Information Systems 23(3-4), June 1998.

Information Modeling in the Time of the Revolution[1]

John Mylopoulos[2]
University of Toronto

Abstract

Information modeling is concerned with the construction of computer-based symbol structures which capture the meaning of information and organize it in ways that make it understandable and useful to people. Given that information is becoming an ubiquitous, abundant and precious resource, its modeling is serving as a core technology for information systems engineering.

We present a brief history of information modeling techniques in Computer Science and briefly survey such techniques developed within Knowledge Representation (Artificial Intelligence), Data Modeling, (Databases), and Requirements Analysis (Software Engineering and Information Systems). We then offer a characterization of information modeling techniques which classifies them according to their *ontologies*, i.e., the type of application for which they are intended, the set of *abstraction mechanisms* (or, *structuring principles*) they support, as well as the *tools* they provide for building, analyzing, and managing application models. The final component of the paper uses the proposed characterization to assess particular information modeling techniques and draw conclusions about the advances that have been achieved in the field.

Keywords: Conceptual model, semantic data model, requirements model, knowledge representation language, ontology, abstraction mechanism, classification, generalization, aggregation, contextualization, materialization, normalization, parameterization, semantic network.

1. Introduction

"...The entity-relationship model adopts ... the natural view that the real world consists of entities and relationships... (The entity-relationship model) incorporates some of the important semantic information about the real world..."

Peter Chen [43]

We live through the Age of the Information Revolution. Thanks to advances in telecommunications, computer hardware and software, we are flooded with ever-growing amounts of information. The tremendous impact of the revolution to individuals and organizations alike is a daily news topic. One important

[1] This paper is based on a keynote address presented at the Ninth Conference on Advanced Information Systems Engineering (CAiSE'97) in Barcelona, Catalunya on June 17, 1997; an earlier version of the paper, titled "Characterizing Information Modeling Techniques for Information Systems Engineering", is included in Bernus, P., Mertins, K., and Schmidt, G., (eds.) *Handbook on Architectures of Information Systems*, Springer-Verlag, 1998 (to appear).
[2] Author's address: Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, Canada M5S 3H5; voice: 416-978-5180, fax: 416-978-1455, email: jm@cs.toronto.edu.

consequence of the revolution is that unlike ten years ago, most of the information available to us today is in computer-based forms, such as files and databases. The task for computer and information scientists is then to develop theories, tools and techniques for managing this information and making it useful.

Not surprisingly, traditional techniques for building information systems are no longer adequate. Firstly, there is relentless demand for new information services, such as cooperative query processing, browsing, similarity-based retrieval, data mining, data translation services, knowledge sharing, and the like. Secondly, there are increased expectations on information management techniques, including bottom up database schema construction, schema evolution, integration and co-existence of formatted, unformatted and hyperformatted data -- just to mention a few. Most importantly, information systems engineering needs to be extended to support new, flexible software architectures which make it possible to construct an information system from new as well as legacy data and software components.

Information modeling constitutes a cornerstone for any technique that claims to address these growing demands for more and better information services and management techniques. To use information, one needs to represent it, capturing its meaning and inherent structure. Such representations are important for communicating information between people, but also for building information systems which manage and exploit this information in the performance of useful tasks. Information modeling has been practiced within Computer Science since the first data processing systems in the '50s, using record and file structures to model and organize information. Since then, there has been a proliferation of proposals for information models, covering many different areas of Computer Science and Information Systems Engineering.

Information modeling plays a central role during information system development. [94] identifies four "worlds" that need to be understood and modelled during the development process. The *subject world* consists of the subject matter for an information 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 created the information system, the team of systems analysts and programmers involved, their adopted methodology and schedule, their design decisions and rationale. All of this information is relevant during the initial development of the system but also later on during operation and maintenance. Consequently, all of this information needs to be represented, somehow, in any attempt to offer a comprehensive framework for information systems engineering.

The purpose of this paper is to characterize information modeling practice and point to some directions for further research. Section 2 of the paper introduces basic definitions and fundamental premises underlying the field. Section 3 presents a brief (and admittedly biased) history of the field, and proposes a characterization of information models along three dimensions. Sections 4 to 6 discuss the space of alternatives for each dimension, while section 7 assesses particular information modeling techniques. Finally, section 8 summarizes the basic thesis of the paper and suggests directions for further research.

2. Preliminaries

Information modeling is concerned with the construction of computer-based symbol structures which model some part of the real world. We will refer to such symbol structures as *information bases*, generalizing the term from related terms in Computer Science, such as *database* and *knowledge base*. Moreover, we shall refer to the part of a real world being modeled by an information base as its *application*. The atoms out of which one constructs the information base are assumed to be *terms* which denote particular individuals in the application (Maria, George, 7, ...), or generic concepts under which the individual descriptions are classified (Student, Employee,...). Likewise, the associations within the information base denote real world relationships, such as physical proximity, social interaction, etc. The information base is queried and updated through special-purpose languages, analogously to the way databases are accessed and updated through query and data manipulation languages.

It should be noted that, in general, an information base will be developed over a long time period, accumulating details about the application, or changing to remain a faithful model of a changing application. In this regard, it should be thought of as a *repository* that contains *accumulated, disseminated, structured* information, much like human long-term memory, or databases, knowledge bases, etc. Assuming that information is entered through statements expressed in some language, the above considerations suggest that the contents of these statements need to be extracted and organized according to their subject matter. In other words, the organization of an information base should reflect its *contents*, not its *history*.

This implies some form of a *locality principle* [38, 125], which calls for information to be organized according to its subject matter. Encouragement for such a principle may come from the tools provided for building and updating an information base, but also from the methodology adopted for its use. For example, insertion operations which expect object descriptions (i.e., an object's name, attributes, superclasses etc.) do encourage this grouping. Insertion operations, on the other hand, which accept arbitrary statements about the application, for example "Maria wants to play with the computer or George is outside", clearly do not.

What kinds of symbol structures does one use to build up an information base? Analogously to databases, these symbol structures need to adhere to the rules of some information model. The concept of an information model is a direct adaptation of the concept of a data model. So is the following definition.

An *information model*[3] consists of a collection of symbol structure types, whose instances are used to describe an application, a collection of operations which can be applied to any valid symbol structure, and a collection of general integrity rules which define the set of consistent symbol structure states, or changes of states. The *relational model* for databases [47] is an excellent example of an information model. Its basic symbol structure types include `table`, `tuple`, and `domain`. Its associated operations include `add`, `remove`, `update` operations for tuples, and/or `union`, `intersection`, `join`, etc. operations for tables. The relational model supports a single integrity rule: No two tuples within a table can have the same key.

Given this definition, one can define more precisely an *information base* as a symbol structure which is based on an information model and describes a particular application.

Is an information model the same thing as a *language*, or a *notation*? For our purposes, it is not. The information model offers symbol structures for representing information. This information may be communicated to different users of an information base (human or otherwise) through one or more languages. For example, there are several different query languages associated with the relational model, of which SQL is the most widely used. In a similar spirit, we see notations as (usually graphical) partial descriptions of the contents of an information base. Again, there may be several notations associated with the same information model. e.g., the graphical notations used for data flow diagrams.

The information models proposed and used over the years have been classified into three different categories. These, roughly speaking, reflect a historical advance of the state-of-the-art on information modeling away from machine-oriented representations and towards human-oriented models which are more expressive and can cope with more complex application modeling tasks.

Physical information models. Such models employed conventional data structures and other programming constructs to model an application in terms of records, strings, arrays, lists, variable names,

[3] Adopted from Ted Codd's classic account of data models and databases [50].

B-trees, and the like. The main drawback of such models is that they force on the programmer/modeler two sets of conflicting concerns, one related to computational efficiency, and the other to the quality of the application model. For example, if one chooses to model persons in the application in terms of 8-character strings and structure an information base in terms a B-tree, these choices are driven by efficiency considerations and have nothing to do with the application.

It is interesting to note that the so-called "Year 2000 problem" has been caused precisely by this tension within physical information models between computational and representational concerns. In particular, for decades programmers abbreviated year dates with two digits, e.g., `1987` was abbreviated as `87`, with the implicit assumption (spread throughout application software) that the missing digits are `19`. All such software has the potential of malfunctioning when dealing with dates whose year component is in the next millennium.

Logical information models. The early '70s saw several proposals for *logical data models* which offered abstract mathematical symbol structures (e.g., sets, arrays, relations) for modeling purposes, hiding the implementation details from the user. The relational and network models for databases are good examples of logical models. Such models free the modeler from implementation concerns, so that she can focus on modeling ones. For instance, once the modeler has chosen the relational model, she can go ahead and use tables to build an information base, without any regard to how these tables are physically implemented. Unfortunately, logical symbol structures are flat and unintuitive as to how they should be used for modeling purposes.

Conceptual information models. Soon after logical information models were proposed, and even before relational technology conquered the database industry, there were new proposals for information models which offered more expressive facilities for modeling applications and structuring information bases. These models (hereafter, *conceptual models*) offer *semantic terms* for modeling an application, such as `Entity`, `Activity`, `Agent` and `Goal`. Moreover, they offer means for organizing information in terms of *abstraction mechanisms* which are often inspired by Cognitive Science [52], such as generalization, aggregation and classification. Such models are supposed to model an application more directly and naturally [82]. In the sequel, we focus the discussion on conceptual models, since they constitute the state-of-the-art in the field for more than two decades.

Most of the conceptual models discussed in this paper adopt some form of the locality principle alluded to earlier. There are several good reasons for this. Firstly, the principle appears to be consistent with accepted theories of human memory organization [4]. In addition, such conceptual models have been generally found to be more perspicuous and usable. Finally, such models offer the promise of efficient implementations because of their commitment to clustered information according to its subject matter. In short, conceptual models adopting such a locality principle have advantages both on psychological and engineering grounds.

Information modeling touches on deep and long-standing philosophical issues, notably the nature of generic terms included in an information base, such as `Person`, `Student`, and `Employee`. Do these terms represent abstract things in the application, in the same way `Maria` or `Myrto` represent concrete ones? Or are these representations of concepts in the mind of the modeler? Philosophers as far back as Plato have taken stands on the problem. Plato, in particular, adopted a naive realism, where objective reality includes abstract ideas, such as the concepts of student or employee, and everything is out there to be discovered. Others, including Aristotle, Locke and Hume adopted various forms of conceptualism, according to which concepts are cognitive devices created through cognitive processes. For a down to earth discussion of the range of stands on this issue within Philosophy, and how these affect the nature of information modeling, see [7].

As well, information modeling touches on fundamental methodological issues that relate to Social Sciences [138]. In particular, all the techniques discussed here adopt an *abstractionist* stance, founded on the notion of a model abstracted from an application, which captures the essence of an application, ignores bothersome details and is intended for analysis or question answering. Natural scientists and engineers use methods. Alternatively, *contextualism* places emphasis precisely on the details and idiosyncrasies of each individual application, as well as the modeling process itself. These define a context and constitute the unique identity of each particular modeling task. Ignoring them can lead to models that are inaccurate and misleading, as they simply miss the essence of each case. Contextualism has been largely developed within the Social Sciences. There are obvious pros and cons for either school of thought. It remains to be seen how can one blend them in a way that retains their strengths while at the same time alleviates their respective weaknesses.

3. Brief History

Over the years, there have been thousands of proposals for conceptual models, most defined and used once, within a single project. We note in this section some of the earliest models that launched fruitful lines of research and influenced the state-of-practice. Interestingly enough, these models were launched independently of each other and in different research areas within Computer Science.

Ross Quillian [140] proposed in his PhD thesis *semantic networks***,** a form of directed, labeled graphs, as a convenient device for modeling the structure of human memory (1966) Nodes of his semantic network proposal represented concepts (more precisely, word senses.) For words with multiple meanings, such as “plant”, there would be several nodes, one for each sense of the word, e.g., “plant” as in “industrial plant”, “plant” as in “evergreen plant “, plant as in “I plant my garden every year”, etc. Nodes were related through links representing semantic relationships, such as `isa` (“A bird is a(n) animal”, “a shark is a fish”), `has` (“A bird has feathers”) , and `eat` (“Sharks eat humans”). Moreover, each concept could have associated attributes, representing properties , such as “Penguins can’t fly”.

There are several novel ideas in Quillian’s proposal. Firstly, his information base was organized in terms of *concepts* and *associations*. Moreover, generic concepts were organized into an `isA` (or, generalization) hierarchy, supported by attribute inheritance. In addition, his proposal came with a radical computational

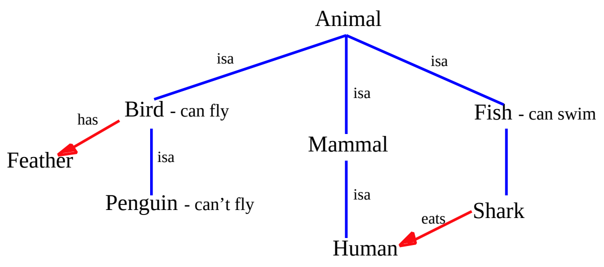

Figure 3.1: A simple semantic network

model termed *spreading activation*. Thus, computation in the information base was carried out by “activating” two concepts and then iteratively spreading the activation to adjacent, semantically related concepts. For example, to discover the meaning of the term “horse food”, spreading activation would fire the concepts `horse` and `food` and then spread activations to neighbors, until the two semantic paths

```
          horse --isa--> animal --eats--> food
          horse --isa--> animal --madeOf--> meat --isa--> food
```

are discovered. These paths correspond to two different interpretations of “horse food”, the first amounts to something like “food that horses eat”, while the second to “food made out of horses”.

