Toronto, 1993.

[Coad90] Coad, P., and Yourdon, E., *Object-Oriented Analysis*, Yourdon Press, Englewood Cliffs, NJ, 1990.

[Coll88] Collins, A., and Smith, E., *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, Morgan-Kaufmann, 1988.

[Conk88] Conklin, J., and Begeman, M., "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *Trans. on Office Info. Systems, 6(4),* Oct. 1988, pp. 281-318.

[Cons94] Constantopoulos, P., Jarke, M., Mylopoulos, J. and Vassiliou, Y., "The Software Information Base: A Server for Reuse," *VLDB Journal* (to appear).

[Dard91] Dardenne, A., S. Fickas, and A. van Lamsweerde, "Goal–Directed Concept Acquisition in Requirements Elicitation," in *Proc. of the Sixth International Workshop on Software Specification and Design*, October, 1991.

[Dard93] Dardenne, A., S. Fickas, and A. van Lamsweerde, "Goal–Directed Requirements Acquisition," in *Science of Computer Programming, 20*, 1993, pp. 3-50.

[Dubo86] Dubois, E., Hagelstein, J., Lahou, E., Ponsaert, F. and Rifaut, A., "A knowledge representation language for requirements engineering," *Proc. IEEE 74(10)*, 1986.

[Dubo92] Dubois, E., Du Bois, P. and Rifaut, A., "Elaborating, Structuring and Expressing Formal Requirements for Composite Systems," Proc. Int. Conf. on Adv. Info. Systems Eng.. (CAiSE-92), Manchester, 1993.

[Dubo94] Dubois, E., Du Bois, P., DuBru, F., "Animating Formal Requirements Specifications of Cooperative Information Systems," *Proc. 2nd Int. Conf. on Cooperative Information Systems*, Toronto, 1994.

[Embley92] Embley, D., Kurtz, B., Woodfield, S., *Object-Oriented Systems Analysis,* Yourdon Press, Prentice-Hall, 1992.

[ESEC93] *Proc. of the 3rd European Software Engineering Conference,* Milan, Italy, Springer-Verlag, 1993.

[Feat87] Feather, M. “Language support for the specification and derivation of concurrent systems," *ACM Trans. on Prog. Lang*. 9(2), April 1987, pp. 198-234.

[Feat91], Feather, M., “Requirements Engineering: Getting Right From Wrong,” in [ESEC93], pp. 485-488.

[Fick88] Fickas, S., Nagarajan, P., “Critiquing Software Specifications: a Knowledge-Based Approach,” in *IEEE Software*, Nov. 1988.

[Fick91] Fickas, S., Position papers for panel on “Neats vs. Scruffies,” S. Fickas, organizer, in [ESEC93].

[Find79] Findler, N. V., (ed.), *Associative Networks: Representation and Use of Knowledge by Computers,* Academic Press, New York, 1979.

[Fink92] A. Finkelstein, J. Kramer, *et al.*, "Viewpoints: A Framework for Multiple Perspectives in System Development", *Int. Journal of Soft. Engineering and Knowledge Eng., 2(1),* World Scientific Publishing, pp. 31-57, March 1992.

[Fink93] A. Finkelstein, D. Gabbay, *et al* , “Inconsistency handling in multi-perspective specifications”, in [ESEC93], pp. 84-99.

[Gall86] Gallagher, J. and Solomon, L., "CML Support System," SCS Technische Automation und Systeme GmbH, Hamburg, June 1986.

[Gaus89] Gause, D., C., and Weinberg, G. M., *Exploring Requirements: Quality Before Design*, Dorset House, 1989.

[Gree82] Greenspan, S., Mylopoulos, J. and Borgida, A., "Capturing More World Knowledge in the Requirements Specification," *Proc. 6th Int. Conf. on SE*, Tokyo, 1982. Reprinted in Freeman, P., and Wasserman, A. (eds.) *Tutorial on Software Design Techniques*, IEEE Computer Society Press, 1984. Also in R. Prieto-Diaz and G. Arango, *Domain Analysis and Software Systems Modeling*, IEEE Comp. Sci. Press, 1991.

[Gree83] Greenspan, S., and J. Mylopoulos, "A Knowledge Representation Approach to Software Engineering: The Taxis Project," *Proc. Conf. Canadian Info. Processing Society*, Ottawa, Ontario, May 1983, pp. 163-174.

[Gree84] Greenspan, S., Requirements Modeling: A Knowledge Representation Approach to Requirements Definition, Ph.D. thesis, Department of Computer Science, University of Toronto, 1984.

