4
Generalization/Specialization as a Basis for Software Specification

Alexander Borgida, John Mylopoulos
and Harry K.T. Wong
University of Toronto

ABSTRACT *This paper describes a software specification methodology based on the notion of concept specialization. The methodology, which is particularly useful for Information Systems, applies uniformly to the various components of such systems, such as data classes, transactions, exceptions, and user interfaces (scripts), and its goal is the systematic and structured description of highly detailed world models, where concepts occur in many variations. An example from the domain of university information systems is used to illustrate and motivate the approach.*

1. Introduction

Complaints about the high cost of software development and maintenance are now commonplace. Research in Programming Languages, Software Engineering, and Database Management attempts to deal with this problem by proposing tools and techniques for managing the ever increasing complexity of software. Many of these techniques are based on *abstraction mechanisms* that advocate the development of software in a stepwise fashion, each step involving only some of the details of the

---

Alexander Borgida's current address: Dept. of Computer Science, Rutgers University, New Brunswick, NJ 08903.

Harry K.T. Wong's current address: Lawrence Berkeley Labs, Berkeley, CA 94720.

whole problem while others, hopefully the less relevant ones, are suppressed until some later step.[1]

For example, some current methodologies advocate the creation of a sequence of *models* ranging from the initial "real-world problem" to the final machine-executable program. This abstraction, called *Representation* in [SS79], involves implementation details and is supported by a number of languages and methodologies (*e.g.,* [PARN72] and [WLS76], among others). A second abstraction that has been advocated involves grouping a collection of units into a new conceptual unit (*Aggregation*). Software development through stepwise refinement [WIRT71] is based on this abstraction and offers decomposition as a methodological tool for building complex systems. This chapter focuses on *conceptual modelling*, *i.e.,* the specification of models that are closer to the human's conception of reality than to the machine's representation, and proposes a stepwise methodology based on *concept specialization.* In this case, the abstraction involves factoring out the commonalities in the description of several concepts into the description of a more general concept, and the refinement process reintroduces these details by specifying the ways in which a more specialized concept differs from the more general one. This methodology, which we call *taxonomic specification*, is complementary to stepwise refinement and methodologies based on Representation, and we feel that it is particularly appropriate when there are a large number of relatively simple, but interrelated, facts to be captured.

Section 2 elaborates on the notions of "model" and "abstraction," and Section 3 discusses Generalization as an abstraction mechanism and compares our version to others that have been proposed in the literature. In Section 4 we present a long example using a language along the lines of TAXIS[2] [MBW80] [WONG81] to illustrate the nature and the virtues of taxonomic specification. Section 5 sketches scripts that facilitate the description of the user dialogues that need to be supported by the system under design, and Section 6 discusses exceptions and an exception-handling mechanism that can be used as a tool in cases of over-abstraction. Finally, Section 7 presents conclusions and directions for further research.

---

[1] The term "abstraction" is used here in a more general sense than usual in the field of Programming Languages, where its meaning is usually that of "representation abstraction," as that notion is defined below.

[2] TAXIS is a programming language for the design of interactive information systems, such as on-line inventory control and airline reservations, which supports taxonomic programming and offers many of the features discussed in the rest of the chapter.

2. Models and Abstractions

The observation that a computer system constitutes a model of a "world" or "slice of reality" about which it contains information has been made repeatedly in the literature (*e.g.*, [ABRI74] [BC75] [WILS75]), and is most obvious in the case of information systems. This observation motivates our first axiom: in a substantial number of cases, the process of software specification can be viewed as the process of building an accurate model of some enterprise. In order to facilitate the task of the modeler, as well as communication with the eventual users, we also assume that these models should reflect naturally and directly the users' conceptualization of the universe of discourse.

Unfortunately, the term "model" has several different technical meanings and it seems appropriate to contrast them with the sense used in this chapter.

The term receives its most precise and technical sense in the field of mathematical logic where, given a set of axioms and their deductive consequences, one interprets them in terms of a "model" (*i.e.*, a set of mathematical entities and relations which satisfy the axioms). This notion underlies in one way or another all other uses of the term, but in this technical sense its use is restricted to the theory of mathematical logic.

Two other uses of the term, namely as an analogue device (*e.g.*, a wind-tunnel model of an airplane) and as a mathematical model (*e.g.*, Maxwell's equations as a model of electricity) are common in science and engineering, but they are quite distinct from the term as used in this chapter.

From the cognitive sciences we obtain the notion of "conceptual model," which is much closer to what we want. Such a model consists of a number of symbol structures and symbol structure manipulators which, according to a rather naive mentalistic philosophy, are supposed to correspond to the conceptualizations of the world by human observers. This view appears to underlie work on "semantic data models" (*e.g.*, survey in [BORG82b]) and "knowledge representation" (*e.g.*, overviews in [BD81] and [MYLO81]).

Another sense of the term "model" is current in the area of Data Base Management Systems under the guise of "data model." A *data model* (see [TL82] for example) specifies the rules according to which data are structured and what associated operations are permitted on them. The traditional data models underlying commercial Database Management Systems consider as data only strings and numbers, and they are concerned primarily with the manner in which data is accessed by the user (in some cases reflecting how data is stored in the computer), and have little or no regard for the *interpretation* process required to make *information* out of *data*.

Since our concern here is with human oriented models of a world, we adopt the "conceptual model" sense of the term rather than one of the others.

If one accepts the need for conceptual models he is immediately faced with the problem of identifying the constructs that facilitate their creation. Not surprisingly, many of the proposed constructs have their roots in epistemological methods for organizing knowledge.

Abstraction is a fundamental conceptual tool used for organizing information. The following are just a few aspects of abstraction that are useful in describing complex conceptual models:

- *Classification.* Grouping entities that share common characteristics into a *class* over which uniform conditions hold. The class `PERSON`, for example, can be derived from the entities `john smith`, `mary brown`, *etc.,* through classification. The inverse of Classification, *Instantiation*, can be used to obtain other entities that conform to the constraints associated with the definition of the class `person`.

- *Aggregation.* Treating a collection of concepts as a single concept. For example, `person` could be thought of, rather naively, as an aggregation of its `name`, `address`, and `profession`. *Decomposition* is the opposite of Aggregation since it decomposes a class into its constituent parts.

- *Generalization.* Extracting from one or more given classes the description of a more general class that captures the commonalities but suppresses some of the detailed differences in the descriptions of the given classes. `Employee`, for instance, is a generalization of the classes `secretary`, `trucker`, and `accountant`. The process that has the opposite effect to Generalization (*i.e.,* creates a new class by introducing additional detail to the description of an existing one) is called *Specialization.*

There are other abstraction mechanisms, such as "normalization" (suppression of details that deal with deviations from the norm and emphasis of details that deal only with the normal or ordinary circumstances [BORG82a]) but the three above have received the most attention. Conceptual models of complex worlds are bound to be large if they are to account for sufficiently many properties of their subjects. The abstraction mechanisms discussed above offer both *organizational principles* and *design methodologies* for conceptual models.

Not surprisingly, each of these mechanisms, as well as the representation abstraction noted in the introduction, has led to proposals for software development methodologies. For example, Representation has led to abstract data type-related methodologies and important programming languages such as Simula [DH72], CLU [LSAS77], Alphard [WLS76], *etc.* (See the chapter by Shaw.) These have been defined to support the development of software through the gradual introduction of implementation detail. Similarly, aggregation has led to proposals for the design of software through stepwise refinement (by decomposition) (*e.g.,* [WIRT71] [DIJK72]), and languages such as Pascal have been found suitable for supporting this abstraction. Object-oriented programming is based on classification and is supported by Simula and some of its successors, notably Smalltalk [INGA78] and Actors [HBS73]. Finally, generalization leads to methodologies that organize the collection of classes constituting a model into hierarchies (taxonomies). Simula (again!) and in a different context [SS77] follow this route.

Of course, a successful software development methodology has to employ as many of the above mentioned abstraction mechanisms as possible. For the purposes of research strategy, however, it seems fruitful to focus on one mechanism and to formalize it, examine its applicability, and study its usefulness; hence, this chapter concentrates on the notion of Generalization/Specialization.

3. Generalization/Specialization

We are interested in formulating a methodology for building conceptual models based on Generalization/Specialization. The key idea of such a methodology is that a model can be constructed by modelling first, in terms of classes, the most general concepts and tasks in the application area, and then proceeding to deal with sub-cases through more specialized classes. Models constructed through such a process have their classes structured into a *taxonomy* or so-called *IS-A hierarchy.* For example, when building a student enrollment system for a university, one might consider first the concepts of student and course and the task of enrolling a student for a course. Later, the designer can consider graduate and undergraduate students and courses, full- and part-time students, day and evening courses, and the rules and regulations that apply to these classes. Indeed, we believe that most situations of application programming, such as ones for student enrollment, inventory control, airline or hotel reservations, inherently involve *large* amounts of *simple* detail, and that taxonomies offer a fundamental tool for coping with such situations.

Taxonomies of classes have been used in one form or another in artificial intelligence, programming languages and databases for over a decade (*e.g.,* [QUIL68] [DH72]). However, each author formulates them differently. The rest of this section outlines the main points of view and contrasts them to the one adopted in the chapter.

One of the advantages of organizing descriptions into taxonomies is the notion of *inheritance*. Since instances of a subclass are generally also instances of its superclasses, there is no need to repeat the information specified in the description of a class for each of its subclasses, and their own subclasses, *etc.* As a result, taxonomic descriptions can be abbreviated, and some clerical errors can be avoided by reducing the amount of repetition in descriptions.

There are two basic ways to view the description one associates with a class such as `PERSON` (intended to model the concept of person). The first is that the description simply characterizes a prototypical instance of the concept, *i.e.*, a prototypical person. It follows from this view that some of the assertions of the description (*e.g.*, that a person has a telephone number) might be contradicted by a particular instance of the class (*e.g.*, `bill brown` who doesn't have a telephone number because he doesn't have a telephone). Much of the Knowledge Representation research within Artificial Intelligence views classes (concepts/frames/units/ . . .) in those terms. The second view treats the description associated with a class as asserting necessary conditions that must be satisfied by all of its instances. Simula and the semantic data models adopt this view. The consequences of this choice on the nature of taxonomies (IS-A hierarchies) are immediate.

1. *Class as prototype.* If class C *IS-A* class B (*i.e.*, C is a specialization of B and therefore lower down on the hierarchy) and B has some properties, these properties can be over-ridden in the definition of C. Thus even though "All birds fly" and "Penguins are birds," penguins do not necessarily fly (in fact they don't!). Inheriting properties from a more general class is done by default here (*default inheritance*), *i.e.*, only if the description of the more specialized class does not assert otherwise. The assertions associated with a class have an "unless-otherwise-told" or default nature.

2. *Class as template.* If C *IS-A* B and B has some properties, then necessarily C must have the same properties. Asserting that every bird flies implies, with this view, that penguins *can't* be birds since they don't fly. Thus inheritance of properties from a class to its specializations is now strict (*strict inheritance*).

We adopt the second view of what a class is, and, as we will see in Sections 5 and 6, treat exceptions through a separate exception-handling mechanism rather than by weakening the assertional force of a class definition.

There is still another choice to consider once one has made this decision. Neither Simula, nor the semantic data models that have been proposed, with the exception of TAXIS, allow a property to be "refined" as one specializes a class. For instance, if we have asserted that every person has an age between 0 and 120 (years) in the description of the class PERSON, we would like to refine this property to "every student has an age between 12 and 80 when PERSON is specialized to obtain STUDENT. With strict inheritance a new class inherits all the properties of its generalizations and it can also have new properties of its own. However, it cannot refine any of the properties of its generalizations along the lines suggested above.

We consider that an important modelling tool is the ability to refine properties of a class as one generates its specializations.

We would like to emphasize even at this stage that the utility of generalization hierarchies for software engineering is not limited to the use of inheritance, although this feature is often the most visible. In particular, the *construction* of the hierarchies systematizes and structures the *process of information/requirements gathering*, and since the hierarchies persist even after the system is designed, they provide an *organization* of the information that facilitates the *location of information* and the *estimation of the effect of changes* (*e.g.*, changes to a class description will affect all subclasses of that class) during program maintenance. Furthermore, the development of procedures through stepwise refinement by specialization down the IS-A hierarchy permits an incremental testing of such programs, in contrast to programs developed through stepwise refinement by decomposition [WIRT71], where only the final stage of the refinement is an executable program. In this chapter, we shall concentrate on the use of Generalization hierarchies in the description of Information Systems; we describe elsewhere the application of these ideas to requirements specification [GBM82] and verification [WONG81] [BORG81], among others.

4. Taxonomic Specification

As argued earlier, we view our task as having to describe the conceptual data objects and activities that occur in the domain of discourse, and their associated constraints. Throughout this section our ideas will be illustrated with examples from the student enrollment process at the University of Toronto. In order not to distract from the methodological aspects of this chapter, we have chosen to present the examples in a semiformal way, preferring a skeletal language augmented with English descriptions of programming language code; the interested reader is referred to [WONG81] and [MBW80] for details of a programming language into which these descriptions can be immediately translated. Better known languages such as Simula [DH72] and Smalltalk [INGA78], and recent systems such as Pie [BG80a], also support versions of the abstractions involved in our methodology.

Aggregation is supported by the notion of *property*—a function which, when evaluated for an entity, returns one of its components or, more vaguely, a related entity. For example, if "cs100" was some particular course, then *title*("cs100"), *limit*("cs100"), *size*("cs100") and *class-list*("cs100") might represent respectively the title of this course, 'Introduction to Programming,' the maximum and current enrollment in the course, say 800 and 647 respectively, and the list of students currently enrolled in "cs100." In the case of aggregation as applied to an activity, some properties would have as value those activities that would result from one step of decomposition as suggested by stepwise refinement. Other properties of an activity would indicate the participants in the activity, as well as possibly other "meta-information" such as its beginning time, deadline for completion, *etc.* For example, if "e" is the activity of enrolling the student "bilbo" into the course "cs100," then we might have *student*("e") = "bilbo," *course*("e") = "cs100" as the participants, and one of the component activities, say *p2*("e"), would add *student*("e") to the *class-list* of *course*("e").

Clearly, describing a model in terms of such specific "factual" information is hardly satisfactory for the purposes of software specification. In fact, the information required is of a "generic" nature, and this imposes limitations on the facts that the computer system might record. These are known as semantic constraints [HM75].

Classification provides one important means for introducing such generic information by allowing us to present both the properties applicable to all of the instances of a class and the constraints that restrict the possible values of these properties. For example, the definition of the class COURSE might include as properties *title, limit, size, class-list, instructor*, *etc.* Constraints on these would include at the very least an indication of the range classes of these functions, as well as possibly more complex limitations. For example, *size* might have 0 to 2000 as range, as well as the additional constraint that the *size* of the course must be no greater than its *limit* (constraint *course-limit* of COURSE). This kind of information is obviously closely related to the notions of type declaration in Programming Languages and to schema definition in Databases. The diagram in Figure 4.1 describes two classes of objects, STUDENT and COURSE, which will play a central role in our enrollment system.

Whereas classification of objects yields data class definitions, classification of activities yields transaction class definitions, similar to procedure declarations in Programming Languages. The participants in the activity play the role of formal parameters in procedure declarations; the constraints on the values of these properties include type information and initial, final, and invariant conditions, which act as 'well-formedness' constraints on the transaction invocation and execution. As suggested above, other properties indicate the subactions comprising the definition of the procedure. Figure 4.2 is a schematic description of the ENROL transaction class in our proposed student record system.

```
data class STUDENT with
  attributes
    name: PERSON_NAME;
    age: 12..80;
    home-address: ADDRESS;
    univ-address: ADDRESS;
    faculty: {Arts&Science, GradSchool, Medicine};
    status: {Full-time, Part-time}
    courses-taken: set of COURSE;
    taking-courses: set of COURSE;
end STUDENT;

data class COURSE with
  attributes
    title: STRING;
    dept: DEPARTMENT;
    limit: 0..2000;
    size: 0..2000;
    enrollment: set of STUDENT;
    level: {1st-year,...,4th-year,intro-grad,adv-grad};
    instructor: PROFESSOR;
  invariants
    course limit: (size ≤ limit);
end COURSE;
```

Figure 4.1 *Definition of STUDENT and COURSE Classes*

```
transaction ENROL with
  parameters
     s: STUDENT;
     c: COURSE;
  prerequisites
     Not-full?:  (c.size<c.limit);
  actions
     a1: add c to the list taking-courses of s;
     a2: add s to the enrollment-list of c;
     a3: increment size of c by 1;
end ENROL;
```

Figure 4.2. *The ENROL Transaction Definition*

Since defining the appropriate classes is an important aspect of our methodology we offer two heuristics for designers.

1. In general, properties, regarded as functions, are undefined everywhere except over the instances of the class they have been associated with, and usually every new class definition introduces several new properties; hence a new class ought to be introduced whenever

we desire a property whose domain of definition is not an already existing class.

2. Secondly, almost all semantic constraints are stated through quantification over the instances of classes (*e.g.,* all instances of STUDENT must have as *age* an instance of INTEGER in the range 12 to 80).