Ole-Johan Dahl proposed in 1966 *Simula*, an extension of the programming language ALGOL 60, for simulation applications which require some “world modeling”. Simula [57] allows the definition of classes which serve as a cross between processes that can be executed and record structures. A class can be instantiated any number of times. Each instance first executes the body of the class and then remains as a passive data structure which can only be operated upon by procedures associated to the class. For example, the class `histo` defined in figure 3.2 is supposed to compute frequency histograms for a random variable, i.e., how often the random variable falls within each of n+1 intervals $(-\infty, X_1)$, (X_1, X_2), ...(X_n, ∞). Each histogram will be computed by an instance of the class `histo`. When the class is instantiated, the array `T` is initialized. Then each instance keeps count of a random variable's readings through use of the procedure `tabulate`, while procedure `frequency` computes the frequency for interval i.

Simula advanced significantly the state-of-the-art in programming languages, and has been credited with the launch of object-oriented programming. Equally importantly, Simula influenced information modeling by recognizing that for some programming tasks, such as simulating a barber shop, one needs to build a model of an application. According to Simula, such models are constructed out of class instances (*objects*, nowadays). These are the basic symbol structures which model elements of the application. Classes themselves define common features and common behaviours of instances and are organized into subclass hierarchies. Class declarations can be inherited by subclasses through some form of (textual, actually) inheritance.

```
class histo (X, n);array X;integer n;
 begin integer N; integer array T[0;n];
   procedure tabulate (Y); real Y;
       begin integer i; i := 0; ... end;
   procedure frequency (i); integer i;
       frequency := T[i]/N;
 integer i;
 for i := 0 step 1 until n do
       T[i] := 0; N := 0
 end
end.
```

Figure 3.2: A Simula class definition

Jean-Raymond Abrial proposed the *semantic model* for databases in 1974 [2], shortly followed by Peter Chen's *entity-relationship model*[4] [43]. Both were intended as advances over logical data models, such as Codd's relational model proposed only a few years earlier.

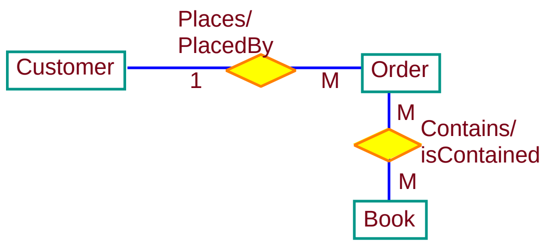

Figure 3.3: An entity-relationship diagram

The entity-relationship diagram of figure 3.3 shows entity types `Customer`, `Order` and `Book`, and relationship `Places/PlacedBy` , `Contains/isContained`. Roughly speaking, the diagram represents the fact that "Customers place orders" and "Orders contain books". The `Places` relationship type is one-to-many, meaning that a customer can place many orders but each order can only be placed by a single customer, while `Contains` is a many-to-many relationship type ("an order may contain many books, while a book may be contained in many orders").

[4] The model was actually first presented at the First Very Large Databases (VLDB) Conference in 1975.

Novel features of the entity-relationship model include its built-in terms, which constitute *ontological assumptions* about its intended application domain. In other words, the entity-relationship model assumes that applications consist of *entities* and *relationships*. This means that the model is not appropriate for applications which violate these assumptions, e.g., a world of fluids, or ones involving temporal events, state changes, and the like. In addition, Chen's original paper showed elegantly how one could map a schema based on his conceptual model, such as that shown on figure 3.3, down to a logical schema. These features made the entity-relationship model an early favorite, perhaps the first conceptual model to be used widely world-wide.

On the other hand, Abrial's semantic model was more akin to object-oriented data models that became popular more than a decade later. His model also offers entities and relations, but includes a procedural component through which one can define procedures for performing four operations on instances of a class and can attach these to classes.

Douglas Ross proposed in the mid-'70s the *Structured Analysis and Design Technique* (SADT™) as a "language for communicating ideas" [147, 148]. The technique was used by Softech, a Boston-based software company, in order to specify requirements for software systems.

According to SADT, the world consists of activities and data. Each activity consumes some data, represented through input arrows from left to right, produces some data, represented through output arrows from left to right, and also has some data that control the execution of the activity but are neither consumed nor produced. For instance, the `Buy Supplies` activity of figure 3.4 has input arrow `Farm Supplies`, output arrows `Fertilizer` and `Seeds` and control arrows `Prices` and `Plan & Budget`. Each activity may be defined through a diagram such as that shown in figure 3.4 in terms of sub-activities. Thus `Growing Vegetables` is defined in terms of the sub-activities `Buy Supplies`, `Cultivate`, `Pick Produce` and `Extract Seeds`.

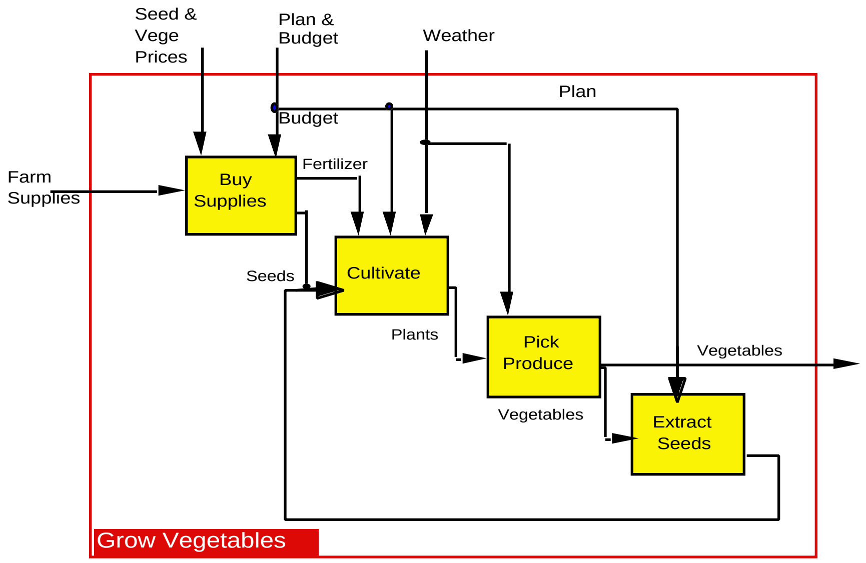

Figure 3.4: An SADT activity diagram

One of the more elegant aspects of the SADT conceptual model is its duality: Data are described in terms of diagrams with input, output and control arrows too, but these now represent activities which can produce, consume or affect the state of a given datum.

Ross' contributions include a conceptual model with some advanced ontological assumptions. Unlike the entity-relationship model, applications consist of a static and a dynamic part according to SADT. He also was influential in convincing software engineering researchers and practitioners alike that it pays to have diagrammatic descriptions of how a software system is to fit its intended operational environment. This contributions helped launch Requirements Engineering as an accepted and important early phase in software development.

After these pioneers, research on conceptual models[5] and modeling broadened considerably, both in the number of researchers working on the topic, and in the number of proposals for new conceptual models. In Databases, dozens of new *semantic data models* were proposed, intended to "capture more of the semantics of an application" [49]. For instance, *RM/T* [49] attempts to embed within the relational model the notion of entity and organizes relations into generalization hierarchies. *SDM* (*Semantic Data Model*) [82], offers a highly sophisticated set of facilities for modeling entities and supports the organization of conceptual schemata in terms of generalization, aggregation, as well as a grouping mechanism. *Taxis* [123] adopts ideas from semantic networks and Abrial's proposal to organize all components of an information system, even exceptions and exception-handling procedures, in terms of generalization hierarchies (taxonomies). [166] presents an early but thorough treatment of data models and modeling, and [89, 133] survey and compare several semantic data models.

The rise of object-oriented programming as the programming paradigm of the '80s (and '90s) led to object-oriented databases, which adopted some ideas from semantic data models and combined them with concepts from object-oriented programming [8, 178]. Early object-oriented data models supported a variety of sophisticated modeling features (e.g., *Gemstone* [55], based on the information model of Smalltalk), but the trend with recent commercial object-oriented database systems seems to converge towards the information model of popular object-oriented programming languages, such as C++. As such, object-oriented data models seem to be taking a step backwards with respect to conceptual modeling. The rise of the internet and the World Wide Web has created tremendous demand for integrating heterogeneous information sources. This has led to an emphasis on *metamodeling* techniques in Databases, where one is modeling the meaning and structure of the contents of different information sources, such as files, databases, digitized pictorial data etc., rather than an application [101, 174].

Within Artificial Intelligence (AI), semantic network proposals proliferated in the seventies [73], including ones that treated semantic networks as a graph-theoretic notation for logical formulas. During the same period, [119] introduced the notion of *frames* as a suitable symbol structure for representing common sense knowledge, such as the concept of a room or an elephant. A frame may contain information about the components of the concept being described, links to similar concepts, as well as procedural information on how the frame can accessed and change over time. Moreover, frame representations focus specifically on capturing common sense knowledge, a problem that still remains largely unresolved for Knowledge Representation research. Examples of early semantic network and frame-based conceptual models include *KRL* [19], *KL-ONE* [30] and *PSN* [106].

Since the early eighties there have been attempts to integrate ingredients from semantic networks, Logic and procedural representations. An early example of this trend is *Krypton* [31] and later *terminological languages* such as *CLASSIC* [27]. A CLASSIC information base consists of two components: a terminological component where terms are described, and an assertional one including assertions about the application. For example, a CLASSIC information base may include a description for the term `Bachelor`, which uses other more primitive terms such as `Married`, `Male`, and `Person`, along with an assertion involving a particular bachelor, for example, `Bachelor(John)`. The '80s also witnessed a

[5] The term "conceptual modelling" was used in the '70s either as a synonym for semantic data modelling or in the technical sense of the ANSI /X3/SPARC report [6] where it refers to a model that allows the definition of schemata lying between external views, defined for different user groups, and internal ones defining one or several physical 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 [35].

growing interest in the study of tradeoffs between the expressiveness and the tractability of knowledge representation techniques [33]. Such studies are now serving as major methodological vehicles in Knowledge Representation research. Knowledge Representation is thoroughly presented in [34], reviewed in [107] and overviewed in [102].

Requirements Engineering was born around the mid-'70s, partly thanks to Ross and his SADT proposal, partly thanks to others such as [15] who established through empirical study that "the rumored 'requirements problems' are a reality". The case for world modeling was articulated eloquently by Michael Jackson [92], whose software development methodology [93] starts with a "model of reality with which [the system] is concerned." The use of conceptual models for information systems engineering was launched by [157], while Bubenko's *Conceptual Information Model*, or CIM [39] is perhaps the first comprehensive proposal for a formal requirements modeling language. Its features include an ontology of entities and events, an assertional sublanguage for specifying constraints, including complex temporal ones. Greenspan's RML (*Requirements Modeling Language*) [79, 80, 23, 81]. attempts to formalize SADT by using ideas from knowledge representation and semantic data models. The result is a formal requirements language where entity and activities are organized into generalization hierarchies, and which in a number of ways predates object-oriented analysis techniques by several years.

During the same period, the *GIST* specification language [10], developed at ISI over the same period as Taxis, 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., [97]). *ERAE* [65] 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 [58]. The language offered for requirements modeling provides facilities for modeling goals, agents, alternatives, events, actions, existence modalities, agent responsibility and other concepts. KAOS relies heavily on a metamodel to provide a self-descriptive and extensible modeling framework. In addition, KAOS offers an explicit methodology for constructing requirements which begins with the acquisition of goal structures and the identification of relevant concepts, and ends with the definition of actions, to be performed by the new system or existing agents in the system's environment.

The state-of-practice in Requirements Engineering was influenced by SADT and its successors. *Data flow diagrams* (e.g., [62]) adopt some of the concepts of SADT, but focus on information flow within an organization, as opposed to SADT's all-inclusive modeling framework. The combined use of data flow and entity-relationship diagrams has led to an information system development methodology which still dominates teaching and practice within Information Systems Engineering. Since the late '80s, however, object-oriented analysis techniques [154, 46, 149, 91, 22] have been introduced and are becoming increasingly influential. These techniques offer a more coherent modeling framework than the combined use of data flow and entity-relationship diagrams. The framework adopts features of object-oriented programming languages, semantic data models and requirements languages. A recent proposal, the *Unified Modeling Language* (UML) [167] attempts to integrate features of the more pre-eminent models in object-oriented analysis, thereby enhancing reusability.

An early survey of issues in Requirements Engineering appears in [146] and the requirements modeling terrain is surveyed in [173]. [160] includes a monumental in volume tutorial on Requirements Engineering. Several recent textbooks on the same topic, e.g., [60], touch on modeling and survey a broad range of techniques.