[Gree86] Greenspan, S., A. Borgida, and J. Mylopoulos, "A Requirements Modeling Language and Its Logic," *Information Systems, 11(1),* pp. 9-23, 1986. Also appears in *Knowledge Base Management Systems*, M. Brodie and J. Mylopoulos, Eds., Springer-Verlag, 1986.

[Gree91] Greenspan, S., M. Feblowitz, C. Shekaran, and J. Tremlett, "Addressing Requirements Issues Within a Conceptual Modeling Environment," *Proc. of the 6th Int. Workshop on Soft. Spec. and Design*, October, 1991.

[Gree93a] Greenspan, S., and M. Feblowitz, "Requirements Engineering Using the SOS Paradigm," in [RE93], pp. 260-265.

[Gree93b] Greenspan, S., Panel on Recording Assumptions and Rationale, in [RE93], pp. 282-285.

[Hage93] Hagelstein, J., D. Roelents, P. Wodon, “Formal requirements made practical," in [ESEC93], pp. 127-144.

[ICRE94] *Proc. IEEE International Conference on Requirements Engineering,* April 18-22, 1994.

[IWSSD] *Procs. 5th/6th/7th Int. Workshops on Software Specification and Design,* IEEE Computer Society Press, Tracks on Requirements Engineering, 1989/1991/1993.

[Jaco92] Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.

[Jack78] Jackson, M., *Proc. 2nd Int. Conf. on Software Engineering*, 1976, pp. 72-81.

[Jack83] Jackson, M., *System Development*, Prentice-Hall, 1983.

[Jark88] Jarke, M., Eherer, S., Gallersdoerfer, R., Jeusfeld, M., Staudt, M., “ConceptBase -- a deductive object manager,” Special Issue on Deductive and Object-Oriented Databases,” M. Kifer (ed), J. of Intelligent Information Systems (to appear).

[Jark92] Jarke, M., Mylopoulos, J., Schmidt, J. and Vassiliou, Y., "DAIDA: An Environment for Evolving Information Systems," *ACM Trans. on Info. Sys., 10, 1,* 1992, pp. 1-50.

[Jark93a] Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, Y. Vassiliou, “Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis,” in [RE93], pp. 19-33.

[Jark93b] Jarke, M., Jeusfeld, M., Szczurko, P.: "Three aspects of intelligent cooperation in the quality cycle." *Int. J. Intel. Coop. Information Systems 2(4)*, 1993.

[John92] Johnson, W.L., M. Feather and D. Harris, “Representing and presenting requirements knowledge," *IEEE Trans. on SE,* October 1992, pp. 853-869.

[Kent93] Kent, S., Maibaum T., and Quirk, W., “Formally Specifying Temporal Constraints and Error Recovery,” in [RE93], pp. 208-215.

[Koub88] Koubarakis, M., *An Implementation of CML,* M.Sc. thesis, Dept. Comp. Sci., Univ. of Toronto, 1988.

[Lali93] Lalioti, V. and Loucopoulos, P., ”Visualization for Validation”, in [CAiSE93], pp. 143-164.

[Luba93] Lubars, M., Potts, C., and Richter, C., “A Review of the State of the Practice in Requirements Modeling,” in [RE93], pp. 2-14.

[Mylo80] Mylopoulos, J., Bernstein, P. A. and Wong, H. K. T., "A Language Facility for Designing Data-Intensive Applications," *ACM Trans. on Database Systems 5(2)*, June 1980. Reprinted in Zdonik, S. and Maier, D., *Readings in Object-Oriented Database Systems,* Morgan-Kaufmann, 1989.

[Mylo90] Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M., "Telos: Representing Knowledge About Information Systems," *ACM Transactions on Information Systems*, October 1990.

[Mylo92] Mylopoulos, J., Chung, L., Nixon, B., “Representing and Using Non-Functional Requirements,” in [TSE92], pp. 483-497.

[Nixo93] Nixon, B., "Representing and Using Performance Requirements During the Development of Information Systems," in [RE93], pp. 42-49.

[Nuse93] B. Nuseibeh, Kramer, J., and Finkelstein, A., "Expressing the Relationships Between Multiple Views in Requirements Specification," *Proc. 15th Int. Conf. on SE,* pp. 187-196, Baltimore, MD, May 1993, IEEE CS Press.

[Ping81] Brodie, M. and Zilles, S., (eds.) *Proc. of Workshop on Data Abstraction, Databases and Conceptual Modeling*, Pingree Park Colorado, Joint SIGART, SIGMOD, SIGPLAN Newsletter, January 1981.