Additional expressive machinery occasionally may be needed but we feel that these heuristics are adequate for a large number of modelling situations.

Our methodology so far has only systematized and slightly extended normal practices in software specification. As stated above, we are particularly interested in extending this methodology to deal with situations where there are a multitude of minute details relating to many classes that share common characteristics. Specialization allows us to describe each *subclass* of a more general class by specifying only the additional details necessary for its definition, through the notion of inheritance described in Section 3. As discussed there, in specializing a class we can 1) "strengthen" any of the constraints stated for the parent class (*i.e.,* replace a constraint of the parent class, say A, with a stronger one B such that B implies A); 2) provide additional constraints; and/or 3) introduce new properties and related constraints. Figure 4.3 is a description of the subclass GRAD_STUDENT of STUDENT.

```
data class GRAD_STUDENT is a STUDENT with
  attributes
    faculty: {GradSchool};
    dept: DEPARTMENT;
    advisor:PROFESSOR;
    level: {MSc, PhD};
end GRAD_STUDENT;
```

Figure 4.3 *A Specialized Subclass of STUDENT*

Note that the *faculty* property of STUDENT was (consistently) refined in GRAD_STUDENT so that it admits only one value, GradSchool. Also, three new properties, *dept, advisor,* and *level* were introduced; these only apply to graduate students while all other properties of STUDENT are of course inherited by GRAD_STUDENT. It may be worth contrasting our notion of inheritance with that used in Simula or Smalltalk. Simula allows only complete textual inheritance in the sense that in describing subclasses one cannot alter the code described for the superclass, but must inherit it completely (strict inheritance). Smalltalk on the other hand uses default inheritance. Our description of Specialization above requires that the refined version of a constraint must not contradict the original one; this ensures that all instances of subclasses are also legal instances of all the classes higher up in the hierarchy. Finally, it appears that neither Simula nor Smalltalk allows multiple inheritance (*i.e.*, inheritance from several distinct classes), a feature that we shall find quite useful in describing transactions.

In the case of transactions, the specialization of parameters proceeds in the same way as that of properties for data classes. As far as component properties are concerned, one can either specialize the transaction class of which a property must be an instance, or provide additional properties. The example in Figure 4.4 illustrates the specialization of the transaction class ENROL whose instances have graduate students as the student *s* participant.

```
specialize ENROL (s : GRAD_STUDENT, c : COURSE)
  add
    prerequisite
      Advanced? (c.level ≥ 4th-year);
    action
      a4:  inform School of Graduate Studies
                about the enrollment;
end;
```

Figure 4.4 *Specializing the ENROL Transaction*

Here an additional prerequisite is added to ENROL for graduate students to ensure that they are not allowed to take courses that are below Fourth Year. There is also an extra action that will be carried out only during the enrollment of graduate students. We will henceforth use ENROL (*s* : C1, *c* : C2) to denote the subclass of ENROL whose instances have the student *s* in C1 and the course *c* in C2.

In order to provide a more convincing demonstration of the utility of specialization as a specification methodology, we present next a partial list of conditions on the enrollment of students in Computer Science courses. Many of these were actually required until recently at the University of Toronto, although some were not checked until the students were told that they had not taken the appropriate program and would have to take one or more additional courses! It would clearly be beneficial if these conditions were incorporated into any computerized student information system, and hence modelling these constraints becomes an important goal. For ease of reference, we have labeled each with a mnemonic name followed by a question mark, indicating that each of these will be a constraint predicate. The list in Figure 4.5 is clearly haphazardly drawn up and it is exactly our point that such restrictions should be gathered in a more systematic way by system designers, and hence our software development methodology should support such systematization. In this case, we have chosen to encode most, although not all, of these constraints as prerequisites of the

ENROL transaction. The understanding is that if any of these prerequisites is false, an *exception* will be raised and execution of the transaction will be suspended. (We leave unspecified for the moment what action should be taken to handle such exceptions.) As the first step in our description, we present in Figure 4.6 the prerequisites and actions of ENROL that apply to all students and all courses. This is in accord with our proposal that one ought to describe first the more general classes, and hence the constraints which apply to most objects.

Not-taken-before?	A student cannot take the same course more than once.
Permission?	An undergraduate student requires the permission of the instructor before taking a graduate course.
Part-time-min.?	Part-time students need not take any courses in any particular year.
At-least-4th-year?	Graduate students cannot take first, second, or third year courses.
Not-full?	A student cannot enroll in a course whose enrollment limit has been reached.
Undergrad-min.?	A full-time undergraduate must take at least 5 courses each year.
Undergrad-max.?	A full-time undergraduate may not take more than 6 courses in one year.
Not-excluded?	There exist groups of mutually exclusive undergraduate courses and an undergraduate may take at most one course from such a group.
Before-deadline?	Undergraduates must register in courses by October 13th.
Offered?	A student cannot take a course that is not offered at the time requested.
Has-preparation?	An undergraduate course may have prerequisites that an undergraduate student must have taken in previous terms.
Part-time-max.?	Part-time students may not take more than 3 courses a year.
Areas-OK?	A graduate Computer Science student must have taken courses in each of three major areas.
Another-1st-year?	At most 6 First Year courses may be counted toward the 22 required for a B.Sc. degree.

Specialist?	Arts & Science students desiring a specialist's degree must have taken the appropriate selection of courses.
Has-coreqs?	Certain undergraduate courses may require undergraduates to take other courses at the same time (*e.g.*, in mathematics).
Probation-max.?	An undergraduate student on probation may take no more than 5 courses a year.
Grad-max?	Graduate students should not take more than 6 half-courses a year.

Figure 4.5 Restrictions on Enrolling in Computer Science Courses

```
ENROL (s: STUDENT, c: COURSE);
  prerequisites
    Offered?;
    Not-full?;
    Not-taken-before?;
  actions
     a1: add c to the list taking-courses of s;
     a2: add s to the enrollment-list of c;
     a3: increment size of c by 1;
end ENROL;
```

Figure 4.6 Most General Definition of ENROL

In order to introduce further details about ENROL, we can first describe two subclasses of STUDENT, namely GRAD_STUDENT and UNDERGRAD_STUDENT. Since one of the properties of ENROL is the parameter *s*, supposedly an instance of STUDENT, we can describe two subclasses of ENROL: one for which *s* is restricted to be an instance of GRAD_STUDENT, another for which *s* is an instance of UNDERGRAD_STUDENT. In each case, we can introduce further constraints that must be checked before enrolling a student in a course (see Figure 4.7).

Similarly, we can distinguish subclass GRAD_COURSE and UNDERGRAD_COURSE of COURSE, and in the case of graduate courses we have a number of additional actions to be done in ENROL, as illustrated in Figure 4.8.

There are two important points to note here about the interpretation of ENROL ("bilbo," "cs100"), assuming "bilbo" is an instance of UNDERGRAD_STUDENT and "cs100" is an instance of GRAD_COURSE: by inheritance, this activity will have all the properties of ENROL(*s*: STUDENT, *c*: COURSE), ENROL(*s*: UNDERGRAD_STUDENT, *c*: COURSE), and ENROL(*s*: STUDENT, *c*: GRAD_COURSE), and by an obviously useful convention, all inherited prerequisites will be checked before any of the inherited actions will be executed. In this example multiple inheritance is obviously a useful tool for the designer of a system.

```
specialize ENROL(s: UNDERGRAD_STUDENT, c: COURSE);
   add
      prerequisites
        Before-deadline?;
        Not-too-many: Undergrad-max?;
end;

specialize ENROL (s: GRAD_STUDENT, c: COURSE);
   add
      prerequisites
        At-least-4th-year?;
        Areas-OK?;
        Not-too-many: Grad-max?;
      actions
        a4: inform School of Graduate Studies
                about the enrollment;
end;
```

Figure 4.7 *Some Specializations of ENROL*

```
specialize ENROL (s: STUDENT, c: GRAD_COURSE);
   add
      actions
        a6: issue to s a key to the library;
        a7: give s an unlimited $ computer account;
end;
```

Figure 4.8 *Another Specialization of ENROL*

Resuming the task of introducing the constraints in Figure 4.5, we can now consider specialization of ENROL where more than one parameter is specialized. As a result, we add additional constraints on ENROL(*s*: UNDERGRAD_STUDENT, *c*: UNDERGRAD_COURSE), ENROL(*s*: UNDERGRAD_STUDENT, *c*: GRAD_COURSE) and ENROL (*s*: GRAD_STUDENT, *c*: UNDERGRAD_COURSE), as in Figure 4.9.

Finally, by creating additional subclasses PART_TIME_STUDENT and STUDENT_ON_PROBATION of UNDERGRAD_STUDENT, we specialize the *Not-too-many?* prerequisite to *Part-time-max?* and *Probation-max?* respectively.

```
specialize ENROL(s: UNDERGRAD_STUDENT, c: GRAD_COURSE);
 add
    prerequisites
      Permission?;
end;

specialize ENROL(s: UNDERGRAD_STUDENT, c: UNDERGRAD_COURSE);
 add
    prerequisites
      Has-preparation?;
      Not-excluded?;
      Another-1st-year?;
end;

specialize ENROL(s: GRAD_STUDENT, c: UNDERGRAD_COURSE);
 add
    prerequisites
      At-least-4th-year?;
end;
```

Figure 4.9 Further Specializations of ENROL

A number of remarks are in order at this point. First, note that some restrictions, such as the minimum number of courses that a student must take, cannot be checked until a student has enrolled in all the courses he or she was going to take, and hence these restrictions should not be placed in the ENROL transaction. Instead, one may have a REGISTER transaction which requires as a parameter the list of courses that the student intends to take in that year, and the above conditions would then be prerequisites on this list of courses. Alternatively, after a certain date has passed, a transaction could be run automatically to check such constraints on the list of courses each student is taking. Secondly, note that since we treat data and transactions in a uniform manner, there is no reason why the conditions in Figure 4.5 could not be considered as invariant assertions for the classes of objects STUDENT and COURSE. In particular, they could be grouped around the *class-list* property of COURSE and properties *taking-courses* and *courses-taken* of STUDENT, and they could be introduced in a manner similar to the one used by describing the appropriate subclasses of STUDENT and COURSE. In this case however, violations of the constraints would be detected when the ENROL procedure attempts to insert a new element in the *taking-courses* list of a student, rather than at the time ENROL is originally invoked.

5. Scripts

As we have remarked in the previous section, not all conditions in Figure 4.5 can be accounted for as prerequisites of the ENROL transaction (for example: checking for co-requisites or for conditions involving a minimum number of courses where one needs to know what other courses the student is or will be enrolling in this year). Furthermore, a central attribute of many systems is the ability to communicate interactively with its users in order to obtain the data which "drives" the transactions. We must therefore be able to specify the communication protocols that make up the user interfaces.

For the above purposes we propose *scripts* (generalized processes that have elaborate communication and synchronization mechanisms for the system designer). The script formalism used is an adaptation of Zisman's Augmented Petri Nets [ZISM78] proposed for office automation systems and it is described in more detail in [BARR80]. Each script is essentially a Petri net that has parameters, local variables, and state transitions. In turn, each transition consists of conditions that must be true in order for the transition to fire, and actions that are to be carried out if the transition does fire. In order to enable communication, scripts can employ operators for *message passing* between a script and a terminal, or, more generally, between any two scripts. These operators are based on Hoare's primitives *give* and *take* [HOAR78], and they provide further ability for synchronization, especially when the clock is allowed to send "wake up" messages at desired times. Although much more elaborate communication mechanisms are being currently developed, we will consider here a message as simply a *form* that has text and slots that can be filled by the user (or some other script) and then sent off. (See [TSIC82] for a detailed discussion of the utility of forms as communication means in an office environment.)

To illustrate the use of scripts let us place *enrolling into courses* into a wider context. At our university, students register first with the university. This includes paying fees, selecting a program of studies that is "correct," *etc.* However, students take courses directly from the departments offering them, thus allowing the departments to have direct contact with students in order, for example, to sell them required lecture notes, lab materials, *etc.* Consider therefore the TAKE_COURSE script which describes the protocol for taking a course (see Figure 5.1).

The script is parameterized by the department *d*, which is supposedly offering the course, and it includes five states represented by circles and five state transitions represented by vertical bars. Each transition has associated conditions and actions, and these are separated by => on the diagrams. An instance of the TAKE_COURSE script is created by the secretary of the appropriate department whenever a student enrolls in a course. The initial transition on the script requests a description of the student *s* and the course *c*, which are properties of the script. Once this information has been received, the script proceeds with the process of enrolling the student for the course, and this includes expecting a grade from the instructor of the course. At the same time, the script is set up to expect and act on a "drop the course" request at any time while the student is taking the course. Following normal procedures for enrolling, once the student and the course are identified the script invokes the ENROL transaction and then awaits the message indicating the grade that the student has received in the course. We remark that this script "lives" until the final state is reached, which may be several months later, and that every student would have several such scripts, one for each course he is taking, thus requiring sophisticated use of the database for maintaining all this information.

```
parameters
  d : department;
locals
  s : student;
  c : course;
```

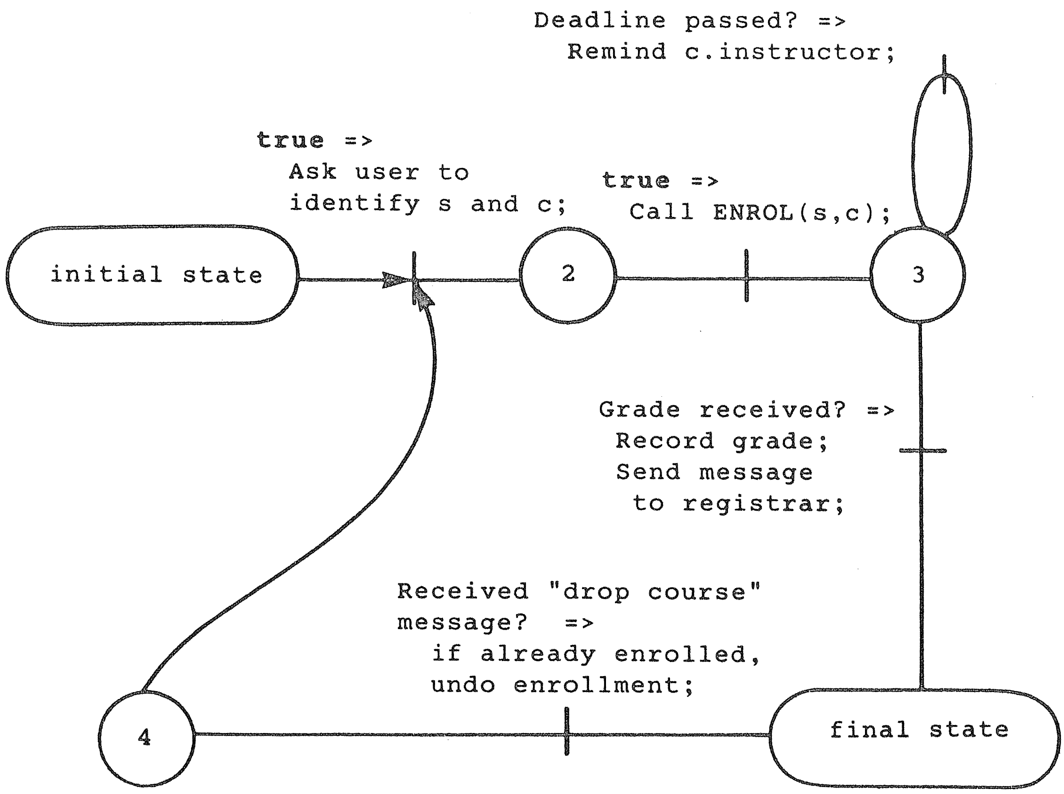


Figure 5.1 Diagram of the TAKE_COURSE Script

Scripts, like all other constructs, are treated as classes in our methodology. Thus their bodies (*i.e.*, their states and transitions), and also all other information associated with their definition is specified through properties that link a script to other classes. (Figure 5.2 shows the definition of the TAKE_COURSE script.)

```
script class TAKE_COURSE(d)
  parameters
    d:  DEPARTMENT;
  locals
    s:  STUDENT;
    c:  COURSE;
    grade:  {0..100};

  states
    initial:  initial_state;
    final:  final_state;
    others:  state_2, state_3, state_4;

  transitions
    obtain information:
      from initial state;
      to state_2, state_4;
      conditions   none
      actions   get s and c from user;
    enrollment:
      from state_2;
      to state_3;
      conditions   none;
      actions   call ENROL(s,c);
    late-grade:
      from state_3;
      to state_3;
      conditions deadline for grade for s,c passed?
      actions send message to instructor of c
    have-grade:
      from state_3;
      to state_4;
      conditions   sent a grade?
      actions   record grade and send message to
                          registrar;
    drop course:
      from state_4;
      to final state;
      conditions   sent "drop course" message?;
      actions   undo enrollment;
end TAKE_COURSE
```

Figure 5.2 Textual Description of TAKE_COURSE Script

It follows that transitions can be specialized in a manner similar to transactions, and more generally one can add new states and transitions in order to create a script that applies in more restricted circumstances than a given script. For example, the Engineering departments may require a mid-term mark to be recorded and mailed to each student;

this could be accomplished by specializing TAKE_COURSE (*d*: DEPARTMENT) to TAKE_COURSE (*d*: ENGINEERING_DEPT) by adding the script in Figure 5.3.

```
                true => call ENROL(s,c);

. . . 2                                          3

                      5
                            Received mid-term grade? =>
                              Record mid-term grade;

   Deadline for mid-term passed?  =>
     Remind c.instructor;
```

Figure 5.3 *Additions for TAKE_COURSE(d:ENGINEERING_DEPT)*

Before leaving this section we present a second example of a script that models the sequence of events from the time an undergraduate student registers for his *n*th ($1 \leqslant n \leqslant 4$) year of (university) study until the time he completes it (Figure 5.4). The basic protocol described by the script involves accepting information about the student, followed by the courses he proposes to take, then waiting for the grades he/she is assigned in these courses, and finally determining that the student has completed the year satisfactorily. Secondary protocols are also defined to take care of such contingencies as withdrawal from the program or late arrival of grades.