The histories of conceptual modeling within the areas reviewed here did not unfold independently of each other. An influential workshop held at Pingree Park, Colorado in 1980 brought together researchers from Databases, AI, Programming Languages and Software Engineering to discuss conceptual modeling approaches, compare research directions and methodologies [35]. The workshop was followed by a series of other interdisciplinary workshops which reviewed the state-of-the-art in information modeling and related topics [36, 37, 150]. The International Conference on the Entity-Relationship Approach[6], held annually since 1979, has marked progress in research as well as practice on the general topic of conceptual modeling.

Several papers and books provide surveys of the whole field of Conceptual Modeling, or one or more of its constituent areas. [110] includes a fine collection of papers on conceptual modeling, most notably a survey [144], while [21] offers a more recent account of the whole field. [124] surveys the interface between AI and Databases, much of it related to conceptual modeling. Along a similar path, [28] discusses the similarities and differences between knowledge representation in AI and semantic data models in Databases, and [126] compares knowledge representation techniques to object-oriented data models.

It should be acknowledged that this discussion leaves out other areas where conceptual modeling has been used for some time, most notably Enterprise Modeling [168, 16, 169] and Software Process Modeling [116].

The proliferation of proposals for new conceptual models calls for some form of a comparative framework, so that one can classify new proposals, or evaluate whether a particular candidate is appropriate for a particular information modeling task. We propose to structure such a framework along three dimensions:

Ontologies. As we saw from the previous section, each conceptual model makes some assumptions about the nature of the applications it is intended to model. Such *ontological assumptions* determine the built-in terms offered by a conceptual model, and therefore its range of applicability.

Abstraction mechanisms. These determine the proposed organization of an information base using a particular conceptual model. This is a fundamental concern for conceptual models because organizations that are natural and intuitive lead to more usable information bases which can be searched effectively and can grow without users losing track of their contents.

Tools. If an information base is to scale up and remain useful for a long time, it needs tools for building, analyzing and otherwise managing an information base, to enhance its usability and give users confidence that its contents are accurate and consistent.

The reader may have noticed that the proposed characterization ignores the methodologies supported by a particular conceptual model. This omission is deliberate. All methodologies that have been proposed, including ones used in practice, are specific to particular *uses* one intends for an information base. For instance, using an information base for requirements engineering, e.g., [46], calls for a very different methodology than, say, one used for data modeling [14], or knowledge engineering in AI [85].

The next three sections describe in detail the nature of each dimension and discuss what various conceptual models offer with respect to each one.

4. Ontologies

Ontology is a branch of Philosophy concerned with the study of what exists. General ontologies have been proposed since the 18th century, including recent ones such as [40, 41]. For our purposes, an ontology characterizes some aspects of a class of applications. For instance, an ontology for time may characterize the temporal aspect of many applications in terms of points and temporal relations among them. Likewise, an ontology for manufacturing, may consist of (industrial) processes, physical and human resources and the like. Research within AI has formalized many interesting ontologies and has developed algorithms for generating inferences from an information base that adopts them (e.g., [170] describes efficient algorithms

[6] Recently renamed *International Conference on Conceptual Modeling* (ER).

for temporal reasoning). Along a very different path, [171, 172] study the adequacy of information models in accommodating a general ontology, such as that proposed in [40].

As indicated in section 2, every conceptual model offers built-in generic symbol structures, or *terms.* For instance, the entity-relationship model offers two built-in, generic terms: `entity` and `relationship` for modeling applications which are assumed to consist of entities and relationships. The reader should note that comparisons of conceptual models on the basis of their built-in terms are vulnerable to problems of synonymy, homonymy etc. In other words, two different models may be appropriate for the same class of applications, but use different terms to talk about these applications. We'd like to have a framework which deems these conceptual models as being comparable with respect to their intended subject matter. Ontologies help us achieve precisely this objective.

In order to give some structure to a broad and highly multidisciplinary topic, we focus on four rather coarse-grained ontologies, based on a broad survey of conceptual models and the primitive terms they support.

Static Ontology. This encompasses static aspects of an application, by describing what things exist, their attributes and interrelationships. Most conceptual models assume that the world is populated by *entities* which are endowed with a unique and immutable identity, a lifetime, a set of attributes, and relationships to other entities. Basic as this ontology may seem, it is by no means universal. For instance, [84] offers an ontology for material substances where entities (say, a litter of water and a pound of sugar) can be merged resulting in a different entity. Also note that very successful models, such as statecharts [83], don't support this ontology, because they are intended for a very different class of applications (real-time systems). Nor is this ontology trivial. For certain applications it is useful to distinguish between different modes of existence for entities, including physical existence, such as that of the author of this paper, abstract existence, such as that of the number 7, non-existence, characteristic of Santa Claus or my canceled trip to Japan, and impossible existence, such as that of the square root of -1 or the proverbial square circle [88].

We have already seen an example of entity and relationship descriptions (figure 3.3) Figure 4.1 shows how entity and relationship classes are defined in the KAOS requirements modeling language. According to the

```
Entity Library
Has available,checkedOut,lost: setOf[BookCopy]
coverageArea: setOf[Subject]
Invariant ( lib:Library)(lib = available   checkedOut   lost
    available ∩ checkedOut =      available ∩ lost =
   checkedOut ∩ lost =    )
...
end Library

Relationship Borrowing
Links        Borrower [Role Borrows, Card 0::N]
 BookCopy [Role BorrowedBy, Card 0::1]
Invariant (  lib:Library,bor:Borrower,bc:BookCopy)
 (Borrowing(bor,bc)   bc   lib
             bc   lib.checkedOut   ★Requesting(bor,bc)]
...
end Borrowing
```

Figure 4.1: Defining entities and relationships in KAOS

example, `Library` is an entity class with associated attributes `available`, `checkedOut`, `lost`, all of which take as values sets of instances of `BookCopy`. The definition includes one set-theoretic invariant constraint, which states that every instance of `Library` (`lib`) is the union of its three associated book copy sets, also that these sets are mutually exclusive. The relationship class defined is named `Borrowing`, and it relates the `Borrower` and `BookCopy` entity classes, has associated cardinality constraints, as well as an invariant. The invariant states that if a borrower has borrowed a book copy and the book copy is in the library, then the book copy is in the library's checked out list. Moreover, the book copy must have been requested by the borrower sometime in the past. Note that, unlike the entity-relationship model, class descriptions here can have associated user-defined invariants specified in a formal Logic-based language.

Spatial information is particularly important for applications which involve the physical world. Such information has been modeled in terms of 2- or 3-dimensional points or larger units, including spheres, cubes, pyramids etc. (for instance [59]). A hard modeling problem for spatial information is its inherently approximate nature, calling for special modeling provisions [162].

Dynamic Ontology. Encompasses dynamic aspects of an application in terms of *states*, *state transitions* and *processes*, Various flavors of finite state machines, Petri nets, and more recent statecharts have been offered since the '60s as appropriate modeling tools for dynamic discrete processes involving a finite number of states and state transitions. Such models are well-known and well-understood and they have been used successfully to describe real-time applications in telecommunications and other fields. Statecharts [83] constitute a more recent proposal for specifying large finite state machines. A statechart is defined in terms of states and transitions too. However, more than one state may be "on" at any one time., and states can be defined as "AND" or "OR" compositions of other statecharts. As a result, statecharts have been proven much more effective in defining and simulating large finite state machines than conventional methods. The statecharts model is supported by a popular CASE tool called STATEMATE.

An alternative to state-transition ontologies is founded on the notion of *process.* A process is a collection of partially ordered steps intended to achieve a particular goal [56]. Processes may be executed by agents, human or otherwise. Under different guises, processes have been modeled and studied in several different areas, including software processes (Software Engineering), activities (Requirements Engineering), plans (AI), tasks (CSCW), office procedures (Office Information Systems), and business processes (Management Studies). Depending on their intended use, process models generally focus on "how" or "what" information. Models intended to support the execution of the process focus on the "how", while models intended for analysis (such as consistency checking) focus on the "what".

CONGOLOG offers a high level specification language for concurrent processes, grounded on a theory of action developed within Knowledge Representation [61, 108]. In CONGOLOG, primitive actions can be defined in terms of pre/postconditions. These can then be composed in terms of modeling constructs such as sequencing ('`;`'), conditional (`if-then`), iteration (`while <condition> do`...) but also concurrent activity ('`||`'), non-deterministic choice (`choose`) and others. An interpreter for the language has been implemented in PROLOG which supports the simulation of, and reasoning about, modeled processes.

The example of figure 4.2 shows the definition of the process `determineCostToSettle` which describes how a claim is to be handled by an insurance company. The process consists of four sequential steps which successively consult with the claims file, get medical and vehicle (damage) appraisals, consult with each and produce a report. The step which involves fetching vehicle and medical appraisals consists of two sub-steps which are carried out in parallel. For these sub-steps, an appraiser is first chosen, and she is then asked to carry out the appraisal. It is interesting that even though CONGOLOG offers a program-like structure for describing processes, the underlying logic is designed to support reasoning with respect to process specifications, as well as simulations, even when the initial state for the process is only partly specified.

Temporal information is fundamental to the nature of dynamic worlds., Such information, for example "Maria graduated before her 20th birthdate" can be modeled in terms of points and associated relations. The temporal dimension of events, such as Maria's graduation, can be represented in terms of a single time point (for instantaneous events) or two time points. These points can then be related through relations such as `before`, `after`. [3] proposes a different ontology for time based on intervals, with thirteen associated relations such as `overlap`, `meet`, `before` and `after`. For database design, many extensions

```
procedure determineCostToSettle(claim)
       consultClaimFile(claim);
        % concurrently obtain vehicle and medical appraisals
             choose v
             [VehicleAppraiser(v)?;
                 % pick an appraiser for vehicle
                    request(v,doVehicleAppraisal(claim))]
         ||
             if ClaimInvolvesMedicalExpenses(claim) then
             choose m
             [ MedicalAppraiser(m)?;
                 % pick a medical appraiser
                     request(m,doMedicalAppraisal(claim))]);
         consultMedicalAppraisalReport(claim);
         consultVehicleAppraisalReport(claim);
         fileCostToSettleReport(claim)
end procedure
```

Figure 4.2: CONGOLOG specification of a composite process

have been proposed to the entity-relationship and other models to accommodate time, e.g., [161]. A related, and important concept in understanding physical systems is the concept of causality. Causality imposes existence constraints on events: if event A causes event B and A has been observed, B can be expected as well, possibly with some time delay. Within AI, formal models of causality have been offered as far back as [113, 142].

Intentional Ontology. Encompasses the world of agents, and things agents believe in, want, prove or disprove, and argue about. This ontology includes concepts such as *agent*, *issue*, *goal*, *supports*, *denies*, *subgoalOf*, etc. The subject of agents having beliefs and goals and being capable of carrying out actions has been studied extensively in AI, e.g., [111] addresses the problem of representing propositional attitudes, such as beliefs, desires and intentions for agents. The importance of the notion of agents, especially for situations involving concurrent actions, has a long tradition in requirements modeling, beginning with [70] and continuing with recent proposals, such as [58].

Modeling the issues which arise during complex decision making is discussed in [54]. The application of such a framework to software design, intended to capture the arguments pro and con, and the decisions they result in, has been a fruitful research direction since it was first proposed in [137], with notable refinements described in [115, 104]. For example, [115] models design rationale in terms of `questions (Q)`, `options` (O) and `criteria` (C). Figure 4.3 shows the structure of a decision space concerning the design of an Automated Teller Machine (ATM), The four questions raised, have associated options. Choice among them will be done by using an associated list of criteria. For example, for the question of what range of services will be offered (by the ATM under design), there are two options, full range and cash only, and two criteria for choosing among them. The cash-only option raises an auxiliary question, whether services can be restricted by having switchable machines, where services can be “masked out”, or by having machines which are inherently limited in the services they offer. On a complementary front, [78] studies the types of contributions a stakeholder can make to an argumentation structure such as the one shown in figure 4.3.