[Pott88] Potts, C and Bruns, G., "Recording the Reasons for Design Decisions," Proceedings Tenth International Conference on Software Engineering, Singapore, 1988.

[Potts93] Potts, C., Organizer, Panel on “I never knew my requirements were object-oriented until I talked to my analyst,” in [RE93], pp. 226-230.

[Rame92] Ramesh, B., and V. Dhar, “Supporting Systems Development Using Knowledge Captured During Requirements Engineering,” in [TSE92].

[RE93] *Proc. IEEE Int. Symp. on Requirements Engineering*, IEEE Computer Society Press, January 1993.

[Reub91] Reubenstein, H., and Waters, R., “The Requirements Apprentice: Automated Assistance for Requirements Acquisition,” *IEEE Trans. on SE,* March 1991, pp. 226-240.

[Robi94] Robinson, W., and Fickas, S., “Supporting Multi-perspective Requirements Engineering,”in [ICRE94].

[Ross77a] Ross, D. T., and Schoman, “Structured Analysis for Requirements Definition,” in [TSE77], pp. 6-15.

[Ross77b] Ross, D. T., and Schoman, “Structured Analysis: A Language for Communicating Ideas,” in [TSE77], pp. 16-34.

[Rumb91] Rumbaugh, J., et al., Object-Oriented Modeling and Design, Prentice-Hall, 1991.

[Ryan93] Ryan, M., “Defaults in Specifications,” in [RE93], pp. 142-151.

[Schl88] Shlaer, S., and Mellor, S., *Object-Oriented Systems Analysis*, Yourdon, Englewood Cliffs, NJ, 1988.

[Scho93], Schoebbens, P. Y., “Exceptions in algebraic specifications, on the meaning of ‘but’," *Science of Computer Programming*, *20*, pp. 73-111, 1993.

[Smit77] Smith, J., and D.C.P. Smith, “Database Abstractions: Aggregation and Generalization,” *ACM Trans. on Database Systems, 2(2)*, Jun. 1977, pp. 105-133.

[Solv79] Solvberg, A., “A Contribution to the Definition of Concepts for Expressing Users’ Information System Requirements,” *Proc. Int. Conf. on E-R Approach to Systems Analysis and Design*,” Dec. 1979.

[Stan86] Stanley, M., *CML: A Knowledge Representation Language with Applications to Requirements Modeling,* M.Sc. thesis, Dept. Comp. Sci., Univ. of Toronto, 1986.

[Thay90] Thayer, R. and Dorfman, M., *System and Software Requirements Engineering*, (two volumes), IEEE Computer Society Press, 1990.

[Topa89] Topaloglou, T. and Koubarakis, M., "An Implementation of Telos," TR-KRR-89-8, Department of Computer Science, University of Toronto.

[TSE77] *IEEE Trans. on Software Engineering*, Special Issue on Requirements Analysis, SE-3, No. 1, Jan. 1977.

[TSE92] *IEEE Trans. on Software Engineering, 18(6) & 18(10),* Special Issue on Knowledge Representation and Reasoning in Software Development, June & October 1992.

[Vila89] 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.

[Webs87] Webster, D.E., "Mapping the Design Representation Terrain: A Survey," TR-STP-093-87, Microelectronics and Computer Corporation, Austin, 1987.

[Wirf90] Wirf-Brock, R., B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software,* Prentice-Hall, 1990.

[Wing90] Wing, J., “Specifiers Introduction to Formal Methods,” *IEEE Computer 23(9),* Sept. 1990, pp. 8-26.

[Yu93] Yu, E., "Modeling Organizations for Information Systems Requirements Engineering," in [RE93], pp. 34-41.

[Yu94] Yu, E. and Mylopoulos, J., "Understanding 'Why' in Software Process Modeling, Analysis and Design," *Proc. Sixteenth Int. Conf. on Soft. Eng.* , Sorrento, 1994.

[Zave91] Zave, P., “A Comparison of the Major Approaches to Software Specification and Design,” in [Thay90], p. 199.

[Zave93] P. Zave and M. Jackson, "Conjunction as Composition", Trans. on Software Engineering and Methodology, 2,4, ACM Press, Oct. 1993, pp. 379-411.

[Zism78] Zisman, M., “Use of Production Systems for Modeling Concurrent processes,” In D. A. Waterman and F. Hayes-Roth (Eds), Pattern-Directed Inference Systems, Academic Press, 1978, pp. 53-68.