To summarize, scripts are useful in enforcing dynamic integrity constraints on transaction call sequences (*e.g.,* one can't receive grades until enrolled in a course), and in defining the format and the protocol of interactions with users. They are a natural place for exception-handling, including exceptions arising because of time delays, as will be seen in the next section.

For simple examples, the introduction of scripts as modelling tools may seem heavy-handed and perhaps unnecessary. There is evidence, however, that designing the environment within which a system will run, including its user interfaces, is one of the thorniest problems facing a system designer [CM79]. We consider this observation a sufficient justification of the introduction and use of scripts within our modelling framework.

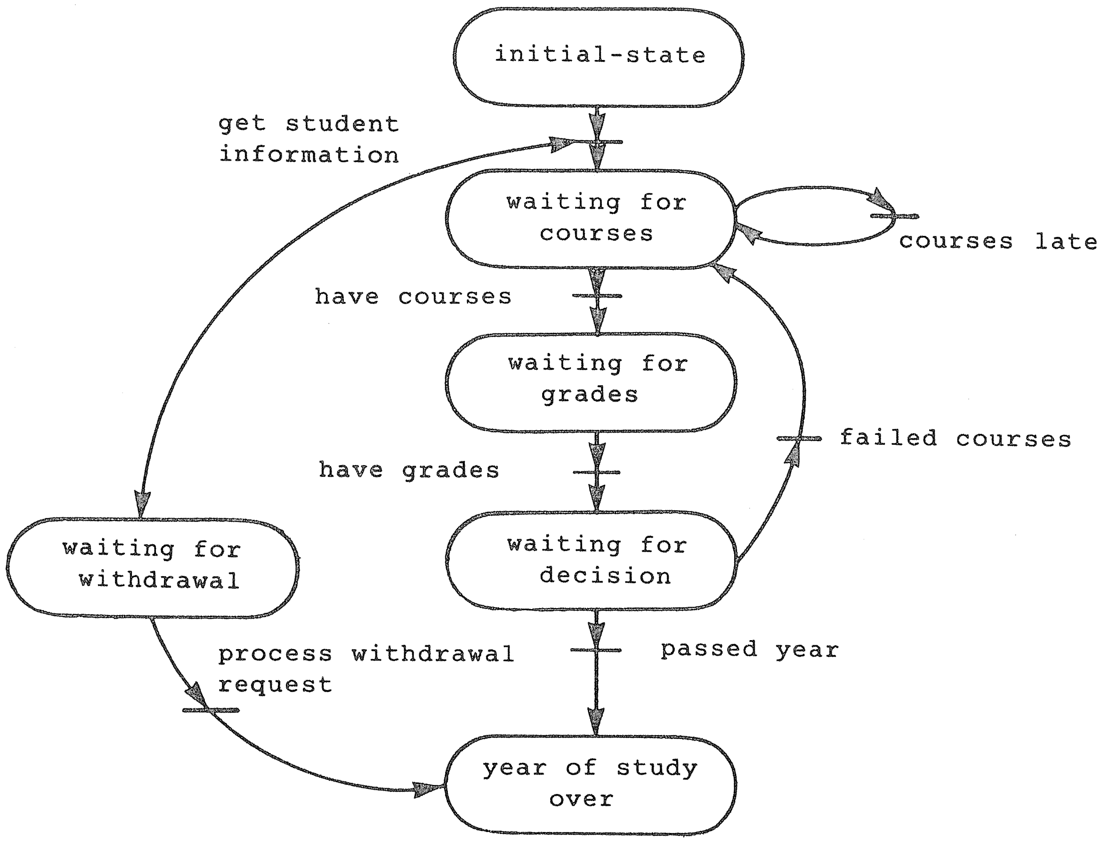


Figure 5.4 *The YEAR_OF_STUDY Script*

6. Exceptions

The ability to *manage exceptions* (*i.e.*, deal with over-abstraction) is characteristic of human behaviour and, we believe, of central importance in managing a multitude of details. Until recently this ability has been noticeably absent from computer application software. In our case, the traditional view dictates that a transaction such as ENROL should be aborted, hopefully with at least an error message, if any of its constraints (*e.g.*, prerequisites or restrictions on parameters) were not met. Alternatively, one might replace the prerequisites by successive IF-THEN conditionals, specifying in each case the course of action to be taken if the constraint were false. In addition to the limitation that this imposes on handling exceptions (see [LEVI77]), it appears to run counter to the "natural" flow of description: one has to constantly take detours from describing how students usually enroll in courses in order to say what is to be done in rare special cases. A more palatable alternative appears to be to adopt the convention that whenever a constraint (such as a prerequisite) evaluates to false, an *exception object* is raised (*i.e.*, is inserted as an instance of a special class). One can then describe in a separate pass the ways in which exceptions are to be handled. Furthermore, exceptions and exception-handlers can be described using the same methodology of concept specialization.

In order to simplify the discussion, we assume in our example that whenever a prerequisite is not satisfied, an instance of the exception labelled by the negation of the condition is raised (*e.g.,* if *Not-full?* is false then the exception *Full* is raised).

One possibility in specifying exceptions is to proceed methodically down the specialization hierarchy of ENROLs and specify which exceptions are raised when different conditions are violated. Alternatively, we may want to organize exceptions, including those raised by conditions that could not be checked in ENROL, into a specialization hierarchy organized along different lines than that of ENROL.[3] Figure 6.1 illustrates a hierarchy of exception classes constructed by answering the question "What can go wrong with enrolling in a course?" at various levels of generality.

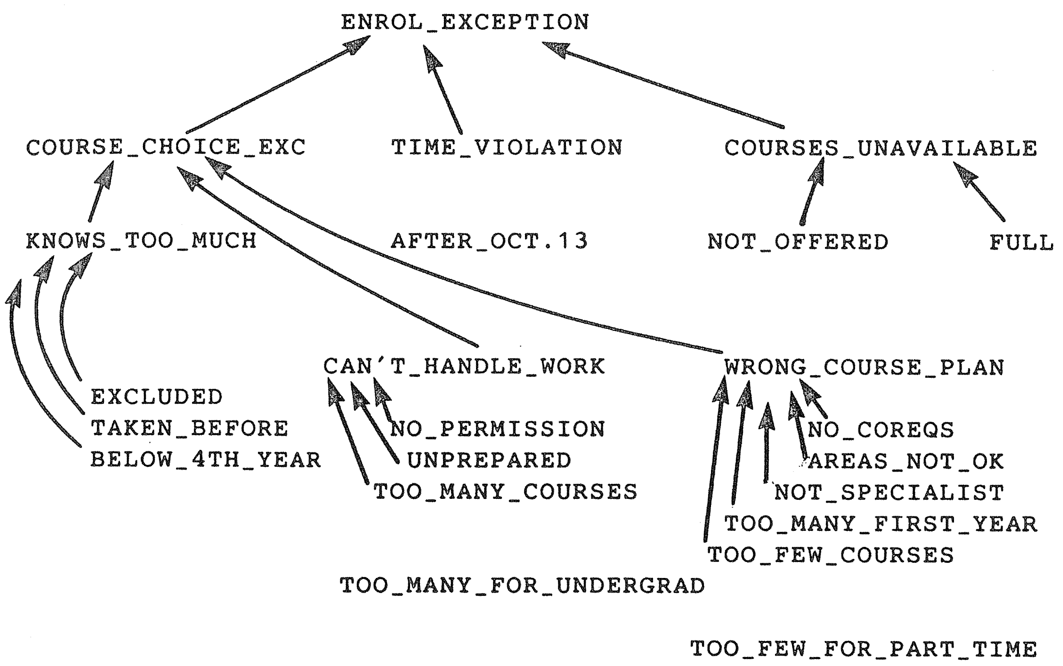


Figure 6.1 *Specialization Hierarchy for Exceptions*

The topmost exception class on the hierarchy, ENROL_EXCEPTION is intended simply to signal that an exception was raised during the execution of an ENROL transaction. Turning to its immediate subclasses, COURSES_UNAVAILABLE indicates that the course cannot be accommodated at the university, so one must abort the ENROL that caused the exception. Alternatively, the system may give the student

[3] Clearly, in doing this type design, computer aids are imperative when one checks to see whether all prerequisites have been accounted for.

information about when the course will next be given, and it might put the student on a waiting list before aborting. For TIMING_VIOLATIONS, students can petition to a special committee in order to be allowed to register late. In an automated office environment this might result in the ENROL being suspended while a PETITION is taking place. As far as COURSE_CHOICE_EX is concerned, there is nothing we can say that is applicable in all cases; one of the advantages of our methodology over more traditional approaches such as decision tables is that we do not have to say anything if we have no new information to add. However, considering subclasses of COURSE_CHOICE_EX, we note that instances of KNOW_TOO_MUCH should always be handled by "abort," since students should not be allowed to pick up credits gratis, while instances of CAN'T_HANDLE_WORK require petitioning again. In neither case need we say anything more about the subclasses of these two types of exceptions. Finally, the WRONG_COURSE_PLAN exceptions require distinctive individual treatment. For NO_COREQUISITE, the student must take the other course. For TOO_FEW_COURSES, he must take other courses, *etc.*

Turning to exception-handling, we assume that for each exception raised within a transaction *T*, the exception-handler, another transaction or script, is specified by the caller of *T*. The following specifications illustrate the exception-handling mentioned in the first half of this section.

To start with, we specify in TAKE_COURSE (*d*: DEPARTMENT), and, in particular, in association with the call to the ENROL transaction, the exception-handler to be used in case of an ENROL_EXCEPTION (Figure 6.2(a)). As indicated earlier, at this level exception-handling simply consists of a message to the user naming the exception that has been raised. Let's call this most general exception-handler EX_HANDLER (*e*: ENROL_EXCEPTION), shown in Figure 6.2(b). EX_HANDLER assumes that its exception argument has two associated properties, *std* and *crs*, through which one can determine the student and the course for which the exception was raised.

Dealing with some of the more specialized exceptions involves the adding of actions to be carried out by EX_HANDLER. For example, EX_HANDLER (*e*: COURSE_UNAVAILABLE) may output, in addition to the exception message, another message that specifies when the course is given in the future, and then abort the attempted enrollment (see Figure 6.3).

Other exceptions require additional states and transitions in EX_HANDLER. Thus TIMING_VIOLATION exceptions involve petitioning a committee and may eventually result in an enrollment (see Figure 6.4).

```
script class TAKE_COURSE(d:  DEPARTMENT)
  ....
  enrollment:
   ....
   actions call ENROL(s,c)
     for exception e∈ENROL_EXCEPTION with std←s, crs←c
     use EX_HANDLER(e)
     ....

     Modified enrollment transition in TAKE_COURSE
```

(a)

```
script class EX_HANDLER(e)
  parameters
    e:  ENROL_EXCEPTION;
  states
    initial:  initial_state;
    final:  final_state;
    others:  none;
  transitions
      send message;
        from initial state;
        to final state;
        conditions   none
        actions   send Class_of(e).message
  end EX_HANDLER;
```

Class_of(e) evaluates to the most specialized class **e** is an instance of, say C, and C.message evaluates to a(n error) message associated with that class as an attribute.

(b)

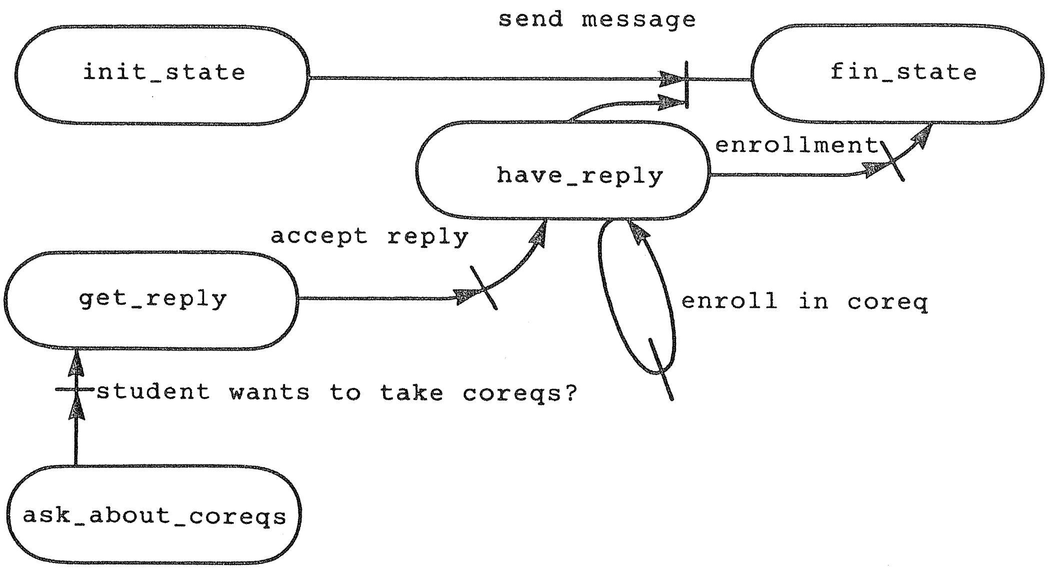
Figure 6.2 *Raising and Handling Exceptions*

```
specialize EX_HANDLER (e: COURSE_UNAVAILABLE);
 transitions
  send message:
   actions
     inform user when the course is given next;
     send "drop course" message to TAKE_COURSE
          (std,crs);
end;
```

Figure 6.3 *Specialization of an Exception-Handler*

```
specialize EX_HANDLER (e:  TIMING_VIOLATION);
  locals
    reply:  {YES,NO};
  states
    others:  awaiting_reply;
  transitions
    send message;
      from initial_state;
      to awaiting_reply;
      conditions none;
      actions
           send type (e).message;
           send petition (e.std, e.crs) to committee script;
    positive reply;
      from awaiting_reply;
      to final_state;
      conditions sent reply = YES?
      actions
           inform user;
           call ENROL(e.std,e.crs);
    negative reply;
      from awaiting_reply;
      to final_state;
      conditions sent reply = NO?
      actions
           inform user;
           send "drop course" message to TAKE_COURSE(std, crs);
end;
```

Figure 6.4 *Another Specialization of EX_HANDLER*



Ask_about_coreqs is also an initial state. Transition send message only fires if the reply has been negative; otherwise, enrollment fires.

Figure 6.5 *Exception Handler for Course Corequisites*

Among COURSE_CHOICE_EX exceptions, only WRONG_COURSE_PLAN requires special exception-handling facilities. For NO_COREQS exceptions, for example, EX_HANDLER must communicate with the user to determine whether the student is willing to enroll in all these courses. If he is, EX_HANDLER carries out all such enrollments and then proceeds with the enrollment originally requested. This specialization of EX_HANDLER is shown graphically in Figure 6.5.

We conclude this section by noting that the ability to describe scripts, exceptions, and exception-handlers within the same framework as provided for the normal data and transaction classes gives a pleasing uniformity and conceptual parsimony to the proposed methodology.

7. Conclusions

We have outlined and illustrated the principal elements of a software specification methodology that combines stepwise refinement by decomposition with concept specialization in order to introduce the multitude of details typically associated with large interactive systems. To recapitulate, the methodology suggests that the designer should start by defining the most general naturally occurring classes of objects and events in the domain. This is to be accomplished by the use of named properties that connect related concepts, and by the use of assertions that restrict the potential relationships. Further details of the proposed system are then introduced in successive iterations by describing subclasses of already presented classes and specializing transactions in order to deal with the objects in these classes.[4] The result is a hierarchy (taxonomy) of data, transaction, and script classes on which inheritance operates to abbreviate natural redundancy without losing the benefit of being able to check consistency. Once the usual/normal aspects of the system are described to some level of detail, the designer can describe, using the same methodology, the exceptions raised by the failure of assertions and their handling mechanism.

In evaluating the methodology, we feel that it is conducive to a natural style of description because it is oriented toward the *conceptual object* and *activities* occurring in the user's world. Our heuristics for identifying classes, and the suggestion of describing first general classes and then more specialized subclasses, provide some needed guidance to the designer. Similarly, the virtual specialization hierarchy that results

[4] Of course, this does not prevent one from introducing at any stage new classes, transactions, and scripts as they are needed.

when considering the possible specializations of each parameter for a transaction is a convenient conceptual rack on which to hang the details of the problem domain. In addition to its role of abridging descriptions, multiple inheritance, as illustrated in our example, allows one to think separately about independent aspects of the world (*e.g.*, undergraduate students and graduate courses) with inheritance taking care of their interaction. Finally, we feel that the systematic treatment of exceptions and exception-handlers within the same framework of data, transaction, and script classes supports another important abstraction principle: the ability to disregard the exceptional or unusual situations during the first pass in the design.