More recently, [45] proposes *softgoals* as a suitable concept for modeling software non-functional requirements, such as software usability, security, reliability or user-friendliness. Softgoals can be thought

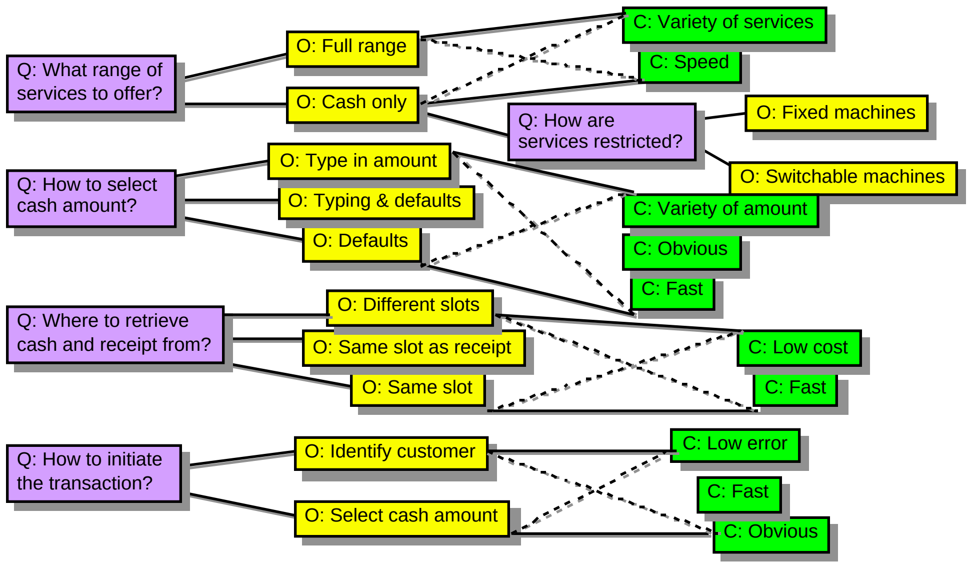

Figure 4.3: Modeling design rationale in terms of questions, options and criteria [115]

as ill-defined goals, without a clear-cut definition of when they have been satisfied (hence their name). Nevertheless, softgoals have been found to play a pivotal role in many design tasks, serving as guiding rules for choosing among alternative design decisions.

Social Ontology. This ontology covers social settings, permanent organizational structures or shifting networks of alliances and inter-dependencies [77, 120, 134, 153]. Traditionally, this ontology has been characterized in terms of concepts such as *actor*, *position*, *role*, *authority*, *commitment* etc. [175, 176, 177] proposes a novel set of concepts which focus on *strategic dependencies* between actors. Such a dependency exists when an actor has committed to satisfy a goal or softgoal, carry out a task,

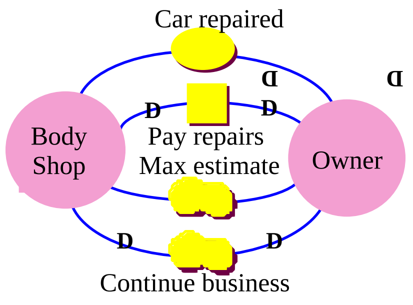

Figure 4.4: Strategic dependencies between actors

or deliver resources to another actor. Using these concepts, 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 any reengineering of business processes, and the subsequent development of software systems.

Figure 4.4 shows a simple strategic dependency graph between two actors, a (car) owner and a body shop. The dependencies shown on the graph include a goal dependency, "owner depends on the Body shop to fix the car", a resource dependency, "Body shop depends on owner to pay for repairs", and two softgoal dependencies, “Owner depends on body shop to maximize estimates”, while “Body shop depends on owner to continue business”.

5. Abstraction Mechanisms

By definition, abstraction involves suppression of (irrelevant) detail. For example, the generic concept of person can be abstracted from those of particular persons (George, Maria, Chryss,...) by suppressing personal details such as each person’s age, preferred food, etc., so as to concentrate on commonalties: persons have an address, an age ranging from 0 to 120, etc. Likewise, the concept of employee might be abstracted from those of secretary, teacher, manager and clerk by suppressing particular features of these concepts (teachers teach a subject, managers manage some group of people) and focus on commonalties (all employees have a salary, a position, a job description,...)

Abstraction mechanisms organize the information base and guide its use, making it easier to update or search it Not surprisingly, abstraction mechanisms have been used in Computer Science even before the advent of conceptual models. For instance, abstraction was used heavily in pioneering programming languages such as ALGOL 60 and LISP. Of course, the source of ideas for suitable abstraction mechanisms has to be grounded in Cognitive Science [52]. In the discussion that follows, we list for each abstraction mechanism one reference which surveys the literature (when available).

Classification (see [121])**.** This is a fundamental abstraction mechanism for human cognition, and it has proven just as fundamental for conceptual models and information bases. According to this abstraction mechanism, sometimes called *instanceOf*, an atom (entity, relationship, attribute, activity or whatever) within an information base is *classified* under one or more generic atoms (*classes*), thereby making it an instance of these classes. Instances of a class share common properties. For example, all atoms classified under `Person`, have an address and an age, while others classified under `Dog`, possess a master (sometimes) and have four legs.

Classification has been used under a number of guises to support syntactic and semantic consistency. For example, sorts in Logic [51] and types in programming languages are used mostly for syntactic checking. So do tables or relations in the relational model. In semantic networks and object-oriented information models, classification distinguishes between *tokens or objects,* which represent particular individuals in the application, and *types* or *classes* which represent generic concepts.

Besides syntactic and semantic consistency, classification can also lead to more efficient search algorithms for a knowledge base. If, for instance, the system is looking for an object whose student number is `98765432` and it is known that only students have student numbers, then only the set of instances of `Student` must be searched.

Some information models (e.g., Smalltalk) allow classification to be recursive; i.e., classes may (or must) themselves be instances of other classes. In this case the class `Person` might be an instance of the (meta)class `AnimateClass` which has as instances all classes describing animate entities.

In such situations classification may be unconstrained, allowing both a token and a class that it is an instance of to be instances of some common class, or constrained in the sense that there is a linear order of strata (or levels) so that every atom in the information base belongs to a unique level and an atom at level τ can only be an instance of classes at level τ +1. To allow for classes which have themselves as instances, such as `Class`, the class that has all classes as instances, one needs a special ω level.

Telos [125] adopts such a stratified classification scheme and uses it to classify not only entities but all atoms in an information base, including attributes and relationships. Figure 5.1 shows portions of a Telos information base. The entity[7] `EntityClass` is a metaclass at level 2 of the Telos stratosphere (as indicated by its superscript). Its instances include classes `Student`, but also `Person` and `Professor` (the latter `instanceOf` arrows are not shown on the diagram). Likewise, `RelationshipClass` is a binary relationship metaclass relating entity classes to other entity classes. Going one level down, `Student` is an instance of `SimpleEntityClass` but also of the ω-class `Class` (which actually, has all classes and metaclasses shown in the figure as instances, thought this is not shown). `Parent` and

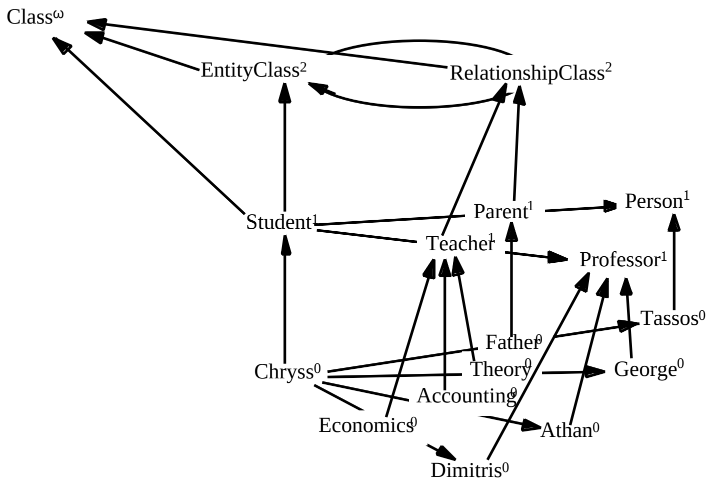

Figure 5.1: Multi-level classification of entities and relationships in Telos

`Teacher` are relationship classes and instances of `RelationshipClass`. Finally, looking at level 0, `Chryss` is a student and has three teachers and one parent. Note that the four relationships `Chryss` participates in have their own labels (so that one can distinguish between the three teachers of `Chryss` as her `Theory`, `Economics` and `Accounting` teachers respectively.)

A major advantage of any classification scheme which allows for metaclasses is that it is in a strong sense extensible. If the modeler wants to use the concept of process for the information base of figure 5.1, she can do so by adding the metaclass `ProcessClass` (with associated information, whose nature depends on the information model being used) and then use it the same way `EntityClass` and `RelationshipClass` are on figure 5.1. This is the essence of metamodeling. For more discussion on this topic, see [96].

Classification mechanisms offered in different conceptual models vary widely as to the features they offer and the overall structure they impose on the information base. In most proposals, classification has only two levels (tokens/type, instances/class, tuples/relation, etc.) Some proposals treat classes like atoms which need to be classified under metaclasses (see above). In other schemes, including Telos, everything in an information base needs to be classified under one or more classes. Moreover, some schemes allow multiple classifications for an atom, such as placing the token `Chryss` under classes `Student`, `Employee` and `HockeyPlayer`, even though these classes are unrelated to each other. Lastly, some classification schemes treat classification as an invariant property of each atom, while others allow classifications of an atom to change over its lifetime in the information base. For instance, the entity `George` might be classified first under `Newborn`, then `Child`, `Adolescent`, `Adult` during the lifetime of an information base, reflecting changes in the application being modeled.

[7] For consistency, we are using here the terminology introduced earlier in this paper, rather than that used in [125].

Generalization (see [32]). As we have already seen from previous sections, atoms in an information base which represent generic concepts have been traditionally organized into taxonomies, referred to as *isA*[8] or *generalization hierarchies*, which organize all classes in terms of a partial order relation determined by their generality/specificity. For example, `GradStudent` may be declared as a specialization of `Student` ("Every grad student is a student"), which is in turn a specialization of `Person` ("Every student is a person").

Inheritance is a fundamental ingredient of generalization hierarchies. Inheritance is an inference rule that states that attributes and properties of a class are also attributes and properties of its is-a descendants. Thus the `address` and `age` attributes of `Person`, are inherited by `Student` and, transitively, by `GradStudent`. This inheritance may be *strict* in the sense that constraints on attributes and properties can be strengthened but cannot be overridden, or *default*, in which case overriding is allowed. For example, if the age of persons has been declared to range from 0 to 100 years old, with strict inheritance the age of students can be declared to range from 5 to 80 but not from 5 to 120. Default inheritance, on the other hand, allows students to be 120 years old, though persons were declared to live only up to 100 years, or penguins to not fly though birds were declared to do so.

Generally, the organization of classes/concepts into a generalization hierarchy is left entirely up to the human modeler. An interesting alternative to this practice is offered by terminological logics [27], where term definitions can be automatically compared to see if one is more general ("subsumes") the other. For instance, the term "patients with age under 64" is subsumed by "patients with age under 70" and is disjoint from "persons with age over 72"). Within such conceptual models, generalization hierarchies can be automatically computed, simplifying the task of extending but also searching an information base.

Aggregation (see [122]). This mechanism, also called *partOf*, views objects as aggregates of their components or parts. Thus, a person can be viewed as a (physical) aggregate of a set of body parts -- arms, legs, head and the like -- or as a (social) aggregate of a name, address, social insurance number etc. Components of an object might themselves be aggregates of yet other simpler components. For example, the address of a person might be declared as the aggregation of a street number, street name, city, etc.

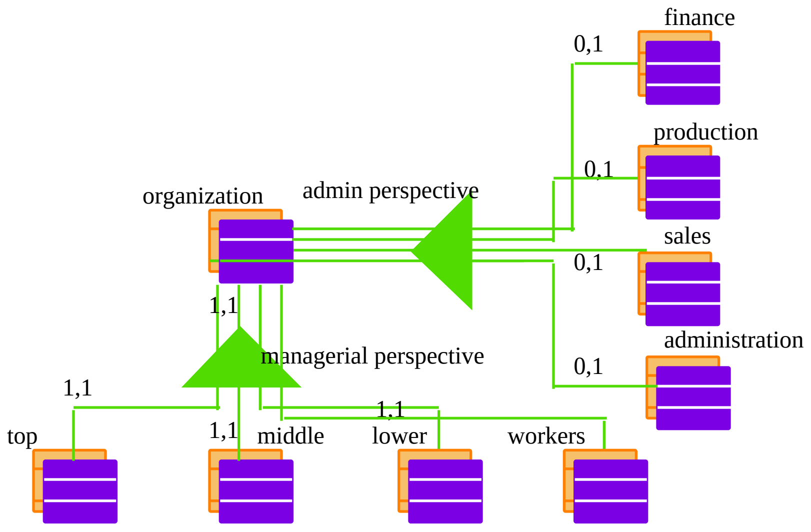

Figure 5.2: Multiple decompositions of the concept of organization

[8] Note that the literature on Conceptual Modeling has generally treated *isA* as a semantic relationship between generic atoms, such as "a shark is a fish", rather than as a relationship between an instance and its class, as in "Clyde is a fish". In AI, some of the frame-based representations used *isA* ambiguously to represent both generalization and classification relationships.

There is considerable psychological evidence that most of the information associated with a concept is of the aggregation variety [118]. Within Computer Science, [100] proposed a formalization of aggregation within his object-oriented data model in order to move away from pointer-type references between objects. In his proposal, components may be *dependent* on the aggregates to which they belong. This means that if an aggregate is removed from the information base, so are its dependent components. Likewise, a component may be *exclusive*, which means that it can only be part of a single aggregate. In addition, aggregation may be strictly hierarchical or recursive. For instance, an employee may be defined as the aggregation of a department, a salary and another employee who serves as the employee's manager. Finally, an atom in the information base may be treated as an aggregate in more than one ways. Figure 5.2 models an organization as an aggregate in two complementary ways: as a hierarchical aggregation of different managerial levels (managerial perspective), or as a vertical aggregation of departments serving different functions, such as production and marketing (administrative perspective). The notation used on figure 4.4 is adopted from [69] and it depicts aggregations in terms of triangles. Moreover, the allowable (min, max) cardinality of each aggregate is indicated by the two numbers shown next to each aggregation link. In particular, looking at the administrative perspective, an organization may have zero to one finance, production, sales, and administrative departments respectively.

Contextualization. A problem inherent in any modeling task is that there are often differences of opinion or perception among those gathering, or providing information. Contextualization can be seen as an abstraction mechanism which allows partitioning and packaging of the descriptions being added to an information base. In a situation where one is modeling how patients are admitted into a hospital, this abstraction mechanism allows relative descriptions of the process, i.e., the process according to a particular person, or even a hospital unit, rather than insisting on a description which captures all the viewpoints in one shot.

Various forms of a contextualization mechanism have been used routinely in advanced information system applications [130]. Since the early days of AI, *contexts* have found uses in problem solving, as means for representing intermediate states during a search by a problem solver in its quest for a solution [87], in knowledge representation, as representational devices for partitioning a knowledge base (e.g., [86]), In CAD and Software Engineering, *workspaces*, *versions* and *configurations* [98] are by now generally accepted notions offering respectively mechanisms for focusing attention, defining system versions and means for defining compatible system components. Database *views* have been traditionally used to present partial, but consistent, viewpoints of the contents of a database to different user groups.

More recently, such mechanisms have been adopted for object-oriented databases [1, 152, 18]. Programming language *modules*, *scopes* and *scope rules* determine which parts of a program state are visible to a particular program segment. *Perspectives*, have been proposed as a structuring mechanism for hypertext bases [139], or general purpose information bases [128].

Within Requirements Engineering, the modeling of relative viewpoints has emerged as a significant research issue in requirements engineering as well as in distributed, heterogeneous databases. [74] describes early and influence work on this issue from a Requirements Engineering perspective. Using viewpoints to relativize descriptions in an information base is serving as a mechanism for dealing with inconsistency in requirements specifications [75, 131, 143].

KAOS supports contextualization in terms of a view mechanism, intended to structure requirements specifications according to different actors (stakeholders), who may a different viewpoint and different goals for a forthcoming software system. In particular, a view is defined as a ternary relation between an actor, a master concept and a facet. For instance, a meeting may be have different associated facets from a scheduler’s and a participant’s point of view, as shown in figure 5.3. The scheduler wants to know of time constraints and the required equipment, while a participants is primarily interested in the importance of the meeting in order to plan her own schedule.

```
Entity MeetingA
Has date: Calendar, location: Place

Entity MeetingToSchedule
FacetOf Meeting SeenBy Scheduler
Has excludedDates: SeqOf[Calendar],
    requiredEquipment: SetOf[Equipment]
Invariant ( m: Meeting)(m.date not in m.excludedDate)

Entity MeetingToAttend
FacetOf Meeting SeenBy Participant
Has priority: {low,medium,high}
```

Figure 5.3: Multiple views on the concept `meeting`

Materialization (see [135])**.** This abstraction mechanism relates a class, such as `CarModel`, to a more concrete class, such as `Car`. Other examples of materialization include the relationship between a (theatrical) play, say "Hamlet", and particular productions of the play, say the one now playing at the Royal Alexandra theater. These can be further materialized by particular shows of each production, such as the matinee show on October 26, 1997. This is clearly a very useful abstraction mechanism for manufacturing applications, which involve multiple, often indistinguishable entities, of the same type. As argued in [135], the formal properties of materialization constitute a combination of those of classification and generalization.

Normalization. Special, extraordinary circumstances abound in any application, and considerably complicate its understanding, especially so during early modeling stages. This has led to proposals for a *normal-case first abstraction* [24], where only the common/typical entities, states and events in the application are modeled first, while successive pass(es) deal with the special/exceptional situations and how they are to be treated. This abstraction mechanism is particularly successful if there is some *systematic* way to find the abnormal cases and moreover, 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 examples were generalization leads 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"). [26] analyzes the conflicting desiderata for subclass hierarchies that allow such 'improper specialization', and then suggests 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 [75, 151], among others. It seems interesting to contrast and perhaps combine these approaches.

Parameterization. This is a well known abstraction technique, imported from Mathematics, that has been used with great success in programming and formal specification languages such as Z [159]. Among requirements modeling languages, ERAE and its successors [66] support parameterization to enhance the reusability of requirements. For example, one may define a requirement model with two parameters, `resource` and `consumer`, which includes actions such as `request` and `grant` and constraints such as "a grant will take place for an available resource if there is a waiting consumer". This model can then be instantiated with resource bound to `book` and customer bound `to libraryUser`. Alternatively, the model may be instantiated for a car rental application with different bindings for the two parameters.

6. Tools

Computer-based structures, for information modeling or anything else, are useless without *tools* that facilitate their analysis, design, construction and management. Assessment of a conceptual model needs to take into account the availability of such tools, alongside their expressiveness and support for abstraction mechanisms.

It is interesting to note that successful tools developed in other areas of Computer Science are founded on elaborate theoretical research, produced over many years. For example, compilers in programming languages greatly facilitate programming by performing various forms of syntactic and semantic analysis, also by generating efficient machine-executable code. Likewise, database management systems (DBMS) greatly simplify the task of managing databases, thanks to facilities such as query optimization, transaction processing and error recovery. In both cases, these tools are based on theoretical research, such as formal languages, code optimization techniques, concurrency control algorithms, and query optimization techniques.

For this paper, we focus on three basic classes of tools which we consider essential for any technology intended to support the creation and evolution of information bases: *analysis*, *design* and *management* tools.

6.1 Analysis Tools: Verification and Validation

Analysis tools perform various forms of checking on the contents of an information base to establish that they are consistent and accurately reflect the application, thereby giving users confidence that the information base is meaningful and correct. One type of checking, which we shall refer to as *verification*, treats an information base as a formal symbol structure which must satisfy syntactic and semantic rules provided by its conceptual model. Verification can take the form of establishing that syntactic rules are obeyed, checking candinality constraints for entity-relationship-like models, or checking semantic consistency of rules and constraints included in the information base.

Verification tools are grounded on the formal definition of a conceptual model. There is little non-trivial analysis one can do for a conceptual model that is only informally defined, such as SADT or data flow diagrams. There is much analysis that can be done (*but*, at great computational cost) for formal, and expressively powerful conceptual models which offer an assertional sublanguage, such as RML or KAOS. In between these extremes we have conceptual models which are formally defined, but offer no assertional sublanguage, and therefore don't need a computationally expensive inference engine. Among many others, various forms of the extended entity-relationship model fit this in-between category (e.g., [161]). This points to a great advantage of conceptual models which offer built-in terms that cover the ontology of a particular application over ones that do not, but offer instead general facilities for defining the terms that one needs for a given application: analysis tools based on the former type of conceptual model will generally be much more efficient than analysis tools based on the latter type.

Since requirements definition for software systems calls for the reconciliation of possibly inconsistent demands by different stakeholders, the analysis of inconsistency has received much attention in the literature [165]. Several approaches to the problem begin by using forms of logic where inconsistency does not lead to arbitrary conclusions[9]. [90] exploits such a logic to derive from inconsistent requirements useful conclusions. For example, if there already exist two inconsistent requirements

```
        (req1)  number(gadgets) = 1
        (req2)  number(gadgets) = 2
```

one can derive a weaker requirement,

```
                number(gadgets) ≥ 1
```

[9] Conventional logics suffer from the so-called *paradoxes of implication* which state that "anything implies a true proposition" (formally, A (B A)) and "a contradiction implies anything" (formally A ¬A B).

from both (req1) and (req2) and use it as starting point towards reconciliation of the inconsistency.

Many researchers have focused on goal inconsistencies and have proposed techniques for resolving them. [Robinson89] proposes a technique for identifying conflicting requirements, characterizing them as goal inconsistencies, and resolving them through negotiation. Along similar lines, [127, 45] offer a qualitative reasoning procedure for detecting inconsistencies between non-functional requirements, represented as softgoals. Likewise. [20] proposes an iterative process for identifying conflicting conditions, evaluating their respective risks, and resolving them through negotiation.

Other forms of inconsistencies have also been explored. [158] describes a framework for detecting overlap in requirements, noting that overlap is a prerequisite to inconsistency, while [72] explores deviations of a running system from its stated requirements. Along a different path, [24, 11] describe frameworks for modifying the contents of an information base so that it will accommodate an exception without creating an inconsistency.

A popular approach to dealing with inconsistency in AI has been to support *truth maintenance* mechanisms [64]. Such mechanisms maintain a network of links representing logical relations among facts. These can be used when inconsistencies are discovered, to identify the source of these inconsistencies and to determine what facts in the information base need to be revised. Another approach to inconsistency involves using weaker logics which explicitly exclude the above paradoxes of implication, such as *relevance logics* [5].

Here is a partial list of different types of verification-type analysis that may be offered by a particular information model, depending on the ontologies that it supports:

· Static ontology -- checking cardinality constraints, checking invariants, spatial reasoning
· Dynamic ontology -- simulation of finite state machines and extensions thereof, proving that state invariants are preserved by processes defined in terms of pre/postconditions; proving termination and liveness properties, temporal reasoning
· Intentional ontology -- goal satisfaction algorithms for AND/OR goal graphs, marker-passing algorithms
· Social ontology -- means-ends analysis

Whereas verification tools are concerned with the internal consistency of an information base vis-à-vis its conceptual model, *validation* tools check for the consistency of an information base with respect to its application. Clearly, such consistency is critical to the usefulness of an information base. Validation tools adopt a variety of strategies, depending on the nature of the conceptual model used. Some tools support validation by paraphrasing in natural language the contents of the information base [117]. The biggest problem for paraphrasing how much to include in the generated text, so it is neither ambiguous, nor verbose.

A second approach, which only applies to process descriptions in the information base, is to animate or simulate them, thereby confirming that they behave consistently with the modeler’s expectations. This is relatively straightforward for state transition models, including statecharts [83], possible but much harder for formal process specification models such as CONGOLOG [108], or Albert II [67]. A variety of tools use heuristics to pinpoint potential problem areas in the contents of an information base. [42, 63] describe two research efforts in this category.

Turning to AI-oriented tools, [136] describes early work on validating expert system rules (here the information base is capturing expertise in the performance of some task) by examining the performance of the expert system and noting failures, which are then traced back to the use or lack thereof of particular rules.

6.2 Design Tools

An information base constitutes an artifact. As such, it needs careful crafting, or design, based on rules which guide the design process. These rules suggest when is an artifact well structured and when it is not. For information modeling, such rules have been proposed for the relational model [48], and they define formally various *normal forms* for relational schemata. Placing a relational schema in these forms eliminates the danger for certain types of anomalies which can occur in the database upon the insertion/removal/update of tuples. To the extend that this work is based on tuple attributes, it also applies to other information models, such as the entity-relationship model, which offer attributes and are relatively unstructured. For more expressive conceptual models, [68] studies the quality of schemata, measured in terms of metrics, and proposes transformations for improvement and integration.

One aspect of the information base design problem which has received considerable attention for databases is schema integration. This topic encompasses techniques which take as input a number of partial database schemata, presenting partial views of the information to be handled by a database, and build a global schema. To accomplish the task, one needs to address terminological problems, such as synonymy (two different names for the same information), or ambiguity (same name used differently within different partial schemata), semantically equivalent but syntactically different symbol structures, semantic inconsistencies etc. [13] presents an authoritative survey of work in this area.

A number of expert system tools have been reported in the literature which support database design, by automating, or semi-automating schema integration, the generation of a logical schema from a conceptual one, and the selection of indexing scheme etc. [29] describes a tool in this category and compares it with others in the same area.

Since non-technical people communicate best in natural language, several projects have attempted to build tools which generate portions of an information base from natural language descriptions . OICSI [145] is one of the best-known systems in this area. It's approach is to take simple input sentences, such as "Subscribers have a name, address and subscription number" and map them first into a deep structure, which then is integrated with the current version of the information base. This approach has been attempted for database design, e.g., [99], for requirements acquisition, e.g., [145], as well as knowledge acquisition, e.g., [9, 141].

A related approach to information base design involves acquiring knowledge from examples and forms. For instance, [12] describes a method for deriving a conceptual database schema from forms in a bottom-up fashion. The method begins by building up increasingly more complex entity description and forming generalizations when two or more descriptions have overlapping components. In a similar spirit, [155] reports on an expert system which builds up a database schema from business documents, while [25] proposes a system which applies machine learning techniques to modify an existing database schema and its associated constraints in order to deal with semantic inconsistencies. The role of form analysis in computer-aided software engineering is thoroughly discussed in [44].