By developing program specifications according to the above methodology, one also gains some advantage in verifying the correctness of the final system. For example, having verified that a "general" transaction (*i.e.*, one high in the generalization hierarchy) maintains an invariant, one can often (because of inheritance) reuse this proof in demonstrating that the various specializations of the transaction also maintain the invariant (see [WONG81] [BORG81]).

Two other chapters in this book, those by King and McLeod and Brodie and Ridjanovic, also address the problem of designing complete database systems; hence a brief comparison of the three approaches is in order.

To begin with, there are a number of striking similarities in the general philosophy of the approaches taken, similarities due in no small part to the principles expressed in the title of this book: "conceptual modelling." Thus all three chapters start the design process with a conceptual model of the enterprise as seen by the system's eventual users. This model is meant to capture as much of the *semantics* of the real world as possible, certainly more than in traditional database design; in other words, all three chapters would be classified as work in "semantic/conceptual data models." Among others, this leads to an emphasis on modelling entities and their semantic relationships rather than on pure data organization. In a departure from most other semantic data models, all three emphasize the importance of modelling the *dynamic/behavioural*, not just the static parts of an enterprise, and the need to *integrate* these two facets of the description. Furthermore, all three chapters recognize the difficulties that arise in designing large, complex systems, and hence they emphasize the importance of a *methodology of design* that is inseparably linked to the modelling features offered. As a natural extension to this concern, the three research groups also offer a variety of computer tools that are meant to assist the designer in achieving a complete and accurate design.

Among the notable general differences are the fact that TAXIS, at least as presented here, focuses on design at one level only, while both the others consider design at several levels of detail. Thus in ACM/

PCM (see the chapter by Brodie and Ridjanovic), there is a general graphical schema, a more precise predicate-based technique for specification, and finally a functional technique for full details, wherever desired, while the "event model" of King and McLeod has an initial design schema, which drives the building of a conceptual schema, which in turn forms the basis of a physical design. On the other hand, through the notions of scripts, messages, exceptions, and exception-handlers, TAXIS probably addresses in detail a wider variety of aspects of an information system, though we should point out that the event model does model at least part of what scripts are intended to accomplish.

Although both this chapter and that of Brodie and Ridjanovic look for *uniformity* in the way objects and activities are modeled, they come up with different answers. ACM/PCM sees association of objects as paralleling iteration, and specialization of concepts corresponding to choice, while TAXIS's notion of iteration is not related to the abstraction principles, and specialization of transactions is quite similar in spirit to specialization of entities. A different attempt at uniformity shows up in the chapter of King and McLeod, in their novel attempt at incorporating the *modelling events* themselves into the model, thus bringing program maintenance into the same uniform framework.

Finally, the chapters can be distinguished by the basic metaphors which in some sense "drive" the design process. In the event model, the design schema describes mostly events and their interactions, and this drives the process of describing entities, *etc.* In contrast, the other chapters use the structure of the data descriptions to "drive" the description of the activities. In ACM/PCM, this is evident from the way that the actions associated with an application object are determined by its structure — the "context" (*e.g.*, in the Hotel-reservation actions). In TAXIS, on the other hand, the hierarchy of data classes "drives" the specialization of transactions, while the hierarchy of exceptions drives that of exception-handlers.

There are, of course, many other comparisons that could be drawn, but space limits us to those which we feel are most significant.

To conclude, we reiterate our belief that taxonomic organization is an essential human activity that allows us to cope with multitudes of detail. Our goal is to propose linguistic and computer tools that would support precisely such an organization during the development of a software system. Evidence of such tools can be seen in [BMW82] and in forthcoming MSc theses by B. Nixon and P. O'Brien. Since, in the end, the only demonstration of the importance of an idea is its successful practical use, our group has attempted to model in significant detail a number of applications in the university and hospital environments, with results presented in [WONG81] and in forthcoming theses by C. Di Marco and I. Buchan. Finally, research is still in progress on the use of these ideas for general requirements specification and for designing the language of interaction between users and a specific system.

The authors gratefully acknowledge the permission of North Holland Publishing Company to reproduce portions of an earlier version of this paper.

8. References

[ABRI74] [BD81] [BARR80] [BG80a] [BC75] [BORG81] [BORG82a] [BORG82b] [BW81] [BMW82] [CM79] [DH72] [DIJK72] [GBM82] [HM75] [HBS73] [HOAR78] [INGA78] [LEVI77] [LSAS77] [MS81] [MYLO81] [MBW80] [PARN72] [QUIL68] [SS79] [TSIC82] [TL82] [WASS77] [WILS75] [WIRT71] [WONG81] [WLS76] [ZISM78]

References

[ABRI74]
Abrial, J.R., "Data Semantics," in J.W. Klimbie and K.L. Koffeman (eds.), *Data Management Systems,* North-Holland, Amsterdam, The Netherlands, 1974.

[ACC82]
Atkinson, M., K. Chisholm, and P. Cockshott, "PS-ALGOL: An ALGOL with a Persistent Heap," *SIGPLAN Notices,* Vol. 17, No. 7, July 1982.

[AGBB77]
Ambler, A.L., D.I. Good, J.C. Browne, W.F. Burger, R.M. Cohen, C.G. Hoch and R.E. Wells, "Gypsy: A Language for Specification and Implementation of Verifiable Programs," *SIGPLAN Notices,* Vol. 12, No. 3, March 1977, pp. 1-10.

[AGP78]
Arvind, (no initial), K.P. Gostelow, and W. Plouffe, "The (Preliminary) Id Report," Technical Report 114, Dept. of Information and Computer Science, Univ. of California, Irvine, 1978.

[AI80]
Artificial Intelligence, Special Issue on Non-Monotonic Logic, D. Bobrow (ed.), Vol. 13, Nos. 1 and 2, April 1980.

[AM81]
Ariav, G., and H.L. Morgan, "MDM: Handling the Time Dimension in Generalized DBMSs," Decision Sciences Working Paper 81-05-06, Wharton School, Univ. of Pennsylvania, 1981.

[ANSI75]
ANSI/X3/SPARC (Standards Planning and Requirements Committee), "Interim Report from the Study Group on Database Management Systems," *FDT* (Bulletin of ACM SIGMOD), Vol. 7, No. 2, 1975.

[AS81]
Attardi, G. and M. Simi, "Semantics of Inheritance and Attributions in the Description System Omega," *Proc. International Joint Conference on Artificial Intelligence*, Vancouver, B.C., Canada, August 1981.

[AU79]
Aho, A. V., and J. D. Ullman, "Universality of Data Retrieval Languages," *Proc. 6th ACM Symposium on Principles of Programming Languages,* 1979.

[AV80]
Apt, K. R., and Van Emden, M. H., "Contributions to the Theory of Logic Programming," Research Report CS-80-12, Dept. of Computer Science, Univ. of Waterloo, Waterloo, Ont., Canada, 1980.

[BACH77]
Bachman, C. W., "The Role Concept in Data Models," *Proc. 3rd International Conference on Very Large Databases,* Tokyo, Japan, 1977.

[BACK78]
Backus, J., "Can Programming be Liberated from the von Neumann Style?," *Communications of the ACM,* Vol. 21, No. 8 August 1978, pp. 613-641.

[BALZ81]
Balzer, R., "Transformational Implementation: An Example," *IEEE Transactions on Software Engineering,* Vol. SE-7, No. 1, January 1981.

[BARB82]
Barber, G. R., "Office Semantics," Ph.D. thesis, Massachusetts Institute of Technology, 1982.

[BARR80]
Barron, J. L., Dialogue Organization and Structure for Interactive Information Systems, Master's thesis, (CSRG Technical Report) Dept. of Computer Science, Univ. of Toronto, January 1980.

[BARS77]
Barstow, D. R., "Automatic Construction of Algorithms and Data Structures Using A Knowledge Base of Programming Rules," Stanford AIM-308, November 1977.

[BARW81]
Barwise, J., "Some Computational Aspects of Situation Semantics (Abstract)," Unpublished manuscript, 1981.

[BBC80]
Bernstein, P. A., B. T. Blaustein, and E. M. Clarke, "Fast Maintenance of Integrity Assertions Using Redundant Aggregate Data," *Proc. 6th International Conference on Very Large Databases,* Montreal, Que., Canada, October 1980.

[BBD77]
Bell, M. L., D. C. Bixler, and M. E. Dyer, "An Extendible Approach to Computer-Aided Software Requirements Engineering," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1977.

[BBDG81a]
Bauer, F. L., M. Broy, W. Dosch, R. Gnatz, F. Geiselbrechtinger, W. Hesse, B. Krieg-Brueckner, A. Laut, T. A. Matzner, B. Moeller, H. Partsch, P. Pepper, K. Samelson, M. Wirsing, H. Woessner, "Report on a Wide Spectrum Language for Program Specification and Development" (tentative version), Institut fuer Informatik der TU

Muenchen, TUM-I8104, 1981; also in: *Lecture Notes in Comptuer Science,* Springer-Verlag, New York, 1981.

[BBDG81b]
Bauer, F.L., M. Broy, W. Dosch, R. Gnatz, B. Krieg-Brueckner, A. Laut, M. Luckmann, T.A. Matzner, B. Moeller, H. Partsch, P. Pepper, K. Samelson, R. Steinbrueggen, M. Wirsing, H. Woessner, "Programming in a Wide Spectrum Language: a Collection of Examples," *Science of Computer Programming* Vol. 1, 1981, pp. 73-114.

[BBG78]
Beeri, C., P.A. Bernstein, and N. Goodman, "A Sophisticate's Introduction to Database Normalization Theory," *Proc. 4th International Conference on Very Large Databases,* West Berlin, September 1978.

[BC75]
Bobrow, D., and A. Collins, (eds.), *Representation and Understanding,* Academic Press, New York, 1975.

[BD77]
Burstall, R.M., and J.L. Darlington, "A Transformation System for Developing Recursive Programs," *Journal of the ACM,* Vol. 24, No. 1, January 1977.

[BD81]
Barr, A., and J. Davidson, "Representation of Knowledge," in A. Barr and E. Feigenbaum (eds.), *Handbook of Artificial Intelligence,* William Kaufmann Inc., 1981.

[BDMN73]
Birtwistle, G.M., O.-J. Dahl, B. Myhrhaug, K. Nygaard, *Simula Begin,* Van Nostrand Reinhold, New York, 1973.

[BF79]
Buneman, O.P., and R.E. Frankel, "FQL—A Functional Query Language," *Proc. 1979 ACM SIGMOD International Conference on the Management of Data,* Boston, Mass., May 1979.

[BFN81]
Buneman, O.P., R.E. Frankel, and R. Nikhil, "A Practical Functional Programming System for Databases," *Proc. ACM Conference on Functional Programming and Archictecture,* New Hampshire, 1981.

[BFN82]
Buneman, O.P., R.E. Frankel, and R. Nikhil, "An Implementation Technique for Database Query Languages," *ACM Transactions on Database Systems,* Vol. 7, No. 2, June 1982.

[BG80a]
Bobrow, D. and Goldstein, I., "Representing Design Alternatives," *Proc. Society for Study of Artificial Intelligence and Simulation of Behavior Conference,* Amsterdam, The Netherlands, July 1980.

[BG80b]
Burstall, R.M., and J.A. Gougen, "The Semantics of CLEAR: A Specification Language," *Proc. Copenhagen Winter School on Abstract Software Specification,* Copenhagen, Denmark, 1980.

[BHR80]
Bayer, R., H. Heller, and A. Reiser, "Parallelism and Recovery in Database Systems," *ACM Transactions on Database Systems,* Vol. 5, No. 2, June 1980.

[BI82]
Borning, A.H., and D.H. Ingalls, "Multiple Inheritance in Smalltalk-80," *Proc. AAAI National Conference,* Pittsburgh, Penn., August 1982.

[BISK81]
Biskup, J., "Null Values in Data Base Relations," in [GM78].

[BJ78]
Bjorner, D., and C.B. Jones, *The Vienna Development Method,* Springer-Verlag, New York, 1978.

[BL82]
Brachman, R.J., and H. Levesque, "Competence in Knowledge Representation," *Proc. AAAI National Conference*, Pittsburgh, Penn., August 1982, pp. 189-192.

[BM76]
Basu, S., and J. Misra, "Some Classes of Naturally Provable Programs," *2nd International Conference on Software Engineering,* San Francisco, October 1976.

[BM77]
Buneman, O.P., and H.L. Morgan, "Alerting Techniques for Database Systems," *IEEE COMPSAC Conference,* Chicago, Ill., November 1977.

[BMS80]
Burstall, R.M., D.B. MacQueen, and D.T. Sanella, "HOPE: an Experimental Applicative Language," *Proc. Lisp Conference,* Stanford, Calif., 1980.

[BMW82]
Borgida, A.T., J. Mylopoulos, and H.K.T. Wong, "Methodological and Computer Aids for Interactive Information System Design," in H.J. Schneider and A. Wasserman (eds.), *Automated Tools for Information System Design—Proc. of IFIP Conference,* North-Holland, Amsterdam, The Netherlands, 1982.

[BOBR75]
Bobrow, D.G., "Dimensions of Representations," in [BC75].

[BOBR77]
Bobrow, D.G., "A Panel on Knowledge Representation," *Proc. 5th International Joint Conference on Artificial Intelligence,* Cambridge, Mass., August 1977.

[BORG81]
Borgida, A.T., "On the Definition of Specialization Hierarchies for Procedures," *Proc. 7th International Joint Conference on Artificial Intelligence,* Vancouver, B.C., Canada, August 1981.

[BORG82a]
Borgida, A. T., "Conceptual Modeling for Information System Development," *Proc. 1st AUC Conference on Databases,* Medellin, Colombia, August 1982.

[BORG82b]
Borgida, A. T., "Prospectus for Research on Flexible Information Systems," Dept. of Computer Science, Rutgers Univ., 1982.

[BOWL77]
Bowles, K. L., *Microcomputer Problem Solving Using Pascal,* Springer-Verlag, New York, 1977.

[BP80]
Barwise, J., and J. Perry, "The Situation Underground," unpublished manuscript, 1980.

[BP81a]
Barwise, J., and J. Perry, "Situations and Attitudes," *Journal of Philosophy,* Vol. 78, No. 11, October 1981, pp. 668-691.

[BP81b]
Barwise, J., and J. Perry, "Semantic Innocence and Uncompromising Situations," in P. A. French, T. E. Uehling, Jr., and H. K. Wettstein, (eds.), *The Foundations of Analytic Philosophy,* Midwest Studies in Philosophy, Vol. VI, Univ. of Minnesota Press, Minneapolis, 1981, pp. 387-404.

[BP81c]
Broy, M., and P. Pepper, "Program Development as a Formal Activity," *IEEE Transactions on Software Engineering,* Vol. SE-7, No. 1, January 1980.

[BPP76]
Bracchi, G., P. Paolini, and G. Pelagatti, "Binary Logical Associations in Data Modelling," in J. M. Nijssen (ed.), *Modelling in Database Management Systems* (Proc. IFIP TC2 Conference, Freudenstadt), North-Holland, Amsterdam, The Netherlands, 1976.

[BR70]
Buxton, J. N., and B. Randell (eds.), *Software Engineering Techniques,* NATO, 1970 (report on a conference sponsored by the NATO Science Committee, Rome, Italy, October 27-31, 1969).

[BRAC76]
Brachman, R. J., "A Structural Paradigm for Representing Knowledge," BBN Report No. 3605, Bolt, Beranek and Newman Inc., Cambridge, Mass., 1976.

[BRAC79]
Brachman, R. J., "On the Epistemological Status of Semantic Networks" in [FIND79], pp. 3-50.

[BRAC80a]
Brachman, R. J., "I Lied about the Trees," unpublished manuscript, 1980.

[BRAC80b]
Brachman, R. J., "An Introduction to KL-ONE," in R. J. Brachman, *et al.* (eds.), *Research in Natural Language Understanding, Annual Report (1 Sept. 78-31 Aug. 79)*, Bolt, Beranek and Newman Inc., Cambridge, Mass., 1980, pp. 13-46.

[BRES72]
Bressan, A., *A General Interpreted Modal Calculus*, Yale Univ. Press, New Haven, Conn., 1972.

[BRIN75]
Brinch Hansen, P., "The Programming Language Concurrent Pascal," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 4, June 1975, pp. 199-207.

[BROD78]
Brodie, M. L., "Specification and Verification of Database Semantic Integrity," Ph.D. thesis (Computer Systems Research Group Technical Report No. 91), Univ. of Toronto, April 1978.

[BROD80a]
Brodie, M. L., "The Application of Data Types to Database Semantic Integrity," *Information Systems*, Vol. 5, No. 4, 1980.

[BROD80b]
Brodie, M. L., "Data Abstraction, Databases and Conceptual Modeling," *Proc. 6th International Conference on Very Large Databases*, Montreal, Que., Canada, October 1980.

[BROD80c]
"Data Quality in Information Systems," *Information & Management*, Vol. 3, 1980.

[BROD81a]
Brodie, M. L., "Association: A Database Abstraction for Semantic Modelling," *Proc. 2nd International Entity-Relationship Conference*, Washington, D.C., October 1981.

[BROD81b]
Brodie, M. L., "On Modelling Behavioural Semantics of Data," *Proc. 7th International Conference on Very Large Databases*, Cannes, France, September 1981.

[BROD82]
Brodie, M. L., "Axiomatic Definitions for Data Model Semantics," *Information Systems*, Vol. 7, No. 2, 1982.

[BROO75]
Brooks, F. P., Jr., *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, Reading, Mass., 1975.

[BROW73]
Brown, J. S., "Steps Towards Automatic Theory Formation," *Proc. International Joint Conference on Artificial Intelligence*, Palo Alto, Calif., August 1973.

[BROW80]
Browne, J. C., The Interaction of Operating Systems and Software Engineering," *Proc. IEEE,* Vol. 68, No. 9, September 1980.

[BRUC75]
Bruce, B., "Case Systems for Natural Language," *Artificial Intelligence,* Vol. 6, 1975, pp. 327-360.

[BS78]
Brodie, M. L., and J. W. Schmidt, "What is the Use of Abstract Data Types?," *Proc. 4th International Conference on Very Large Databases,* West Berlin, September 1978.

[BS79]
Bentley, J. L., and M. Shaw, "An Alphard Specification of a Correct and Efficient Transformation on Data Structures," *Proc. IEEE Conference on Specifications of Reliable Software,* April 1979, pp. 222-237.

[BS80]
Brachman, R. J., and B. Smith, Special Issue on Knowledge Representation, *SIGART Newsletter,* No. 50, February 1980.

[BS82a]
Bobrow, D. G., and M. J. Stefik, "Loops: An Object Oriented Programming System for Interlisp," Draft Report, Xerox PARK, 1982.

[BS82b]
Brodie, M. L., and J. W. Schmidt (eds.), "Final Report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group," *SIGMOD Record,* Vol. 12, No. 4, July 1982.

[BS82c]
Brodie, M. L., and E. O. Silva, "Active and Passive Component Modelling: ACM/PCM," in [OSV82], pp. 41-91.

[BSD82]
Byrd, R. J., S. E. Smith, S. P. de Jong, "An Actor-Based Programming System," *SIGOA Conference on Office Information Systems,* June 1982.

[BSR80]
Bernstein, P. A., D. W. Shipman, and J. B. Rothnie, "Concurrency Control in a System for Distributed Databases (SDD-1), *ACM Transactions on Database Systems,* Vol. 5, No. 1, March 1980.

[BUNE79]
Bunemann, O. and E. Clemons, "Efficiently Monitoring Relational Databases," *ACM Transactions on Database Systems,* Vol 4., No. 3, September 1979.

[BURG75]
Burge, W. H., *Recursive Programming Techniques*, Addison-Wesley, Reading, Mass., 1975.

[BW77]
Bobrow, D., and T. Winograd, "An Overview of KRL, a Knowledge Representation Language," *Cognitive Science*, Vol. 1, No. 1, January 1977.

[BW81]
Borgida, A.T., and H.K.T. Wong, "Data Models and Data Manipulation Languages: Complimentary Semantics and Proof Theory," *Proc. 7th International Conference on Very Large Databases*, Cannes, France, September 1981, pp. 260-271.

[BYTE81]
Special Issue on Smalltalk, *BYTE*, August 1981.

[BZ81]
Brodie, M.L., and S.N. Zilles (eds.), *Proc. Workshop on Data Abstraction, Databases, and Conceptual Modelling, SIGART Newsletter*, No. 74, January 1981; *SIGMOD Record*, Vol. 11, No. 2, February 1981; *SIGPLAN Notices*, Vol. 16, No. 1, January 1981.

[CB74]
Chamberlin, D.D., and R.F. Boyce, "SEQUEL: a Structured English Query Language," *Proc. 1974 ACM SIGMOD International Conference on the Management of Data*, 1974.

[CBLL82]
Curry, G., L. Baer, D. Lipkie, B. Lee, "Traits: An Approach to Multiple-Inheritance Subclassing," *Proc. Conference on Office Information Systems, SIGOA Newsletter*, Vol. 3, Nos. 1 and 2, June 1982.

[CHAM75]
Chamberlin, D.D., *et. al.*, "Views, Authorization and Locking in a Relational Data Base System," *Proc. 1975 National Computer Conference*, Anaheim, Calif., May 1975.

[CHAM76]
Chamberlin, D.D., "Relational Database Management Systems," *Computing Surveys*, Vol. 8, No. 1, March 1976.

[CHEA81]
Cheatham, T.E., "Program Refinement by Transformation," *Proc. 5th International Conference on Software Engineering*, San Diego, Calif., March 1981.

[CHEN76]
Chen, P.P.S., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, Vol. 1, No. 1, March 1976.

[CHEN78]
Chen, P.P.S., *The Entity-Relationship Approach to Logical Database Design*, Monograph No. 6, QED Information Sciences, Wellesley, Mass., 1978.

[CHIL77]
Childs, D. L., "Extended Set Theory," *Proc. 3rd International Conference on Very Large Databases,* Tokyo, Japan, October 1977.

[CHUR40]
Church, A., "A Formulation of the Simple Theory of Types," *Journal of Symbolic Logic,* Vol. 5, 1940, pp. 56-68.

[CHUR41]
Church, A., "The Calculi of Lambda-Conversion," *Annals of Mathematics Studies No. 6,* Princeton Univ. Press, 1941.

[CL73]
Chang, C. L., and R. C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving,* Academic Press, New York, 1973.

[CLAR78]
Clark, K. L., "Negation as Failure," in [GM78].

[CLIN81]
Clinger, W. D., "Foundations of Actor Semantics," Technical Report MIT/AI/TR-633, MIT Laboratory for Artificial Intelligence, May 1981.

[CM79]
Carlson, E. and W. Metz, "A Design for Table-Driven Display Generation and Management Systems," Technical Report, IBM Research Laboratory, San Jose, Calif., 1979.

[CODA71]
CODASYL Data Base Task Group, April 1971 Report.

[CODD70]
Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* Vol. 13, No. 6, June 1970, pp. 377-387.

[CODD71]
Codd, E. F., "A Database Sublanguage Founded on the Relational Calculus," *Proc. SIGFIDET Workshop,* San Diego, Calif., 1971.

[CODD72]
Codd, E. F., "Relational Completeness of Database Sublanguages," in R. Rustin (ed.), *Data Base Systems,* Prentice-Hall, Englewood Cliffs, N.J., 1972.

[CODD79]
Codd, E. F., "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems*, Vol. 4, No. 4, December 1979, pp. 397-434; IBM Research Report RJ2599, San Jose, Calif., August 1979.

[CODD82]
Codd, E. F., "Relational Database: A Practical Foundation for Productivity," *Communications of the ACM,* Vol. 25, No. 2, February 1982.

[COHE78]
Cohen, P.R., *On Knowing What to Say: Planning Speech Acts*, Ph.D. thesis (TR-118), Dept. of Computer Science, Univ. of Toronto, 1978.

[CWAM75]
Collins, A., E. Warnock, N. Aiello, and M. Miller, "Reasoning from Incomplete Knowledge," in [BC75].

[DATE81]
Date, C.J., *An Introduction to Database Systems,* 3rd ed., Addison-Wesley, Reading, Mass., 1981.

[DATE83]
Date, C.J., *An Introduction to Database Systems Volume II,* Addison-Wesley, Reading, Mass., 1983.

[DAVI58]
Davis, M., *Computability and Unsolvability,* McGraw-Hill, New York, 1958.

[DAVI77]
Davis, R., "Interactive Transfer of Expertise: Acquisition of New Inference Rules," *Proc. 5th International Joint Conference on Artificial Intelligence,* Cambridge, Mass., August 1977.

[DB82]
Dayal, U., and P.A. Bernstein, "On the Updatability of Network Views—Extending Relational View Theory to the Network Model," *Information Systems,* Vol. 7, No. 1, 1982.

[DD80]
Demers, A.J., and J.E. Donahue, "Data Types, Parameters, and Type Checking," *Proc. ACM Symposium on Principles of Programming Languages,* SIGACT and SIGPLAN, January 1980, pp. 12-23.

[DEJO80]
de Jong, S.P., "The System for Business Automation (SBA): a Unified Application Development System," *Proc. 1980 IFIP Congress,* Tokyo, Japan, 1980.

[DENN74]
Dennis, J.B., "First Version of a Data Flow Procedure Language," *Proc. Symposium on Programming,* Institut de Programmation, Univ. of Paris, Paris, France, April 1974, 241-271.

[DEUT81]
Deutsch, L.P., "In Summary of Workshop Session on Types," *Proc. Workshop on Data Abstraction, Databases, and Conceptual Modelling, SIGPLAN Notices,* Vol. 16, No. 1, January 1981, p. 49.

[DH72]
Dahl, O.-J., and C.A.R. Hoare, "Hierarchical Program Structures," in O.-J. Dahl, E.W. Dijkstra, and C.A.R. Hoare (eds.), *Structured Programming,* Academic Press, New York, 1972, pp. 175-220.

[DH73]
Duda, R. O., and P. E. Hart, *Pattern Classification and Scene Analysis,* Wiley-Interscience, New York, 1973.

[DIJK68]
Dijkstra, E. W., "Goto Statement Considered Harmful," *Communications of the ACM,* Vol. 11, No. 3, March 1968, pp. 147-148.

[DIJK72]
Dijkstra, E. W. "Notes on Structured Programming," in O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare (eds.), *Structured Programming,* Academic Press, New York, 1972.

[DIJK76]
Dijkstra, E. W., *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, N.J., 1976.

[DISE77]
diSessa, A., "On Learnable Representations of Knowledge: A Meaning for the Computational Metaphor," Memo MIT/AIM-441, MIT Laboratory for Artificial Intelligence, September 1977.

[DK75]
Davis, R., and J. King, "An Overview of Production Systems," Memo AIM-271, Stanford Artificial Intelligence Laboratory, 1975.

[DK76]
DeRemer, F., and H. H. Kron, "Programming-in-the-Large vs. Programming-in-the-Small," *IEEE Transactions on Software Engineering,* Vol. SE-2, No. 2, June 1976, pp. 80-86.

[DMN68]
Dahl, O.-J., B. Myrhaug, and K. Nygaard, *Simula 67 Common Base Language,* Pub. S-22, Norwegian Computing Center, Oslo, 1968.

[DN66]
Dahl, O.-J., and K. Nygaard, "SIMULA — An ALGOL-Based Simulation Language," *Communications of the ACM,* Vol. 9, No. 9, September 1966, pp. 671-678.

[DoD78]
Department of Defense, *Steelman Requirements for High Order Computer Programming Languages,* June 1978.

[DoD79]
Department of Defense, *Revised Steelman Requirements for High Order Computer Programming Languages,* 1979.

[DoD80]
Department of Defense, *Requirements for Ada Programming Support Environments: Stoneman,* February 1980.

[DOYL80]
Doyle, J., "A Model for Deliberation, Action and Introspection," Technical Report MIT/AI/TR-581, MIT Laboratory for Artificial Intelligence, 1980.

[DR79]
Davis, A. M., and T. G. Rauscher, "Formal Techniques and Automatic Processing to Ensure Correctness in Requirements Specifications," *Proc. IEEE Conference on Specifications of Reliable Software,* IEEE Catalog No. 79 CH1401-9C, 1979, pp. 15-35.

[DT80]
Deutsch, L. P., and E. A. Taft, "Requirements for and Experimental Programming Environment," Xerox PARC Report CSL-80-10, 1980.

[DV77]
Davis, C. G., and C. R. Vick, "The Software Development System," *IEEE Transactions on Software Engineering,* Vol. SE-3, No. 1, January 1977.

[EARL71]
Earley, J., "Toward an Understanding of Data Structures," *Communications of the ACM,* Vol. 14, No. 10, October 1971, pp. 617-627.

[EGL76]
Eswaren, K. P., J. N. Gray, R. A. Lorie, and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," *Communications of the ACM*, Vol. 19, No. 11, November 1976.

[ESWA76]
Eswaren, K. P., "Specifications, Implementations and Interactions of a Trigger Subsystem in an Integrated Database System," IBM Research Report RJ1820, San Jose, Calif., August 1976.

[FAHL79]
Fahlman, S. E., *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, 1979.

[FALK80]
Falkenberg, E. D., "Conceptualization of Data," *Infotech State-of-the-Art Report on Data Design,* Pergamon Infotech Limited, London, 1980.

[FAUS81]
Faust, G., "Semiautomatic Translation of COBOL into HIBOL," M.S. thesis (Technical Report MIT/LCS/TR-256), MIT Laboratory for Computer Science, March 1981.

[FEIG77]
Feigenbaum, E. A., "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering," *Proc. 5th International Joint Conference on Artificial Intelligence,* Cambridge, Mass., August 1977.

[FIND79]
Findler, N. V., *Associative Networks: Representation and Use of Knowledge by Computer*, Academic Press, New York, 1979.

[FLOY67]
Floyd, R. W., "Assigning Meanings to Programs," in J. T. Schwartz (ed.), *Proc. Symposium in Applied Mathematics,* Vol. 19, American Mathematical Society, 1967, pp. 19-32.

[FN79]
Feiertag, R., and P.G. Neumann, "The Foundations of a Provably Secure Operating System (PSOS)," *Proc. National Computer Conference,* 1979, pp. 329-334.

[FOGG82]
Fogg, D., "Parser Support for Abstract Data Types in INGRES," Masters Report, Univ. of California, Berkeley, September 1982.

[FW76a]
Friedman, D.P., and D.S. Wise, "CONS Should Not Evaluate its Arguments," in *Automata, Languages, and Programming*, Edinburgh Univ. Press, Edinburgh, Scotland, 1976.

[FW76b]
Friedman, D.P., and D.S. Wise, "The Impact of Applicative Programming on Multiprocessing," *Proc. ACM International Conference on Parallel Processing,* 1976, pp. 263-272.

[GANE80]
Ganes, C.P., "Data Design in Structured System Analysis," in P. Freeman and A.I. Wasserman (eds.), *Tutorial on Software Design Techniques,* 1980.

[GBM82]
Greenspan, S., A.T. Borgida, and J. Mylopoulos, "Capturing More World Knowledge in the Requirements Specification," *Proc. 6th International Conference on Software Engineering,* Tokyo, Japan, 1982.

[GERH75]
Gerhart, S.L., "Knowledge About Programs: a Model and Case Study," *Proc. of International Conference on Reliable Software,* June 1975, pp. 88-95.

[GH78]
Guttag, J.V., and J.J. Horning, "The Algebraic Specification of Abstract Data Types," *Acta Informatica*, Vol. 10, 1978, pp. 27-52.

[GH80]
Guttag, J.V., and J.J. Horning, "Formal Specification as a Design Tool," *Proc. ACM Symposium on Principles of Programming Languages,* SIGACT and SIGPLAN, January 1980, pp. 251-261.

[GHM78]
Guttag, J.V., E. Horowitz, and D.R. Musser, "Abstract Data Types and Software Validation," *Communications of the ACM,* Vol. 21, No. 12, December 1978, pp. 1048-1064.

[GKB82]
Gustafsson, M.R., T. Karlsson, and Bubenko, J.A., Jr., "A Declarative Approach to Conceptual Information Modelling," in [OSV82], pp. 93-142.

[GM78]
Gallaire, H., and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, New York, 1978.

[GM80]
Goodenough, J.B., and C.L. McGowan, "Software Quality Assurance: Testing and Validation," *Proc. IEEE,* Vol. 68, No. 9, September 1980.

[GMS77]
Geschke, C.M., J.H. Morris, Jr., and E.H. Satterthwaite, "Early Experience with Mesa," *Communications of the ACM,* Vol. 20, No. 8, August 1977, pp. 540-553.

[GMW79]
Gordon, M.J., A.J. Milner, and C.P. Wadsworth, "Edinburgh LCF," *Lecture Notes in Computer Science,* No. 78, Springer-Verlag, New York, 1979.

[GOLD73]
Goldberg, J., "Proceedings of a Symposium on the High Cost of Software," Technical Report, Stanford Research Institute, Stanford, Calif., September 1973.

[GOOD77]
Good, D.I., "Constructing Verified and Reliable Communications Processing Systems," *Software Engineering Notes,* Vol. 2, No. 5, October 1977, pp. 8-13.

[GP77]
Goldstein, I., and S. Papert, "Artificial Intelligence, Language, and the Study of Knowledge," *Cognitive Science,* Vol. 1, No. 1, 1977.

[GR77]
Goldstein, I., and R.B. Roberts, "NUDGE: A Knowledge-Based Scheduling Program," *Proc. 5th International Joint Conference on Artificial Intelligence,* Cambridge, Mass., August 1977.

[GRAY78]
Gray, J.N., "Notes on Data Base Operating Systems," *Proc. Advanced Course on Operating Systems,* Munich, West Germany, in *Lecture Notes in Computer Science,* No. 60, Springer-Verlag, New York, 1978.