The last, but not least, approach to be discussed here includes tools which support reuse-based design. In this category. The basic idea for such tools is that a repository of past cases is consulted whenever a new modeling task needs to be performed. The repository includes past cases that have been cleaned up, and abstracted so that they become more generally applicable. Selection of "similar" cases is achieved by using some form of a similarity metric, which measures the distance between the new modeling task and stored past cases. [76] describes the objects and roles model (ORM) and its application in specifying reusable requirements. [112] describes a more recent, and more sophisticated requirements reuse scheme based on the KAOS model. The matching algorithm used here takes a query, defined by a partial requirements specification which needs to be completed, and matches it against other complete cases in a repository. Depending on the outcome, the query may be generalized so that it will match less similar cases. This way, retrieval from the repository can be interactive and makes full use of the structural richness of the KAOS model.

6.3 Management Tools

Management tools begin with a good implementation which offers facilities for building, accessing and updating an information base. Beyond a mere implementation, one would like to have an *efficient* implementation which scales up in the sense that primitive operations are relatively unaffected by the size of the information base. In the case of databases, such efficiency is derived from elaborate systems research into physical storage management, caching and indexing techniques, and the like

Query optimization makes it possible to efficiently evaluate queries expressed in a high level, declarative language such as SQL. experience from databases suggests that having such a facility broadens the class of users for the information base. In addition, concurrency control can increase dramatically the number of transactions that can be executed against an information base per unit time. Also, error recovery can serve as safeguard against system failure, ensuring that the information base can be returned to a consistent state after a crash.

Of course, all these are accepted features of commercial relational DBMS products. Much work has been done on extending the research which makes these features a reality for relational databases to other, more advanced data models, including object-oriented, and multimedia ones. Generally, there are few supported management tools for conceptual models. However, see ConceptBase [95, 53, 96] for a system that moves in the right direction. It is also worth noting other research projects on the subject, such as [109, 129].

7. Assessment of Conceptual Models

The three-dimensional characterization of conceptual models, can now be exploited to assess different conceptual models, to guide the design of new models so that they truly advance the state-of-the-art, also to evaluate and compare candidates for a given information modeling task. We begin the section by offering an admittedly coarse-grained evaluation of some well known conceptual models. We then present some suggestions to those who have to choose among conceptual models, or dare to design new ones.

7.1 Evaluating Conceptual Models

An obvious way to use the framework proposed in the previous section is to evaluate the degree to which different conceptual models cover the four basic ontologies, support the six abstraction mechanisms and offer the three classes of tools. The overall “mark” for a given conceptual model is some combination of the marks it gets with respect to each dimension. Likewise, the overall mark for each dimension is some combination of the partial marks for each component of the dimension.

A disclaimer is in order here. Like any other form of evaluation scheme, this one is highly dependent on the definition of the dimensions we have proposed, and arbitrary with respect to the actual assigned “marks”. Nevertheless, we consider it a useful coarse-grain instrument for the assessment of conceptual models, certainly better than no evaluation scheme at all.

Let’s use the entity-relationship (ER) model as example to present and illustrate our evaluation scheme. Firstly, the model clearly supports the static ontology. Secondly, the model offers minimal support for the other ontologies in the sense that one can define activities, goals and social dependencies as entities or relationships, but none of the semantics of these terms is embedded in the ER model or the tools it offers. To assign marks, and keep things simple, we will allocate a mark in the range {**great**, **good**, **OK**, **so-so**, **none**}, depending on how well a conceptual model supports each ontology, abstraction mechanism or tool type.

In the case of the ER model, its mark for the static ontology might be **good+**, and **so-so** for all other ontologies. Why only **good+**? Well, there other conceptual models, e.g., [82], which offer a substantially more elaborate notion of entity than the ER model. See also the entity definition facilities of KAOS (figure 4.1), where the modeler can specify invariants using a formal language. In other words, we reserve a perfect mark for the best proposals in the literature in supporting a particular ontology or abstraction mechanism, and in offering tools.

To make more concrete the evaluation of how well the static ontology is supported, let's set some rules:

- Rule SO.1: A built-in term for entity/object/...with associated attributes results in a mark of **OK** or better.
- Rule SO.2: Support for cardinality constraints on attributes adds a mark.
- Rule SO.3: Support for a variety of additional types of constraints, or user-defined constraints gets a top mark.
- Rule SO.4: Facilities for fully defining terms such as entity and relationship, get a mere OK

On that basis, conceptual models such as SDM and KAOS get top marks for their support for the static ontology.

Turning to abstraction mechanisms, ER supports a simple form of classification, in the sense that every entity/relationship is an instance of a single entity/relationship type. This is clearly a long way from the sophistication of some of the more recent proposals, so it only gets a so-so. Other abstraction mechanisms are supported circumstantially. One can define `isA`, `partOf`, `instanceOf`, etc. as relationships, but the semantics of these are not embedded either in tools, or the ER model itself. Let's give ER, rather generously, so-so- marks for other abstractions.

Again, we can make the marking for each abstraction mechanism more concrete by laying down some rules. We will use classification as example:

- Rule CA.1: A two-level classification mechanism gets a so-so.
- Rule CA.2: A multi-level classification mechanism gets at least OK.
- Rule CA.3: Multiple classification adds a mark.
- Rule CA.4: Variable classification adds a mark.

With these rules, conceptual models such as UML and Telos which supports a multi-level classification scheme, as well as multiple and variable classification get top marks.

With regard to tools, there is a variety of tools which perform simple forms of analysis on ER schemata, including normalization tools. Moreover, there are well-accepted techniques for mapping down an ER information base into a relational database. For these reasons, we give ER high marks with respect to the tool dimension, say great, great, and good+ respectively. A few points have been taken away for management tools because whatever is available was developed specifically for the relational model. Overall then, the ER model gets high marks for its support of the static ontology and the availability of management tools, but can use enhancements in all other areas. Of course, for the time it was proposed, this conceptual model is still a classic.

SADT. This model supports, to some extent, both the static and dynamic ontologies, though its marks in both cases are OK. Likewise, with respect to abstraction mechanisms, SADT offers a single structuring mechanism where each box (representing data or activity) can be elaborated into a diagram. This structuring mechanism has no associated semantics, so it can be treated as a rather primitive form of aggregation and lands a mark of OK. Finally, the marks for design and management tools are also OK, since SADT did come with a basic implementation along with some, generally ad hoc, design rules. However, since SADT is only informally defined, little analysis is possible.

Extended entity-relationship model (EER model). This is an extension of the entity-relationship model which has appeared in various forms since the late '70s (used, for example, in [14]). The EER model supports sophisticated forms of generalization and aggregation, plus the simple form of classification found in ER, so we'll give it so-so, good, and good respectively for classification, generalization and aggregation. The marks for supported ontologies don't change, but analysis, design and normalization tools become a bit more problematic than for the ER model, because of the presence of the two abstraction mechanisms.

UML ([UML97]). This proposal promises to be the industry standard for software modeling, with interest and participation in its definition from major hardware and software vendors. Our evaluation here focuses on the features of UML intended for requirements modeling. With respect to ontologies, UML supports adequately static and dynamic ontologies with concepts derived from object-oriented notations. It also seems to support some concepts related to business modeling, so we'll give it good+, good+, none, and OK- respectively. Regarding abstraction mechanisms, UML supports well classification, generalization, also some forms of aggregation, contextualization . Its marks for this dimension: great, great, OK, and OK. For tools, it is not clear yet what will be available. As with SADT, we note that as long as UML remains an informal model, there isn't much that can be done with respect to analysis tools.

ConceptBase ([95]). This is a prototype implementation of a version the Telos model (called O-Telos). This model was designed with metamodeling applications in mind. Accordingly, its most important feature is its extensibility, i.e., new terms can be defined as metaclasses with associated constraints and deductive rules, to be used for a particular modeling task. Not surprising, the model is generally weak with respect to supported ontologies. We'll give it OK for the each of the four ontologies. On the other hand, there is excellent support for generalization and classification, a rudimentary form of aggregation, as well as a solid implementation. Accordingly, we give it great for generalization, classification, OK for aggregation and contextualization, good for analysis, design and management tools respectively.

KAOS ([58]). This conceptual model supports well static, dynamic and intentional ontologies, and gets great marks for each. Along the abstraction mechanism dimension, KAOS gets good to great marks for generalization, classification, aggregation, and contextualization. Moreover, thanks to its formal foundations, it can support a variety of analysis and design tools and has a prototype implementation. On tools then, we'll give KAOS good, good and OK marks respectively.

The table of figure 7.1 summarizes the evaluation of the six conceptual models discussed above.

	ER	**SADT**	**EER**	**UML**	**ConceptBase**	**KAOS**
Static	good+	OK	good+	good+	OK	great
Dynamic	so-so	OK	so-so	good+	OK	great
Intentional	so-so	none	so-so	none	OK	great
Social	so-so	none	so-so	OK-	OK	so-so
Classificati	so-so	so-so	so-so	great	great	great
Generaliz.	so-so-	none	good	great	great	great
Aggreg.	so-so-	so-so	good	OK-	OK	good
Context.	so-so-	so-so	so-so-	OK	OK	great
Analysis	great	so-so	good+	?	good	good
Design	great	OK	great	?	good	good
Managem.	good+	OK	good	?	good	OK

Figure 7.1: Evaluation of six conceptual models

7.2 Choosing a Conceptual Model

Suppose then that you are leading a project that has an application modeling component. How could you use the insights of the proposed assessment methodology to select the conceptual model that is best suited for your project?

A starting point for the selection process is to consider three alternatives. The first involves adopting an existing generic information base. For example, if you are developing a banking system, there may be an existing collection of defined banking terms available, using some conceptual or even logical information model. Adopting it has the obvious advantage of cutting costs and project time. No wonder reuse of generic information bases has received much attention in AI [105, 132], as well as in Requirements Engineering, e.g., [103, 112].

A second alternative is to adopt an existing conceptual model and develop your own information base from scratch. This is clearly a preferred alternative only if the first one does not apply. Selection of an existing conceptual model can proceed by identifying the nature of the application to be modeled, i.e., answering the question “what kinds of things will we need to talk about?”, or “does the application involve temporal or spatial information?” In addition, one needs to make rough guesses on the size of the information base, i.e., “how many generic and token atoms?” For a project which will involve a large number of generic terms, abstraction mechanisms are essential. For instance, for a project involving the description of aircraft designs where the number of generic terms may be in the tens of thousands, use of abstraction is unavoidable. For a project which will require the construction of a very large information base, say with billions of instances, analysis, design and management tools are a must.

It is important to keep in mind during the selection process that not all abstraction mechanisms will be equally useful to any given project. For a project requiring the modeling of few but very complex concepts, aggregation is clearly most helpful and modeling through some form of stepwise decomposition is the most appropriate modeling method. If, on the other hand, the modeling task involves many simple but similar concepts, generalization is the abstraction to turn to. Finally, a project involving heavy use of multiple descriptions for one and the same entity, such as multiple versions of the same design or multiple perspectives on the same data, use of contextualization is recommended to organize and rationalize these perspectives of the same reality.

The last, most time consuming, and least desirable alternative is for your project to develop its own conceptual model. Such an alternative is feasible only for long term projects. Before adopting it, you may want to think twice what is unique about your modeling task, and why it is that none of the existing conceptual models apply. Also to think of the overhead involved in designing and implementing your new conceptual model, before you can actually exploit it.

The moral of this discussion is that not all conceptual models are created equal with regard to their usefulness for your modeling task. The exercise of identifying what is the application like, also what abstractions and tools are most useful can greatly reduce the danger of disappointment later on. Moreover, design of new conceptual models should be avoided at all costs because it is rarely justified when you are trying to model an application, as opposed to furthering the state-of-the-art in conceptual modeling.

8. Conclusions and Directions for Future Research

We have set out to study and characterize information modeling, in research and practice, also in different areas of Computer Science. Our study included a brief history of the area, a proposed comparative framework for conceptual models consisting of three orthogonal dimensions, and the assessment of several conceptual models from the literature.

Even though crude, the assessment of the previous section suggests a direction of progress since the mid-’70s. Specifically, pioneering conceptual models such as ER and SADT support one ontology, or less than two respectively, and offer little in terms of abstraction mechanisms. Conceptual models of the mid-’80s, such as ones embedded in object-oriented databases and requirements languages, support aggregation and generalization and improve on the support of the static and dynamic ontology. Finally, in the mid-’90s we are looking at conceptual models which begin to grapple with the intentional ontology, treat classification with the respect that it deserves, and support various forms of parameterization and contextualization (e.g., [58, 69]).

At the same time, the comparative framework suggests fruitful directions for further research. In particular, we expect that the study of new ontologies, and the consolidation of existing ones, such as the intentional and social ontologies, will continue. Other abstraction mechanisms will be proposed, formalized and integrated into existing or new conceptual models. The field of Databases will continue to push the limits of database management technology so that it applies to ever more powerful and expressive information models, including conceptual ones. As well, new application areas will need to be explored and methodologies will have to be developed for modeling, analogously to the state of practice today for knowledge engineering , data modeling, and requirements engineering.