[GREE69a]
Green, C., "The Application of Theorem Proving to Question-Answering Systems," Ph.D. thesis, Dept. of Electrical Engineering, Stanford Univ., 1969.

[GREE69b]
Green, C., "Theorem Proving by Resolution as a Basis for Question-Answering Systems," in D. Michie and B. Meltzer (eds.), *Machine Intelligence 4*, Edinburgh Univ. Press, Edinburgh, Scotland, 1969.

[GRIF76]
Griffiths, P., and B. Wade, "An Authorization Mechanism for a Relational Data Base System," *ACM Transactions on Database Systems,* Vol. 2, No. 3, September 1976.

[GTW78]
Goguen, J. A., J. W. Thatcher, and E. G. Wagner, "An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types," in R. Yeh (ed.), *Current Trends in Programming Methodology,* Vol. IV, Prentice-Hall, Englewood Cliffs, N.J., 1978.

[GUAR78]
Guarino, L. R., "The Evolution of Abstraction in Programming Languages," Technical Report CMU-CS-78-120, Carnegie-Mellon Univ., May 1978.

[GUTT77]
Guttag, J. V., "Abstract Data Types and the Development of Data Structures," *Communications of the ACM,* Vol. 20, No. 6, June 1977, pp. 396-404.

[GUTT80]
Guttag, J. V., "Notes on Type Abstraction (Version 2)," *IEEE Transactions on Software Engineering,* Vol. SE-6, No. 1, January 1980, pp. 13-23.

[GW79]
Gerhart, S. L., and D. S. Wile, "Preliminary Report on the Delta Experiment: Specification and Verification of a Multiple-User File Updating Module," *Proc. IEEE Conference on Specifications of Reliable Software,* IEEE Catalog No. 79 CH1401-9C, 1979, pp. 198-211.

[GY76]
Gerhart, S. L., and L. Yelowitz, "Observations of Fallibility in Applications of Modern Programming Methodologies," *IEEE Transactions on Software Engineering,* Vol. SE-2, No. 5, September 1976, pp. 195-207.

[HABE73]
Habermann, A. N., "Critical Comments on the Programming Language Pascal," *Acta Informatica,* Vol. 3, 1973, pp. 47-57.

[HAL79]
Hewitt C., G. Attardi, and H. Lieberman, "Specifying and Proving Properties of Guardians for Distributed Systems," *Proc. Conference on Semantics of Concurrent Computation,* INRIA, Evian, France, July 1979.

[HARD80]
Hardgrave, W. T., "Positional Set Notation," internal report, National Bureau of Standards, February 1980.

[HAS80]
Hewitt, C., G. Attardi, and M. Simi, "Knowledge Embedding with a Description System," *Proc. 1st AAAI National Conference,* August 1980.

[HAYE74]
Hayes, P. J., "Some Problems and Non-Problems in Representation Theory," *Proc. AISB Summer Conference,* Essex Univ., Essex, Great Britian, 1974.

[HAYE77]
Hayes, P. J., "In Defense of Logic," *Proc. 5th International Joint Conference on Artificial Intelligence,* Cambridge, Mass., 1977, pp. 559-565.

[HAYE78]
Hayes, P. J., "The Ontology of Liquids," unpublished manuscript, 1978.

[HAYE79]
Hayes, P. J., "The Logic of Frames," in D. Metzing (ed.), *Frame Conceptions and Text Understanding,* Walter de Gruyter and Co., Berlin, 1979, pp. 46-61.

[HAZE76]
Hazen, A., "Expressive Completeness in Modal Language," *Journal of Philosophical Logic,* Vol. 5, 1976.

[HB77]
Hewitt, C., and H. Baker, "Laws for Communicating Parallel Processes," *Proc. 1977 IFIP Congress,* 1977.

[HBS73]
Hewitt, C., P. Bishop, and R. Steiger, "A Universal Modular ACTOR Formalism for Artificial Intelligence," *Proc. International Joint Conference on Artificial Intelligence,* Palo Alto, Calif., August 1973.

[HEND75]
Hendrix, G., "Expanding the Utility of Semantic Networks through Partitioning," *Proc. International Joint Conference on Artificial Intelligence,* Tbilisi, USSR, September 1975.

[HENI79]
Heninger, K. L., "Specifying Software Requirements for Complex Systems: New Techniques and Their Applications," *Proc. IEEE Conference on Specifications of Reliable Software,* IEEE Catalog No. 79 CH1401-9C, 1979, pp. 1-14.

[HENK50]
Henkin, L., "Completeness in the Theory of Types," *Journal of Symbolic Logic,* Vol. 15, 1950, pp. 81-91.

[HEWI69]
Hewitt, C. E., "PLANNER: A Language for Proving Theorems in Robots," *Proc. International Joint Conference on Artificial Intelligence,* Washington, D.C., May 1969.

[HEWI71]
Hewitt, C. E., "PLANNER: A Language for Proving Theorems in Robots," *Proc. International Joint Conference on Artificial Intelligence,* London, Great Britian, August 1971.

[HEWI72]
Hewitt, C. E., *Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot,* Ph.D. thesis, Dept. of Mathematics, MIT, 1972.

[HEWI75]
Hewitt, C. E., "How To Use What You Know," *Proc. International Joint Conference on Artificial Intelligence,* Tbilisi, USSR, August 1975.

[HEWI77]
Hewitt, C. E., "Viewing Control Structures as Patterns of Passing Messages," *Artificial Intelligence,* Vol. 8, 1977, pp. 323-364.

[HEWI80]
Hewitt, C. E., "The Apiary Network Architecture for Knowledgeable Systems," *Conference Record of the 1980 Lisp Conference,* Stanford Univ., Stanford, Calif., August 1980.

[HG74]
Hewitt, C. E., and I. Greiff, "Actor Semantics of PLANNER-73," Working Paper No. 81, MIT Laboratory for Artificial Intelligence, 1974.

[HINT62]
Hintikka, J., *Knowledge and Belief: An Introduction to the Logic of the Two Notions,* Cornell Univ. Press, 1962.

[HK81]
Hecht, M. S., and L. Kershberg, "Update Semantics for the Functional Data Model," Data Base Research Report No. 4, Bell Labs, Holmdell, N.J., 1981.

[HL82]
Haskin, R. L., and R. A. Lorie, "On Extending the Functions of a Relational Database System," *Proc. 1982 ACM SIGMOD International Conference on the Management of Data,* Orlando, Fla., 1982, pp. 207-212.

[HM75]
Hammer, M., and D. McLeod, "Semantic Integrity in a Relational Database System," *Proc. 1st International Conference on Very Large Databases,* Framingham, Mass., September 1975, pp. 25-47.

[HM76]
Hammer, M., and D. McLeod, "A Framework for Database Semantic Integrity," *Proc. 2nd International Conference on Software Engineering,* San Francisco, Calif., October 1976.

[HM78]
Hammer, M., and D. McLeod, "The Semantic Data Model: a Modelling Mechanism for Database Applications," *Proc. 1978 ACM SIGMOD International Conference on the Management of Data,* Austin, Texas, May-June 1978.

[HM81]
Hammer, M., and D. McLeod, "Database Description with SDM: A Semantic Database Model," *ACM Transactions on Database Systems,* Vol. 6, No. 3, September 1981.

[HOAR69]
Hoare, C. A. R., "An Axiomatic Basis for Computer Programming," *Communications of the ACM,* Vol. 12, October 1969, pp. 576-580, 583.

[HOAR72a]
Hoare, C. A. R., "Proof of Correctness of Data Representations," *Acta Informatica,* Vol. 1, No. 4, 1972, pp. 271-281.

[HOAR72b]
Hoare, C. A. R., "Notes on Data Structuring," in O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare (eds.), *Structured Programming,* Academic Press, New York, 1972, pp. 83-174.

[HOAR78]
Hoare, C. A. R., "Communicating Sequential Processes," *Communications of the ACM,* Vol. 21, No. 8, August 1978.

[HOT76]
Hall, P., J. Owlett, and S. J. P. Todd, "Relations and Entities," in J. M. Nijssen (ed.), *Modelling in Database Management Systems,* Elsevier North-Holland, New York, 1976.

[HOUS77]
Housel, B. C., "A Unified Approach to Data Conversion," *Proc. 3rd International Conference on Very Large Databases,* Tokyo, Japan, 1977.

[HOWD79]
Howden, W. E., "An Analysis of Software Validation Techniques for Scientific Programs," Technical Report DM-171-IR, Dept. of Mathematics, Univ. of Victoria, Victoria, B.C., Canada, March 1979.

[HR79]
Hunt, H. B., and D. J. Rosenkrantz, "The Complexity of Testing Predicate Locks," *Proc. 1979 ACM SIGMOD International Conference on the Management of Data,* Boston, Mass., May 1979.

[HSW75]
Held, G., M. Stonebraker, and E. Wong, "INGRES: A Relational Database System," *Proc. AFIPS 1975 National Computer Conference,* Vol. 44, 1975.

[HW73]
Hoare, C. A. R., and N. Wirth, "An Axiomatic Definition of the Programming Language PASCAL," *Acta Informatica*, Vol. 2, No. 4, 1973, pp. 335-355.

[HWY79]
Housel, B. C., V. Waddle, and S. B. Yao, "The Functional Dependency Model for Logical Database Design," *Proc. 5th International Conference on Very Large Databases,* Rio de Janeiro, Brazil, October 1979.

[HY79]
Hevner, A. R., and S. B. Yao, "Query Processing in Distributed Database Systems," *IEEE Transactions on Software Engineering,* Vol. SE-5, No. 3, 1979.

[IBHK79]
Ichbiah, J. D., J. P. G. Barnes, J.-C. Heliard, B. Krieg-Brueckner, O. Roubine, and B. A. Wichmann, "Rationale for the Design of the Ada Programming Language," *SIGPLAN Notices,* Vol. 14, No. 6, Part B, 1979.

[IEEE79a]
IEEE Computer Society (eds.), *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: an Assessment of the State of the Art,* IEEE Catalog No. TH0067-9, 1979.

[IEEE79b]
IEEE Computer Society (eds.), *Proc. Conference on Specifications of Reliable Software,* IEEE Catalog No. 79 CH1401-9C, 1979.

[IKWL79]
Ichbiah, J. D., B. Krieg-Brueckner, B. A. Wichmann, H. F. Ledgard, J.-C. Heliard, J.-R. Abrial, J. P. G. Barnes, and O. Roubine, "Preliminary Ada Reference Manual," *SIGPLAN Notices,* Vol. 14, No. 6, Part A, 1979.

[IKWL80]
Ichbiah, J. D., B. Krieg-Brueckner, B. A. Wichmann, H. F. Ledgard, J.-C. Heliard, J.-R. Abrial, J. P. G. Barnes, M. Woodger, O. Roubine, P. N. Hilfinger, and R. Firth, *Reference Manual for the Ada Programming Language: Proposed Standard Document,* Department of Defense, US Government Printing Office 008-000-00354-8, July 1980; also in: *Lecture Notes in Computer Science,* No. 106, Springer-Verlag, New York, 1981.

[IKWL82]
Ichbiah, J. D., B. Krieg-Brueckner, B. A. Wichmann, H. F. Ledgard, J.-C. Heliard, J.-L. Gailly, J.-R. Abrial, J. P. G. Barnes, M. Woodger, O. Roubine, P. N. Hilfinger, and R. Firth, *Reference Manual for the Ada Programming Language,* Draft Revised MIL-STD 1815; Draft Proposed ANSI Standard Document for Editorial Review, US Department of Defense, Honeywell Inc., and Alsys, July 1982; also available from AdaTEC, ACM order no. 825820.

[IMB81]
Islam, N., T. J. Myers, and P. Broome, "A Simple Optimizer for FP-like Languages," *Proc. ACM Conference on Functional Programming and Architecture,* New Hampshire, 1981.

[INGA78]
Ingalls, D. H., "The Smalltalk-76 Programming System: Design and Implementation," *Conference Record of the Fifth Annual ACM Symposium on Programming Languages,* Tucson, Arizona, January 1978.

[IOS79]
International Organization for Standardization, *Draft Specification for the Computer Programming Language Pascal,* ISO/TC 97/SC 5 N, 1979.

[ISRA80]
Israel, D. J., "What's Wrong with Non-Monotonic Logic?," *Proc. 1st National Conference on Artifical Intelligence,* American Association for Artificial Intelligence, Stanford, Calif., 1980, pp. 99-101.

[ISRA82]
Israel, D. J., "On Interpreting Semantic Network Formalisms," *International Journal of Computer Mathematics,* (to appear in a special issue on Computational Linguistics edited by N. Cercone); also available as BBN Report No. 5117, Bolt, Beranek and Newman Inc., Cambridge, Mass., 1982.

[IVER79]
Iverson, K. E., "Operators," *ACM Transactions on Programming Languages and Systems,* Vol. 1, No. 2, 1979.

[JACO82]
Jacobs, B. E., "On Database Logic," *Journal of the ACM,* Vol. 29, No. 2, April 1982, pp. 310-332.

[JL76]
Jones, A. K., and B. H. Liskov, "An Access Control Facility for Programming Languages," MIT Computation Structures Group and Carnegie-Mellon Univ., MIT Memo 137, 1976.

[JS82]
Jarke, M., and J. W. Schmidt, "Query Processing Strategies in the Pascal/R Relational Database Management System," *Proc. 1982 ACM SIGMOD International Conference on the Management of Data,* Orlando, Fla., June 1982.

[JW74]
Jensen, K., and N. Wirth, *Pascal User Manual and Report,* Springer-Verlag, New York, 1974.

[KAHN79]
Kahn, K. M., "Creation of Computer Animation from Story Descriptions," Ph.D. thesis, MIT, 1979.

[KENT78]
Kent, W., *Data and Reality,* Elsevier North-Holland, New York, 1978.

[KH81]
Kornfeld, W. A., and Hewitt, C., "The Scientific Community Metaphor," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-11, No. 1, January 1981.

[KING82]
King, R., "A Semantics-Based Methodology for Database Design and Evolution," Ph.D. thesis (Technical Report), Computer Science Dept., Univ. of Southern California, October 1982.

[KL80a]
Keller, R. M., and G. Lindstrom, "Parallelism in Functional Programming through Applicative Loops," Technical Report, Univ. of Utah, 1980.

[KL80b]
Krieg-Brueckner, B., and D.C. Luckham, "Anna: Towards a Language for Annotating Ada Programs," *SIGPLAN Notices,* Vol. 15, No. 11, 1980, pp. 128-138.

[KLEE71]
Kleene, S.C., *Introduction to Metamathematics,* Elsevier North-Holland, New York, 1971.

[KLVO82]
Krieg-Brueckner, B., D.C. Luckham, F.W. von Henke, and O. Owe, *Anna: a Language for Annotating Ada Programs,* Springer-Verlag, New York (to appear).

[KM82a]
King, R., and D. McLeod, "Semantic Database Models," in S.B. Yao (ed.), *Principles of Database Design,* Prentice-Hall, Englewood Cliffs, N.J. (to appear).

[KM82b]
King, R., and D. McLeod, "The Event Database Specification Model," *Proc. Second International Conference on Databases: Improving Usability and Responsiveness,* Jerusalem, Israel, June 1982.

[KM82c]
King, R., and D. McLeod, "A Methodology and Tool for Database Life-Cycle Management," Technical Report, Computer Science Dept., Univ. of Southern California, November 1982 (submitted for publication).

[KNUT73]
Knuth, D.E., *The Art of Computer Programming, Vol. 1: Fundamental Algorithms,* Second edition, Addison-Wesley, Reading, Mass., 1973.

[KOLA82]
Kolata, G., "How Can Computers Get Common Sense?," *Science,* Vol. 217, No. 4566, September 24, 1982.

[KONO82]
Konolige, K., "Circumscriptive Ignorance," *Proc. AAAI National Conference,* Pittsburgh, Penn., August 1982.

[KORN82]
Kornfeld, W., "Concepts in Parallel Problem Solving," Ph.D. thesis, MIT, 1982.

[KOWA74]
Kowalski, R., "Predicate Logic as a Programming Language," *Proc. IFIP Congress,* 1974, pp. 569-574.

[KOWA78]
Kowalski, R., "Logic for Data Description," in [GM78].

[KOWA79]
Kowalski, R., *Logic for Problem Solving,* Elsevier North-Holland, New York, 1979.

[KP76]
Kernighan, B.W., and P.J. Plauger, *Software Tools,* Addison-Wesley, Reading, Mass., 1976.

[KR81]
Kung, H.T., and J.T. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Transactions on Database Systems,* Vol. 6, No. 2, June 1981.

[KUHN67]
Kuhns, J.L., "Answering Questions by Computer—a Logical Study," Memorandum RM 2428 PR, Rand Corporation, Santa Monica, Calif., December 1967.

[LAND65]
Landin, P.J., "A Correspondence Between ALGOL 60 and Church's Lambda Notation," *Communications of the ACM,* Vol. 8, Nos. 2 and 3, 1965.

[LAND66]
Landin, P.J., "The Next 700 Programming Languages," *Communications of the ACM,* Vol. 9, No. 3, 1966.

[LENA77]
Lenat, D.B., "The Ubiquity of Discovery," *Proc. International Joint Conference on Artificial Intelligence,* Cambridge, Mass., August 1977.

[LEVE81a]
Levesque, H., "A Formal Treatment of Incomplete Knowledge Bases," Ph.D. thesis, Dept. of Computer Science, Univ. of Toronto, 1981; also available as Technical Report No. 3, Fairchild Laboratory for Artificial Intelligence Research, Palo Alto, Calif.

[LEVE81b]
Levesque, H., "The Interaction with Incomplete Knowledge Bases: a Formal Treatment," *Proc. International Joint Conference on Artificial Intelligence*, Univ. of British Columbia, Vancouver, B.C., Canada, 1981.

[LEVI77]
Levin, R., "Program Structures for Exceptional Condition Handling," Ph.D. thesis, Carnegie-Mellon Univ., June 1977.

[LEWI68]
Lewis, D., "Counterpart Theory and Quantified Modal Logic," *Journal of Philosophy,* Vol. 65, 1968, pp. 113-126.

[LGHL78]
London, R.L., J.V. Guttag, J.J. Horning, B.W. Lampson, J.G. Mitchell, and G.J. Popek, "Proof Rules for the Programming Language Euclid," *Acta Informatica,* Vol. 10, No. 1, 1978 pp. 1-26.

[LHLM77]
Lampson, B.W., J.J. Horning, R.L. London, J.G. Mitchell, and G.J. Popek, "Report on the Programming Language Euclid," *SIGPLAN Notices,* Vol. 12, No. 2, February 1977, pp. 1-79.

[LIEB81a]
Lieberman, H., "A Preview of Act-1," AI Memo No. 625, MIT Artificial Intelligence Laboratory, 1981.

[LIEB81b]
Lieberman, H., "Thinking About Lots of Things At Once Without Getting Confused: Parallelism in Act-1," AI Memo No. 626, MIT Artificial Intelligence Laboratory, 1981.

[LIPS78]
Lipski, W., Jr., "On Semantic Issues Connected with Incomplete Information Data Bases," PAS Report 325, Institute of Computer Science, Warsaw, Poland, 1978.

[LIPS79]
Lipski, W., Jr., "On Semantic Issues Connected with Incomplete Information Databases," *ACM Transactions on Database Systems,* Vol. 4, No. 3, September 1979, pp. 262-296.

[LISK77]
Liskov, B., *et al.,* "Abstraction Mechanisms in CLU," *Communications of the ACM,* Vol. 20, No. 8, August 1977, pp. 564-576.

[LM79]
Levesque, H., and J. Mylopoulos, "A Procedural Semantics for Semantic Networks," in [FIND79].

[LMW79]
Linger, R. C., H. D. Mills, and B. I. Witt, *Structured Programming Theory and Practice*, Addison-Wesley, Reading, Mass., 1979.

[LOCK79]
Lockmann, P. *et al.,* "Data Abstractions for Data Base Systems," *ACM Transactions on Database Systems,* Vol. 4, No. 1, March 1979.

[LOND75]
London, R. L., "A View of Program Verification," *Proc. International Conference on Reliable Software,* IEEE Computer Society, April 1975, pp. 534-545.

[LORI77]
Lorie, R. A., "Physical Integrity in a Large Segmented Database," *ACM Transactions on Database Systems,* Vol. 2, No. 1, March 1977.

[LS80]
Lamersdorf, W., and J. W. Schmidt, "Specification of Pascal/R," Report No. 73/74, Fachbereich Informatik, Univ. of Hamburg, July 1980.

[LSAS77]
Liskov B., A. Snyder, R. Atkinson, and C. Schaffert, "Abstraction Mechanism in CLU," *Communications of the ACM,* Vol. 20, No. 8, August 1977, pp. 564-576.

[LW76]
Lorie, R., and B. Wade, "The Compilation of a Very High Level Data Language," IBM Research Report RJ2008, San Jose, Calif., May 1977.

[LZ74]
Liskov, B., and S. Zilles, "Programming With Abstract Data Types," *SIGPLAN Notices,* Vol. 9, No. 4, April 1974, pp. 50-59.

[LZ75]
Liskov, B., and S. Zilles, "Specification Techniques for Data Abstractions," *IEEE Transactions on Software Engineering,* Vol. SE-1, No. 1, March 1975, pp. 7-19.

[LZ77]
Liskov, B., and S. Zilles, "An Introduction to Formal Specifications of Data Abstractions," in R. Yeh (ed.), *Current Trends in Programming Methodology, Vol. I,* Prentice-Hall, Englewood Cliffs, N.J., 1977.

[MANN74]
Manna, Z., *Mathematical Theory of Computation,* McGraw-Hill, New York, 1974.

[MBW78]
Mylopoulos, J., P. A. Bernstein, and H. K. T. Wong "A Preliminary Specification for TAXIS," Technical Report CCA-78-02, Computer Corporation of America, Cambridge, Mass., January 1978.

[MBW80]
Mylopoulos, J., P. A. Bernstein, and H. K. T. Wong, "A Language Facility for Designing Interactive Database-Intensive Applications," *ACM Transactions on Database Systems,* Vol. 5, No. 2, June 1980, pp. 185-207.

[MCAL80]
McAllester, D. A., "An Outlook on Truth Maintenance," Memo MIT/AIM-551, MIT Laboratory for Artificial Intelligence, August 1980.

[MCCA62]
McCarthy, J., *et al., LISP 1.5 Programmer's Manual,* MIT Press, Cambridge, Mass., 1962.

[MCCA80]
McCarthy, J., "Circumscription — A Form of Non-Monotonic Reasoning," *Artificial Intelligence,* Vol. 13, Nos. 1 and 2, April 1980, pp. 27-39.

[MCDE80]
McDermott, D., "Non-Monotonic Logic II: Non-Monotonic Modal Theories," Research Reort No. 174, Dept. of Computer Science, Yale Univ., February 1980.

[MCGE76]
McGee, W. C., "On User Criteria for Data Model Evaluation," *ACM Transactions on Database Systems,* Vol. 1, No. 4, December 1976.

[MD78]
McDermott, D., and J. Doyle, "Non-Monotonic Logic I," Memo MIT/AIM-486, MIT Laboratory for Artificial Intelligence, 1978.

[MEND64]
Mendelson, E., *Introduction to Mathematical Logic,* Van Nostrand, Princeton, N.J., 1964.

[MG77]
Miller, M. L., and I. Goldstein, "Problem Solving Grammars as Formal Tools for Intelligent CAI," *Proc. ACM,* 1977.

[MH69]
McCarthy, J., and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in D. Michie and B. Meltzer (eds.), *Machine Intelligence 4,* Edinburgh Univ. Press, Edinburgh, Scotland, 1969.

[MILL76]
Millen, J. K., "Security Kernel Validation in Practice," *Communications of the ACM,* Vol. 19, No. 5, May 1976, pp. 243-250.

[MILL79]
Miller, E. (ed.), *Tutorial: Automated Tools for Software Engineering,* IEEE Computer Society, IEEE Catalog No. EHO 150-3, 1979.

[MILN78]
Milner, R., "A Theory of Type Polymorphism in Programming," *Journal of Computer and System Sciences,* Vol. 17, 1978, pp. 348-375.

[MINS75]
Minsky, M., "A Framework for Representing Knowledge," in P. Winston (ed.), *The Psychology of Computer Vision,* McGraw-Hill, New York, 1975, pp. 211-277.

[MISR78]
Misra, J., "Some Aspects of the Verification of Loop Computations," *IEEE Transactions on Software Engineering,* Vol. SE-4, No. 6, November 1978, pp. 478-485.

[MN74]
Moore, J. and A. Newell, "How can MERLIN Understand?," in L. Gregg (ed.), *Knowledge and Cognition,* Lawrence Erlbaum Associates, Hillsdale, N.J., 1974.

[MONT74]
Montague, R., "The Proper Treatment of Quantification in Ordinary English," in R. Thomason (ed.), *Formal Philosophy,* Yale Univ. Press, New Haven, 1974, pp. 247-270.

[MOOR77]
Moore, R., "Reasoning About Knowledge and Action," *Proc. 5th International Joint Conference on Artificial Intelligence,* Cambridge, Mass., August 1977.

[MOOR81]
Moore, R., "Reasoning about Knowledge and Action," Technical Note 191, Artificial Intelligence Center, SRI International, Menlo Park, 1980.

[MOOR82]
Moore, R. C., "The Role of Logic in Knowledge Representation and Commonsense Reasoning," *Proc. AAAI National Conference,* Pittsburgh, Penn., August 1982.

[MORR73a]
Morris, J. H., "Types Are Not Sets," *Proc. ACM Symposium on Principles of Programming Languages,* 1973, pp. 120-124.

[MORR73b]
Morris, J. H., "Protection in Programming Languages," *Communications of the ACM,* Vol. 16, January 1973, pp. 15-21.

[MP82]
Manola, F., and A. Pirotte, "CQLF—A Query Language for CODASYL-Type Databases," *Proc. 1982 ACM SIGMOD International Conference on the Management of Data,* Orlando, Fla., 1982.

[MPBR82]
Manola, F., A. Pirotte, B. Blaustein, and D. R. Ries, "Family of Data Model Specifications for DBMS (Database Management System) Standards," Computer Corporation of America, Cambridge, Mass., December 1982; NBS-GCR-82-419, available as NTIS Report PB83-163394.

[MS73]
McDermott, D., and G. J. Sussman, "The Conniver Reference Manual," Memo MIT/AIM-259A, MIT Laboratory for Artificial Intelligence, 1973.

[MS81]
McLeod, D., and J. M. Smith, "Abstraction in Databases," in [BZ81].

[MSHI78]
McCarthy, J., M. Sato, T. Hayashi, and S. Igarashi, "On the Model Theory of Knowledge," Memo AIM-312, Dept. of Computer Science, Stanford Univ., 1978.

[MW80]
Mylopoulos, J., and H. Wong, "Some Features of the TAXIS Data Model," *Proc. 6th International Conference on Very Large Databases,* Montreal, Que., Canada, October 1980.

[MYER81]
Myers, T. J., Ph.D. Dissertation, Univ. of Pennsylvania, 1981.

[MYLO81]
Mylopoulos, J., "An Overview of Knowledge Representation," in [BZ81].

[NEWE62]
Newell, A., "Some Problems of Basic Organization in Problem-Solving Programs," Memorandum RM-3283-PR, Rand Corporation, Santa Monica, Calif., December 1962.

[NEWE80]
Newell, A., "Physical Symbol Systems," *Cognitive Science,* Vol. 4, No. 2, April-June 1980, pp. 135-183.

[NEWE81]
Newell, A., "The Knowledge Level," *Proc. AAAI National Conference,* (presidential address) Stanford, Calif.; reprinted in *AI Magazine,* Vol. 2, No. 2, 1981.

[NG78]
Nicolas, J. M., and H. Gallaire, "Data Base: Theory vs. Interpretation," in [GM78].

[NILS71]
Nilsson, N., *Problem Solving Methods in Artificial Intelligence,* McGraw-Hill, Englewood Cliffs, N.J., 1971.

[NR69]
Naur, P., and B. Randell (eds.), *Software Engineering,* NATO, 1969 (report on a conference sponsored by the NATO Science Committee, Garmisch, West Germany, October 7-11, 1968).

[NS78]
Navathe, S. B., and M. Schkolnick, "View Representation in Logical Database Design," *Proc. 1978 ACM SIGMOD International Conference on the Management of Data,* Austin, Texas, May-June 1978.

[NY78]
Nicolas, J. M., and K. Yazdanian, "Integrity Checking in Deductive Databases," in [GM78].

[ONG82]
Ong, J., "Specification of an ADT Facility for a Relational Data Base System," Masters Report, Univ. of California, Berkeley, September 1982.

[ORGA76]
Organick, E. I. (chrm.), *Proc. of Conference on Data: Abstraction, Definition, and Structure, SIGPLAN Notices,* Vol. 11, No. 2, 1976.

[OSV82]
Olle, T. W., H. G. Sol, and A. A. Verjn-Stuart, *Information Systems Design Methodologies: A Comparative Review* (Proc. IFIP TC 8 Working Conference on Comparative Review of Information Systems Design Methodologies, Noordwijkerhout, Netherlands, May 1982), Elsevier North-Holland, Amsterdam, The Netherlands, 1982.

[OVER81]
Overmeyer, R., "A Time Expert for INGRES," M.S. thesis, Univ. of California, Berkeley, August 1981.

[PARN71]
Parnas, D. L., "Information Distribution Aspects of Design Methodology," *Proc. of IFIP Congress,* Booklet TA-3, 1971, pp. 26-30.

[PARN72a]
Parnas, D. L., "A Technique for Software Module Specification with Examples," *Communications of the ACM,* Vol. 15, May 1972, pp. 330-336.

[PARN72b]
Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM,* Vol. 15, No. 12, December 1972.

[PETE80]
Peters, L., "Software Design Engineering," *Proc. IEEE,* Vol. 68, No. 9, September 1980.

[POPL73]
Pople, H. E., "On the Mechanization of Abductive Logic," *Proc. International Joint Conference on Artificial Intelligence,* Palo Alto, Calif., August 1973.

[POWE82]
Powell, M., private communication.

[QUIL68]
Quillian, M. R. "Semantic Memory," in M. Minsky (ed.), *Semantic Information Processing,* MIT Press, 1968.

[RAMS79]
Ramshaw, L. H., "Formalizing the Analysis of Algorithms," Ph.D. thesis, Stanford Univ., 1979.

[RAPH71]
Raphael, B., "The Frame Problem in Problem-Solving systems," in N. V. Findler and B. Meltzer (eds.), *Artificial Intelligence and Heuristic Programming,* Edinburgh Univ. Press, Edinburgh, Scotland, 1971.

[RB82a]
Ridjanovic, D., and M. L. Brodie, "Semantic Data Model-Driven Design, Specification and Verification of Interactive Database Transactions," Computer Corporation of America, Cambridge, Mass., April 1982.

[RB82b]
Ridjanovic, D., and M. L. Brodie, "Defining Database Dynamics with Attribute Grammars," *Information Processing Letters,* Vol. 14, No. 3, May 1982.

[RB82c]
Ridjanovic, D., and M. L. Brodie, "Definition of Fundamental Concepts and Tools for Semantic Modelling of Data and Associated Operations," submitted for publication.

[RB82d]
Ridjanovic, D., and M. L. Brodie, "Disciplined Methodology for Database Transaction Design," submitted for publication.

[RB82e]
Ridjanovic, D., and M.L. Brodie, "Functional Specification and Implementation Verification of Database Transactions," submitted for publication.

[RB82f]
Ridjanovic, D., and M.L. Brodie, "Conceptual Modelling of Office Procedures," submitted for publication.

[RB83]
Ridjanovic, D., and M.L. Brodie, "Action and Transaction Skeletons: High Level Language Constructs for Database Transactions," *Proc. 1983 SIGPLAN Conference,* San Francisco, Calif., June 1983.

[REIT77]
Reiter, R., *An Approach to Deductive Question-Answering,* BBN Technical Report 3649, Bolt, Beranek and Newman, Inc., Cambridge, Mass., September 1977.

[REIT78a]
Reiter, R., "Deductive Question-Answering on Relational Databases," in [GM78], pp. 149-177.

[REIT78b]
Reiter, R., "On Closed World Data Bases," in [GM78], pp. 55-76.

[REIT78c]
Reiter, R., "On Reasoning by Default," *Proc. Second TINLAP Conference,* Urbana, Ill., July 1978, pp. 210-218.

[REIT80a]
Reiter, R., "Equality and Domain Closure in First Order Databases," *Journal of the ACM,* Vol. 27, No. 2, 1980, pp. 235-249.

[REIT80b]
Reiter, R., "Databases: A Logical Perspective," in [BZ80], pp. 174-176.

[REIT80c]
Reiter, R., "A Logic for Default Reasoning," *Artificial Intelligence,* Vol. 13, 1980, pp. 81-132.

[REIT81]
Reiter, R., "On Interacting Defaults," *Proc. International Joint Conference on Artificial Intelligence,* Vancouver, B.C., Canada, 1981.

[REIT82]
Reiter, R., "Circumscription Implies Predicate Completion (Sometimes)," *Proc. AAAI National Conference,* Pittsburgh, Penn., August 1982.

[REYN79]
Reynolds, J.C., "Reasoning About Arrays," *Communications of the ACM,* Vol. 22, No. 5, May 1979, pp. 290-298.

[RICH80]
Rich, C., "Inspection Methods in Programming," Ph.D. thesis (Technical Report MIT/AI/TR-604), MIT Laboratory for Artificial Intelligence, December 1980.

[RICH81]
Rich, C., "Multiple Points of View in Modeling Programs," *Proc. Workshop on Data Abstraction, Data Bases and Conceptual Modeling, SIGPLAN Notices,* Vol. 16, No. 1, January 1981, pp. 177-179.

[RICH82]
Rich, C., "Knowledge Representation Languages and Predicate Calculus: How to Have Your Cake and Eat it Too," *Proc. AAAI National Conference,* Pittsburgh, Penn., August 1982.