Of course, advancing information modeling practice to a new generation of ever more powerful conceptual models requires more than just research. Computer science and information systems engineering professionals need solid education and training on the subject, which must begin with the thorough teaching of conceptual modeling as a core subject and an important professional skill in Computer Science and Information Systems curricula.

Acknowledgments

I am grateful to many colleagues who helped shape my ideas on the topic of conceptual modeling, most notably Alex Borgida (Rutgers University), Sol Greenspan (GTE Laboratories), Matthias Jarke (Technical University of Aachen), Hector Levesque (University of Toronto), Nicholas Roussopoulos (University of Maryland), and Eric Yu (University of Toronto). Thanks are also due to the anonymous reviewers for helpful feedback on earlier drafts of the paper..

Funding sources for the research include the Natural Sciences and Engineering Research Council (NSERC) of Canada and the City University of Hong Kong, where the paper was actually prepared.

References

[1] S. Abiteboul, and A. Bonner. Objects and views. Proceedings of the ACM SIGMOD International Conference on the Management of Data, 238 - 247 (1991).
[2] J-R. Abrial. Data semantics. Klimbie and Koffeman (eds.) *Data Management Systems*, North-Holland (1974).
[3] J. Allen. Towards a general theory of action and time. *Artificial Intelligence(23),* 123-154 (1984).
[4] J. Anderson, and G. Bower. *Human Associative Memory*, Winston-Wiley (1973).
[5] A. Anderson and N. Belnap. *Entailment: The Logic of Relevance and Necessity*, Volume I, Princeton University Press (1975).
[6] ANSI/X3/SPARC Study Group on Database Management Systems. Interim report. *FDT 7(2)* (1975).
[7] J. Artz. A crash course on metaphysics for the database designer. *Journal of Database Management 8(4)*, 25-30 (1997).
[8] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *Deductive and Object-Oriented Databases*, Elsevier Science Publishers, Amsterdam (1990).
[9] D. Ayuso, V. Shaked, and R. Weischedel. An environment for acquiring semantic information. Proceeding Twentyfifth Annual Meeting of the Association for Computational Linguistics, (ACL'87), Stanford, 32-40 (1987).
[10] R. Balzer, N. Goldman, and D. Wile. Operational specifications as a basis for rapid prototyping. *Proceedings Symposium on Rapid Prototyping, ACM Software Engineering Notes, 7(5),* 3-16 (1982).

[11] R. Balzer. Tolerating inconsistency. Proceedings International Conference on Software Engineering, Austin, 158-165 (1991).

[12] C. Batini, B. Demo, A. Di Leva. A methodology for conceptual schema design of office databases. *Information Systems 9(3-4),* 251-263 (1984).

[13] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys 18(4),* 323-364 (1986).

[14] C. Batini, M. Lenzerini, and S. Navathe. *Database Design: An Entity-Relationship Approach*, Benjamin Cummings, (1992).

[15] T. Bell, and T. Thayer. Software requirements: Are they really a problem. Proceedings Second International Conference on Software Engineering, 61-68 (1976).

[16] P. Bernus, and L. Nemes (eds.). *Modelling and Methodologis for Enterprise Integration.* Chapman and Hill (1996).

[17] P. Bernus, K. Mertins, and G. Schmidt (eds.). *Handbook on Architectures for Information Systems*. Springer-Verlag, (1998).

[18] E. Bertino. A view mechanism for object-oriented databases. International Conference on Extending Database Technologies (EDBT'92), Vienna, Lecture Notes in Computer Science, Springer-Verlag, (1992).

[19] D. Bobrow, and T. Winograd. An overview of KRL, a knowledge representation language. *Cognitive Science* 1, 3-46 (1977).

[20] B. Boehm, P. Bose, E. Horowits, and M. Lee. Software requirements negotiation and renegotiation aids: a theory-W-based spiral approach. Proceedings, Seventeenth International Conference on Software Engineering, 243-253 (1995).

[21] M. Boman, J. Bubenko, P. Johannesson, and B. Wangler. *Conceptual Modeling*, Prentice Hall (1997).

[22] G. Booch. *Object-Oriented Analysis and Design.* Benjamin-Cummings, second edition, (1992).

[23] A. Borgida, S. Greenspan, and J. Mylopoulos. Knowledge representation as a basis for requirements specification. *IEEE Computer 18(4),* (1985). Reprinted in C. Rich, and R. Waters. *Readings in Artificial Intelligence and Software Engineering.* Morgan-Kaufmann, (1987).

[24] A. Borgida. Features of languages for the development of information systems at the conceptual level. *IEEE Software 2(1)*, (1985).

[25] A. Borgida, T. Michell, and K. Williamson. Learning improved integrity constraints and schemas from exceptions in databases and knowledge bases. In [37].

[26] A. Borgida. Modeling class hierarchies with contradictions. Proceedings ACM SIGMOD International Conference on the Management of Data, 434-443 (1988).

[27] A. Borgida, R. Brachman, D. McGuiness, and L. Resnick. CLASSIC/DB: A structural data model for objects. Proceedings ACM SIGMOD International Conference on the Management of Data, Portland, (1989).

[28] A. Borgida. Knowledge representation and semantic data modelling: Similarities and differences. Proceedings Entity-Relationship Conference, Geneva (1990).

[29] M. Bouzeghoub. Using expert systems in schema design. In [110].

[30] R. Brachman. On the epistemological status of semantic networks. In [73].

[31] R. Brachman, R. Fikes, and H. Levesque. Krypton: A functional approach to knowledge representation. *IEEE Computer 16(10),* 67-74 (1983).

[32] R. Brachman, R. “What is-a is and isn’t”: An analysis of taxonomic links in semantic networks. *IEEE Computer*, (1983).

[33] R. Brachman, and H. Levesque. A fundamental tradeoff in knowledge representation and reasoning (Revised Version). In [34].

[34] R. Brachman, and H. Levesque, (eds.). *Readings in Knowledge Representation.* Morgan Kaufmann, Los Altos, CA, 41-70, (1984).

[35] M. Brodie, and S. Zilles, (eds.). Proceedings of workshop on data abstraction, databases and conceptual modelling. Pingree Park Colorado, Joint SIGART, SIGMOD, SIGPLAN newsletter, (1981).

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

[37] M. Brodie, and J. Mylopoulos, (eds.). *On Knowledge Base Management Systems: Perspectives from Artificial Intelligence and Databases*, Springer-Verlag, (1986).

[38] M. Brodie. On the development of data models. In [37].
[39] J. Bubenko. Information modeling in the context of system development. In *Proceedings IFIP Congress '80*, 395-411 (1980).
[40] M. Bunge. *Treatise on Basic Philosophy: Ontology I -- The Furniture of the World.* Reidel, (1977).
[41] R. Carnap. *The Logical Structure of the World: Pseudoproblems in Philosophy*. University of California Press, (1967).
[42] C. Cauvet, C. Priox, and C. Rolland. Information system design: An expert system approach. Proceedings IFIP WG 2.6 & WG 8.1 Meeting, Canton, (1988).
[43] P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems 1(1)*, (1976).
[44] J. Choobineh, M. Mannino, and V. Tseng. The role of form analysis in computer-aided software engineering. In [110].
[45] L. Chung. *Representing and Using Non-Functional Requirements: A Process-Oriented Approach.* Ph.D. thesis, Department of Computer Science, University of Toronto, (1993).
[46] P. Coad, and E. Yourdon. *Object-Oriented Analysis*. Yourdon Press, Englewood Cliffs, NJ, (1990).
[47] E. Codd. A relational model for large shared data banks. *Communications of the ACM 13(6)*, 377-387 (1970).
[48] E. Codd. Further normalization of the data base relational model. In *Data Base Systems*, Courant Computer Science Symposia Series, Prentice Hall, (1972).
[49] E. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems 4,* (1979).
[50] E. Codd. Relational databases: A practical foundation for productivity. *Communications of the ACM*, (1982).
[51] A. Cohn. On the appearance of sortal literals: a non substitutional framework for hybrid reasoning. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning,* Toronto, 55-66 (1989).
[52] A. Collins, and E. Smith. *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence.* Morgan-Kaufmann, (1988).
[53] Technical University of Aachen. http://www-i5.informatik.rwth.-aachen.de/CBdoc.
[54] J. Conklin, and M. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *Transactions on Office Information Systems, 6(4),* 281-318 (1988).
[55] G. Copeland, and D. Maier. Making Smalltalk a database system. Proceedings ACM SIGMOD International Conference on the Management of Data, Boston, (1984).
[56] B. Curtis, M. Kellner, and J. Over. Process modelling. *Communications of the ACM 35(9)*, (1992).
[57] O-J. Dahl, and C. Hoare. Hierarchical program structures. In O-J. Dahl, E. Dijkstra, and C. Hoare, (eds.). *Structured Programming*, Academic Press, (1972).
[58] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal–directed requirements acquisition. *Science of Computer Programming, 20*, 3-50 (1993).
[59] E. Davis. *Representing and Acquiring Geographic Knowledge.* Pitman, (1986).
[60] A. Davis. *Software Requirements: Objects, Functions and States.* Prentice Hall, (1993).
[61] G. DeGiacomo, Y. Lespérance, and H. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the Situation Calculus. Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97), Nagoya, 1221-1226 (1997).
[62] T. De Marco. *Structured Analysis and System Specification*. Prentice Hall, (1979).
[63] O. deTroyer, R. Meersman, and P. Verlinden. RIDL on the KRIS case: A workbench for NIAM. Proceedings IFIP Conference on Computerized Assistance During the Information System Lifecycle, Elsevier, (1988).
[64] J. Doyle. A truth maintenance system. *Artificial Intelligence 12,* 231-272 (1979).
[65] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert, and A. Rifaut. A knowledge representation language for requirements engineering. *Proceedings of the IEEE 74(10),* (1986).
[66] E. Dubois, P. Du Bois, and A. Rifaut. Elaborating, structuring and expressing formal requirements for composite systems. Proceedings Fourth International Conference on Advanced Information Systems Engineering (CAiSE-92), Manchester, (1992).

[67] E. Dubois, P. Du Bois, and F. Dubru. Animating formal requirements specifications for cooperative information systems. Proceedings International Conference on Cooperative Information Systems, Toronto, 101-112 (1994).

[68] C. Eick, and P. Lockemann. Acquisition of terminological knowledge using database design techniques. Proceedings ACM SIGMOD International Conference on the Management of Data, (1985).

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

[70] M. Feather. Language support for the specification and derivation of concurrent systems. *ACM Transactions on Programming Languages 9(2),* 198-234 (1987).

[71] S. Fickas, and P. Nagarajan. Critiquing software specifications: A knowledge-based approach. *IEEE Software*, (1988).

[72] S. Fickas, and M. Feather. Requirements monitoring in dynamic environments. Proceedings Second IEEE International Symposium on Requirements Engineering, York, (1995).

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

[74] A. Finkelstein, J. Kramer, *et al.* Viewpoints: A framework for multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering, 2(1),* World Scientific Publishing, 31-57 (1992).

[75] A. Finkelstein, D. Gabbay, *et al.* Inconsistency handling in multi-perspective specifications. *Proceedings of the Third European Software Engineering Conference,* Milan, Italy, Springer-Verlag, 84-99 (1993).

[76] M. Fugini, and B. Pernici. Specification reuse. In [110].

[77] J. Galbraith. *Designing Complex Organizations*. Addison Wesley, (1973).

[78] O. Gotel, and A. Finkelstein. Contribution structures. Proceedings Second IEEE International Symposium on Requirements Engineering, York, England, (1995).

[79] S. Greenspan, J. Mylopoulos, and A. Borgida. Capturing more world knowledge in the requirements specification. *Proceedings 6th International Conference on Software Engineering*, Tokyo, (1982). Reprinted in P. Freeman, and A. Wasserman, (eds.). *Tutorial on Software Design Techniques*. IEEE Computer Society Press, 1984.

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

[81] S. Greenspan, A. Borgida, and J. Mylopoulos. A requirements modeling language and its logic. *Information Systems, 11(1),* 9-23, 1986.

[82] M. Hammer, and D. McLeod. Database description with SDM: A semantic data model. *ACM Transactions on Database Systems*, (1981).

[83] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*, (1987).

[84] P. Hayes. The second naive physics manifesto. In J. Hobbs and R. Moore (eds.). *Formal Theories of the Commonsense World,* Ablex Publishing Corp., Norwood, N. J., 1-36, (1985).

[85] F. Hayes-Roth, D. Waterman, and D. Lenat (eds.), *Building Expert Systems,* Addison-Wesley, Reading, MA, (1983).

[86] G. Hendrix. Encoding knowledge in partitioned networks. In [73], 51-92, (1979).

[87] C. Hewitt. Procedural embedding of knowledge in PLANNER. Proceedings Second International Joint Conference on Artificial Intelligence (IJCAI'71) , London, 167-182, (1971).

[88] G. Hirst. Ontological assumptions in knowledge representation. *Proceedings First International Conference on Principles of Knowledge Representation and Reasoning,* Toronto, 157-169 (1989).

[89] R. Hull, and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys 19(3),* (1987).

[90] A. Hunter, and B. Nuseibeh. Analyzing inconsistent specifications. Proceedings Third IEEE International Symposium on Requirements Engineering, Annapolis, 78-86 (1997).

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

[92] M. Jackson. Information systems: Modeling, sequencing and transformation. Proceedings Third International Conference on Software Engineering, 72-81 (1978).

[93] M. Jackson. *System Development*, Prentice Hall, (1983).

[94] M. Jarke, J. Mylopoulos, J. Schmidt, and Y. Vassiliou. DAIDA: An environment for evolving information systems. *ACM Transactions on Information Systems 10(1),* 1- 50 (1992).

[95] M. Jarke, R. Gallersdoerfer, M. Jeusfeld, M. Staudt, and S. Eherer. ConceptBase - A deductive object base for metadata management. *Journal of Intelligent Information Systems 4(2),* (Special Issue on Advances in Deductive Object-Oriented Databases), 167-192 (1995).

[96] M. Jeusfeld, M. Jarke, H. Nissen, and M. Staudt. ConceptBase: Managing conceptual models about information systems. In [17].

[97] W. Johnson, M. Feather, and D. Harris. Representing and presenting requirements knowledge. *IEEE Transactions on Software Engineering,* 853-869 (1992).

[98] R. Katz. Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys22(4),* 375-408, (1990).

[99] M. Kersten, H. Weigand, D. Dignum, and J. Boom. A conceptual modeling expert system. Proceedings Fifth Internation Conference on the Entity-Relationship Approach, Dijon, (1986).

[100] W. Kim, E. Bertino, and J. Garza. Composite objects revisited. Proceedings Object-Oriented Programming Systems, Languages and Applications (OOPSLA'89), 337-347, (1989).

[101] W. Klas, and A. Sheth, (eds.). Special issue: Metadata for digital data. *ACM SIGMOD Record 23(4),* (1994).

[102] B. Kramer and J. Mylopoulos. A Survey of knowledge representation. S. Shapiro (ed.), *The Encyclopedea of Artificial Intelligence*. John Wiley and Sons Inc., second edition, (1991).

[103] W. Lam, J. McDermid, and A. Vickers. Ten steps towards systematic requirements reuse. Proceedings Third IEEE International Symposium on Requirements Engineering, Annapolis, 6-15 (1997).

[104] J. Lee, and K-Y. Lai. What is design rationale?. *Human-Computer Interaction 6(3-4),* (1991).

[105] D. Lenat, and R. Guha. *Building Large Knowledge Based Systems - Representation and Inference in the CYC Project*. Addison-Wesley, (1990).

[106] H. Levesque, and J. Mylopoulos. A procedural semantics for semantic networks. In [73], 93-120 (1979).

[107] H. Levesque. Knowledge representation and reasoning. *Annual Review of Computer Science 1986(1)*, Annual Reviews Inc., 255-287 (1986).

[108] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 59--84 (1997).

[109] P. Lockemann, H-H. Nagel, and I. Walter. Databases for knowledge bases: An empirical study. *Data and Knowledge Engineering, 7,* 115-154, (1991).

[110] P. Loucopoulos, and R. Zicari, (eds.). *Conceptual Modeling, Databases and CASE: An Integrated View of Information System Development*. Wiley, (1992).

[111] A. Maida, and S. Shapiro. Intensional Concepts in Propositional Semantic Networks. *Cognitive Science*, 6, 291-330, (1982).

[112] P. Massonet, and A. van Lamsweerde. Analogical reuse of requirements frameworks. Proceedings Third IEEE International Symposium on Requirements Engineering, Annapolis, 17-26 (1997).

[113] J. McCarthy. Programs with Common Sense. In M. Minsky (ed.), *Semantic Information Processing,* MIT Press, Cambridge MA, 403-418 (1968).

[114] L. Macfarlane, and L. Reilly. Requirements traceability in an integrated development environment. Proceedings Second IEEE International Symposium on Requirements Engineering, York, England, (1995).

[115] A. MacLean, R. Young, V. Bellotti, and T. Moran. Questions, options, criteria: Elements of design space analysis. *Human-Computer Interaction 6(3-4),* (1991).

[116] N. Madhavji, and M. Penedo, (eds.). Special Section on the Evolution of Software Processes. *IEEE Transactions on Software Engineering 19(12),* (1993).

[117] K. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*, Cambridge University Press, (1985).

[118] G. Miller, and P. Johnson-Laird. *Language and Perception*. Harvard University Press, (1976).

[119] M. Minsky. A framework for representing knowledge. In P. Winston, (ed.). *The Psychology of Computer Vision*. MIT Press, (1975).

[120] H. Mintzberg. *The Structuring of Organizations*. Prentice Hall, (1979).

[121] R. Motschnig-Pitrik, and J. Mylopoulos. Classes and instances. *International Journal of Intelligent and Cooperative Systems 1(1),* (1992).

[122] R. Motschnig-Pitrik. The semantics of parts versus aggregates in data/knowledge modeling. Proceedings Fifth Conference on Advanced Information Systems Engineering (CAiSE'93), Paris, (1993).

[123] J. Mylopoulos, P. Bernstein, and H. Wong. A language facility for designing data-intensive applications. *ACM Transactions on Database Systems 5(2),* (1980). Reprinted in S. Zdonik, and D. Maier. *Readings in Object-Oriented Database Systems.* Morgan-Kaufmann, (1989).

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

[125] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, (1990).

[126] J. Mylopoulos. Object-orientation and knowledge representation. Proceedings IFIP Working Conference on Object-Oriented Databases: Analysis, Design and Construction, Windermere, (1990).

[127] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using non-functional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering 18(6),* (1992).

[128] J. Mylopoulos, and R. Motschnig-Pitrik. Partitioning an information base through contexts. Proceedings Third International Conference on Cooperative Information Systems (CoopIS'95), Vienna, (1995).

[129] J. Mylopoulos, V. Chaudhri, D. Plexousakis, A. Shrufi, and T. Topaloglou. Building knowledge base management systems. *Very Large Databases Journal 5(4),* (1996).

[130] M. Norrie, and M. Wunderli. Coordination system modelling. Proceedings of the Thirteenth International Conference on The Entity Relationship Approach, Manchester , (1994).

[131] B. Nuseibeh, J. Kramer, and A. Finkelstein. Expressing the relationships between multiple views in requirements specification. Proceedings Fifteenth International Conference on Software Engineering, Baltimore, IEEE Computer Science Press, 187-196 (1993).

[132] R. Patil, R., Fikes, P. Patel-Schneider, D. Mckay, T. Finin, T. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. Proceedings Third International Conference on Knowledge Representation and Reasoning, Boston, (1992).

[133] J. Peckham, and F. Maryanski. Semantic data models. *ACM Computinf Surveys 20(3),* (1988).

[134] J. Pfeffer, and G. Salancik. *The External Control of Organizations: A Resource Dependency Perspective.* Harper and Row, (1978).

[135] A. Pirotte, E. Zimanyi, D. Massart, and T. Yakusheva. Materialization: A powerful and ubiquitous abstraction pattern. Proceedings Very large Databases Conference (VLDB'94), Santiago Chile, (1994).

[136] P. Politakis. *Empirical Analysis of Expert Systems*. Pitman Publishers, (1985).

[137] C. Potts, and G. Bruns. Recording the reasons for design decisions. Proceedings Tenth International Conference on Software Engineering, Singapore, (1988).

[138] C. Potts. Requirements models in context. Proceedings Third IEEE International Symposium on Requirements Engineering, Annapolis, (1997).

[139] V. Prevelakis, and D. Tsichritzis. Perspectives on software development environments. Proceedings Fifth Conference on Advanced Information Systems Engineering (CAiSE'93), Paris, Lecture Notes in Computer Science 685, Springer Verlag, (1993).

[140] R. Quillian. Semantic memory. In M. Minsky (ed.). *Semantic Information Processing*. MIT Press, Cambridge, MA, 227-270, (1968).

[141] L. Rau, P. Jacobs, and U. Zernik. Information extraction and text summarization using linguistic knowledge acquisition. *Information Processing and Management 25(4),* 419-428 (1989).

[142] C. Rieger. An organization of knowledge for problem-solving and language comprehension. *Artificial Intelligence 7(2),* 89-127, (1976).

[143] W. Robinson, and S. Fickas. Supporting multi-perspective requirements engineering. Proceedings IEEE Conference on Requirements Engineering, (1994).

[144] C. Rolland, and C. Cauvet. Trends and perspectives in conceptual modeling. In [110].

[145] C. Rolland, and C. Proix. A natural language approach to conceptual modeling. In [110].

[146] G-C. Roman. A taxonomy of current issues in requirements engineering. *IEEE Computer 18(4),* (1985).

[147] D. Ross, and A. Schoman. Structured analysis for requirements definition. In [163], 6-15 (1977).
[148] D. Ross. Structured analysis: A language for communicating ideas. In [163], 16-34 (1977).
[149] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*, Prentice Hall, (1991).
[150] J. Schmidt, and C. Thanos (eds.). *Foundations of Knowledge Base Management*. Springer Verlag, (1989).
[151] P. Schoebbens. Exceptions in algebraic specifications: On the meaning of 'but'. *Science of Computer Programming, 20*, 73-111 (1993).
[152] M. Scholl, C. Laasch, and M. Tresch. Updateable views in object-oriented databases. Proceedings Second International Conference on Deductive and Object-Oriented Database Systems, Munich, (1991).
[153] W. Scott. *Organizations: Rational, Natural or Open Systems*. Prentice Hall, second edition, (1987).
[154] S. Shlaer, and S. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data*. Prentice Hall, (1988).
[155] N. Shu, V. Lum, H. Wong, and C. Chang. Specification of forms processing and business procedures for office automation. *IEEE Transactions on Software Engineering 8(5)*, 499-511 (1982).
[156] J. Smith, and D. Smith. Database abstractions: Aggregation and generalization. *ACM Transactions on Database Systems 2(2)*, 105-133 (1977).
[157] A. Solvberg. A contribution to the definition of concepts for expressing users' information system requirements. *Proceedings International Conference on the E-R Approach to Systems Analysis and Design*, (1979).
[158] G. Spanoudakis, and A. Finkelstein. Interference in requirements engineering: The level of ontological overlap. TR-1997-01, City University, London, (1997).
[159] J. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, (1989).
[160] R. Thayer, and M. Dorfman. *System and Software Requirements Engineering*, (two volumes), IEEE Computer Society Press, (1990).
[161] C. Theodoulidis, B. Wangler, and P. Loucopoulos. The entity-relationship-time model. In [110].
[162] T. Topaloglou, and J. Mylopoulos. Representing partial spatial information in databases: A conceptual modeling approach. Proceedings Fifteenth International Conference on Conceptual Modeling (ER'96), Cottbus, (1996).
[163] *IEEE Transactions on Software Engineering 3(1)*. Special Issue on Requirements Analysis. (1977).
[164] *IEEE Transactions on Software Engineering. 18(6) & 18(10)*, Special Issue on Knowledge Representation and Reasoning in Software Development, (1992).
[165] *IEEE Transactions on Software Engineering*. Special Issue on Managing Inconsistency in Software Development, (1998) (to appear).
[166] D. Tsichritzis, and F. Lochovsky. *Data Models.* Prentice Hall, (1982).
[167] Rational Software Corporation. *Unified Modeling Language Resource Centre.* http://www.rational.com/uml, (1997).
[168] F. Vernadat. Computer-integrated manufacturing: On the database aspect. Proceedings of CAD/CAM and Robotics Conference, Toronto, (1984).
[169] F. Vernadat. *Enterprise Modeling and Integration*. Chapman and Hall, (1996).
[170] M. Vilain, H. Kautz, and P. van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In D. Weld, and J. De Kleer, (eds.). *Readings in Qualitative Reasoning About Physical Systems*. Morgan Kaufmann, (1989).
[171] Y. Wand. A proposal for a formal model of objects. In W. Kim, and F. Lochovsky, (eds.). *Object-Oriented Concepts, Databases and Applications*. Addison-Wesley, (1989).
[172] Y. Wand, and R. Weber. An ontological model of an information system. *IEEE Transactions on Software Engineering 16*, 1282-1292 (1990).
[173] D. Webster. Mapping the design representation terrain: A survey. TR-STP-093-87, Microelectronics and Computer Corporation (MCC), Austin, (1987).
[174] J. Widom. Research problems in data warehousing. Proceedings Fourth Conference on Information and Knowledge Management, (1995).
[175] E. Yu. Modeling organizations for information systems requirements engineering. Proceedings IEEE International Symposium on Requirements Engineering, San Diego, IEEE Computer Society Press, 34-41 (1993).

[176] E. Yu, and J. Mylopoulos. Understanding 'why' in software process modeling, analysis and design. Proceedings Sixteenth International Conference on Software Engineering, Sorrento, (1994).

[177] E. Yu, J. Mylopoulos, and Y. Lespérance. AI models for business process re-ejngineering. *IEEE Expert 11(4),* (1996).

[178] S. Zdonik, and D. Maier (eds.). *Readings in Object-Oriented Databases.* Morgan Kaufmann, (1989).