[ROSC75]
Rosch, E., "Cognitive Representations of Semantic Categories," *Journal of Experimental Psychology: General,* Vol. 104, 1975, pp. 192-233.

[ROSS77]
Ross, D.T., "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Transactions on Software Engineering,* Vol. SE-3, No. 1, January 1977.

[ROSS82]
Rosser, B.J., "Highlights of the History of the Lambda-Calculus," *Conference Record ACM Symposium on Lisp and Functional Programming,* Plymouth, Mass., August 1982.

[ROUS77]
Roussopoulos, N., "ADD: Algebraic Data Definition," *Proc. 6th Texas Conference on Computing Systems,* Austin, Texas, November 1977.

[ROUT79]
Routley, R., unpublished manuscript, 1980.

[ROWE82]
Rowe, L.A., *et al.,* "A Form Application Development System," *Proc. 1982 ACM SIGMOD International Conference on the Management of Data,* Orlando, Fla., June 1982.

[RS76]
Rich, C., and H.E. Shrobe, "Initial Report On A LISP Programmer's Apprentice," M.S. thesis (Technical Report MIT/AI/TR-354), MIT Laboratory for Artificial Intelligence, December 1976.

[RS79]
Rowe, L.A., and K.A. Schoens, "Data Abstraction, Views and Updates in RIGEL," *Proc. 1979 ACM SIGMOD International Conference on the Management of Data,* Boston, Mass., May 1979, pp. 71-81.

[RS81]
Reimer, M., and J.W. Schmidt, "Transaction Procedures with Relational Parameters," Report No. 45, Institut fuer Informatik, ETH Zurich, October 1981.

[RSW79]

Rich, C., H.E. Shrobe, and R.C. Waters, "An Overview of the Programmer's Apprentice," *Proc. 6th International Joint Conference on Artificial Intelligence,* Tokyo, Japan, August 1979.

[RSWS78]

Rich, C., H.E. Shrobe, R.C. Waters, G.J. Sussman, and C.E. Hewitt, "Programming Viewed as an Engineering Activity," NSF Proposal, (Memo MIT/AIM-459), MIT Laboratory for Artificial Intelligence, January 1978.

[RUTH73]

Ruth, G., "Analysis of Algorithm Implementations," Ph.D. thesis (MIT Project MAC Technical Report 130), 1973.

[SACE74]

Sacerdoti, E.D., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence,* Vol. 5, No. 2, 1974, pp. 115-135.

[SC75]

Smith, J.M., and P.Y. Chang, "Optimizing the Performance of a Relational Algebra Database Interface," *Communications of the ACM,* Vol. 18, No. 10, 1975.

[SCHM77]

Schmidt, J.W., "Some High Level Language Constructs for Data of Type Relation," *ACM Transactions on Database Systems,* Vol. 2, No. 3, September 1977.

[SCHM78]

Schmidt, J.W., "Type Concepts for Database Definition," *Proc. International Conference on Data Bases,* Haifa, Israel, August 1978.

[SCHM82]

Schmidt, J.W., "Generalized Data Definition and Selection Mechanisms," Fachbereich Informatik, Univ. of Hamburg, 1982 (to appear).

[SCHU71]

Schuman, S.A. (ed.), *Proc. International Symposium on Extensible Languages, SIGPLAN Notices,* Vol. 6, No. 12, December 1971.

[SCHU76a]

Schubert, L.K., "Extending the Expressive Power of Semantic Networks," *Artificial Intelligence,* Vol. 7, No. 2, Summer 1976, pp. 163-198.

[SCHU76b]

Schuman, S.A., "On Generic Functions," in S.A. Schuman (ed.), *New Directions in Algorithmic Languages — 1975,* IRIA, Le Chesnay, France, 1976, pp. 169-192.

[SCHW75]

Schwartz, J.T., "On Programming," Interim Report on the SETL Project, Courant Institute of Mathematical Sciences, New York Univ., June 1975.

[SCOT72]
Scott, D. S., "Lattice Theoretic Models for Various Type-free Calculi," *Proc. 4th International Congress on Logic, Methodology and the Philosophy of Science,* Bucharest, Hungary, 1972.

[SE74]
Stonebraker, M., and E. Wong, "Access Control in a Relational Data Base System by Query Modification," *Proc. ACM Annual Conference,* San Diego, Calif., November 1974.

[SFFH78]
Shaw, M., G. Feldman, R. Fitzgerald, P. Hilfinger, I. Kimura, R. London, J. Rosenberg, and W. A. Wulf, "Validating the Utility of Abstraction Techniques," *Proc. ACM National Conference,* December 1978, pp. 106-110.

[SFL81]
Smith, J. M., S. Fox, and T. Landers, "Reference Manual for ADAPLEX," Technical Report CCA-81-02, Computer Corporation of America, Cambridge, Mass., 1981.

[SGC79]
Schubert, L. K., R. G. Goebel, and N. J. Cercone, "The Structure and Organization of a Semantic Net for Comprehension and Inference," in [FIND79], pp. 121-175.

[SHAP79]
Shapiro, S., "The SNePs Semantic Network Processing System," in [FIND79].

[SHAW79]
Shaw, M., "A Formal System for Specifying and Verifying Program Performance," Technical Report CMU-CS-79-129, Carnegie-Mellon Univ., June 1979.

[SHIP81]
Shipman, D. W., "The Functional Data Model and the Data Language DAPLEX," *ACM Transactions on Database Systems,* Vol. 6, No. 1, March 1981.

[SHM77]
Szolovits, P., L. Hawkinson, and W. A. Martin, "An Overview of OWL, A Language for Knowledge Representation," Technical Memo MIT/LCS/TM-86, MIT Laboratory for Computer Science, 1977.

[SHRO79]
Shrobe, H. E., "Dependency Directed Reasoning for Complex Program Understanding," Ph.D. thesis (Technical Report MIT/AI/TR-503), MIT Laboratory for Artificial Intelligence, April 1979.

[SK77]
Sibley, E. H., and L. Kershberg, "Data Architecture and Data Model Considerations," *Proc. AFIPS National Computer Conference,* Dallas, Texas, 1977.

[SK80a]
Silberschatz, A., and Z. Kedem, "Consistency in Hierarchical Database Systems," *Journal of the ACM,* Vol. 27, No. 1, January 1980.

[SK80b]
Stonebraker, M., and K. Keller, "Embedding Hypothetical Data Bases and Expert Knowledge in a Data Manager," *Proc. 1980 ACM SIGMOD International Conference on the Management of Data,* Santa Monica, Calif., May 1980.

[SL79]
Su, S. Y. W., and D. H. Lo, "A Semantic Association Model for Conceptual Database Design," *Proc. International Conference on the Entity-Relationship Approach to Systems Analysis and Design,* Los Angeles, Calif., December 1979.

[SM72]
Sussman, G. J., and D. McDermott, "Why Conniving is Better Than Planning," Memo MIT/AIM-255A, MIT Laboratory for Artificial Intelligence, 1972.

[SM80]
Schmidt, J. W., and M. Mall, "Pascal/R Report," Report No. 66, Fachbereich Informatik, Univ. of Hamburg, January 1980.

[SMIT82]
Smith, B. C., "Reflection and Semantics in a Procedural Language," Technical Report MIT/LCS/TR-272, MIT Laboratory for Computer Science, May 1982.

[SOWA76]
Sowa, J. F., "Conceptual Structures for a Database Interface," *IBM Journal of Research and Development,* Vol. 20, No. 4, July 1976, pp. 336-357.

[SS75]
Schmid, H. A., and J. R. Swenson, "On the Semantics of the Relational Data Model," *Proc. 1975 ACM SIGMOD International Conference on the Management of Data,* San Jose, Calif., June 1975.

[SS77a]
Smith, J. M., and D. C. P. Smith, "Database Abstractions: Aggregation," *Communications of the ACM,* Vol. 20, No. 6, June 1977.

[SS77b]
Smith, J. M., and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems,* Vol. 2, No. 2, June 1977, pp. 105-133

[SS78a]
Smith, J. M., and D. C. P. Smith, "Principles of Conceptual Database Design," *Proc. NYU Symposium on Database Design,* New York, May 1978.

[SS78b]
Steele, G. L., Jr., and G. J. Sussman, "The Art of the Interpreter, or, The Modularity Complex (Parts Zero, One, and Two)," Memo MIT/AIM-453, MIT Laboratory for Artificial Intelligence, May 1978.

[SS78c]
Steele, G. L., Jr., G. J. Sussman, "The Revised Report on SCHEME: A Dialect of LISP," Memo MIT/AIM-452, MIT Laboratory for Artificial Intelligence, January 1978.

[SS79]
Smith, J. M., and D. C. P. Smith, "A Database Approach to Software Specification," Technical Report CCA-79-17, Computer Corporation of America, Cambridge, Mass., April 1979.

[STAN67]
Standish, T. A., "A Data Definition Facility for Programming Languages," Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon Univ., 1967.

[STEE78]
Steele, G. L., "Rabbit: A Compiler for Scheme (A Study in Compiler Optimization)," Technical Report MIT/AI/TR-474, MIT Laboratory for Artificial Intelligence, May 1978.

[STON75]
Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," *Proc. 1975 ACM SIGMOD International Conference on the Management of Data,* San Jose, Calif., June 1975.

[STON76]
Stonebraker, M., *et al.,* "The Design and Implementation of INGRES," *ACM Transactions on Database Systems,* Vol. 2, No. 3, September 1976.

[STON80]
Stonebraker, M., "Retrospection on a Database System," *ACM Transactions on Database Systems,* Vol. 5, No. 2, June 1980, pp. 225-240.

[STON82a]
Stonebraker, M., *et al.,* "Document Processing in a Relational Database System," Report M82/20, Electronic Research Laboratory, Univ. of California, Berkeley, June 1982.

[STON82b]
Stonebraker, M., *et al.,* "A Rules System for a Relational Database System," *Proc. 2nd International Conference on Databases,* Jerusalem, Israel, June 1982.

[SUNA78]
Sunagren, B., "Database Design in Theory and Practice," *Proc. 4th International Conference on Very Large Databases*, West Berlin, September 1978.

[SUSS75]
Sussman, G. J., *A Computer Model of Skill Acquisition,* MIT Press, 1975.

[SUSS78]
Sussman, G. J., "Slices at the Boundary Between Analysis and Synthesis," in J.-C. Latombe (ed.), *Artificial Intelligence and Pattern Recognition in Computer Aided Design,* Elsevier North-Holland, New York, 1978.

[SW80]
Shaw, M., and W. A. Wulf, "Toward Relaxing Assumptions in Languages and Their Implementations," *SIGPLAN Notices,* Vol. 15, No. 3, March 1980, pp. 45-61.

[SW82]
Schneider, H.-S., and A. I. Wasserman, *Automated Tools for Information Systems Design* (Proc. IFIP WG 8.1 Working Conference on Automated Tools for Information Systems Design and Development, New Orleans, La., January 1982), Elsevier North-Holland, New York, 1982.

[SWC70]
Sussman, G. J., T. Winograd, and E. Charniak, "MICRO-PLANNER Reference Manual," Memo MIT/AIM-203, MIT Laboratory for Artificial Intelligence, 1970.

[SWL77]
Shaw, M., W. A. Wulf, and R. L. London, "Abstraction and Verification in Alphard: Defining and Specifying Iteration and Generators," *Communications of the ACM,* Vol. 20, No. 8, August 1977.

[TARK56]
Tarski, A., *Logic, Semantics, Metamathematics,* Oxford Univ. Press, 1956.

[TF76]
Taylor, D. C., and R. L. Frank, "CODASYL Database Management Systems," *Computing Surveys,* Vol. 8, No. 1, March 1976.

[TF80]
Teorey, T. J., and J. P. Fry, "The Logical Record Access Approach to Database Design," *Computing Surveys,* Vol. 12, No. 2, June 1980.

[TH77]
Teichroew, D., and E. A. Hershey, "PSA/PSL: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Transactions on Software Engineering,* Vol. SE-3, No. 1, January 1977.

[THER82]
Theriault, D., "A Primer for the Act-1 Language," Memo MIT/AIM-672, MIT Laboratory for Artificial Intelligence, April 1982.

[TK78]
Tsichritzis, D., and A. Klug, "The ANSI/X3/SPARC DBMS Framework," *Information Systems,* Vol. 3, No. 4, 1978.

[TL76]
Tsichritzis, D., and F. Lochovsky, "Hierarchical Database Management: A Survey," *Computing Surveys,* Vol 8, No. 1, March 1976.

[TL82]
Tsichritzis, D., and F. Lochovsky, *Data Models,* Prentice-Hall, Englewood Cliffs, N.J., 1982.

[TSIC82]
Tsichritzis, D., "Form Management," *Communications of the ACM,* Vol. 25, No. 7, July 1982.

[TURI37]
Turing, A. M., "Computability and Lambda-Definability," *Journal of Symbolic Logic,* Vol. 2, 1937, pp. 153-163.

[TURN79]
Turner, D. A., "A New Implementation Technique for Applicative Languages," *Software—Practice & Experience,* Vol. 9, No.1, 1979.

[TURN81a]
Turner, D. A., "The Semantic Elegance of Applicative Languages," *Proc. ACM Conference on Functional Programming and Archictecture,* New Hampshire, 1981.

[TURN81b]
Turner, R., "Montague Semantics, Nominalization, and Scott's Domains," unpublished manuscript, 1981.

[TWW78]
Thatcher, J. W., E. G. Wagner, and J. B. Wright, "Data Type Specifications: Parameterization and Power of Specification Techniques," *Proc. SIGACT 10th Symposium on Theory of Computing,* May 1978, pp. 119-132.

[ULLM80]
Ullman, J. D., *Principles of Database Systems,* Computer Science Press, Potomac, Maryland, 1980.

[VASS79]
Vassiliou, Y., "Null Values in Database Management: A Denotational Semantics Approach," *Proc. 1979 ACM SIGMOD International Conference on Management of Data,* Boston, Mass., May 1979, pp. 162-169.

[VASS80]
Vassiliou, Y., "A Formal Treatment of Imperfect Information in Database Management," Ph.D. thesis, Dept. of Computer Science, Univ. of Toronto, 1980.

[VMPK75]
Van Wijngaarden, A., B. J. Maiuoux, J. E. L. Peck, C. H. A. Koster, C. M. Sintzoff, C. H. Lindsay, L. G. L. T. Meertens, and R. G. Fisker, "Revised Report on the Algorithmic Language Algol 68," *Acta Informatica,* Vol. 5, 1975, pp 1-236.

[WALD81]
Wadler, P., "Applicative Style Programming, Program Transformation and List Operators," *Proc. ACM Conference on Functional Programming and Architecture,* New Hampshire, 1981.

[WALK80]
Walker, A., "Time and Space in a Lattice of Universal Relations with Blank Entries," *XP1 Workshop on Relational Database Theory,* Stony Brook, N.Y., June-July 1980.

[WASS77]
Wasserman, A.I., "Procedure-Oriented Exception-Handling," Technical Report 27, Medical Information Science, Univ. of California, San Francisco, February 1977.

[WASS79]
Wasserman, A.I., "The Data Management Facilities of PLAIN," *Proc. 1979 ACM SIGMOD International Conference on the Management of Data,* Boston, Mass., May 1979.

[WATE78]
Waters, R.C., "Automatic Analysis of the Logical Structure of Programs," Ph.D. thesis (Technical Report MIT/AI/TR-492), MIT Laboratory for Artificial Intelligence, December 1978.

[WATE79]
Waters, R.C., "A Method for Analyzing Loop Programs," *IEEE Transactions on Software Engineering,* Vol. SE-5, No. 3, May 1979, pp. 237-247.

[WATE81]
Waters, R.C., "The Programmer's Apprentice: Knowledge Based Program Editing," *IEEE Transactions on Software Engineering,* Vol. SE-8, No. 1, January 1982.

[WB81]
Wirsing, M., and M. Broy, "An Analysis of Semantic Models for Algebraic Specifications," *International Summer School on the Theoretical Foundations of Programming Methodology,* Marktoberdorf, 1981.

[WE79]
Wiederhold, G., and R. El-Masri, "The Structural Model for Database Design," *Proc. International Conference on the Entity-Relationship Approach to Systems Analysis and Design,* Los Angeles, Calif., December 1979.

[WEYH80]
Weyhrauch, R.W., "Prolegomena to a Theory of Mechanized Formal Reasoning," *Artificial Intelligence,* Vol. 13, Nos. 1 and 2, April 1980, pp. 133-170.

[WH78]
Waterman, D.A., and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems,* Academic Press, New York, 1978.

[WILS75]
Wilson, M.L., "The Information Automata Approach to Design and Implementation of Computer-Based Systems," Technical Report FSD76-0093, IBM Federal Systems Division, Gaithersburg, Md., 1975.

[WINO72]
Winograd, T., *Understanding Natural Language,* Academic Press, New York, 1972.

[WINO73]
Winograd, T., "Breaking the Complexity Barrier (Again)," *Proc. SIGIR-SIGPLAN Interface Meeting,* November 1973.

[WINO75]
Winograd, T., "Frame Representation and the Declarative-Procedural Controversy," in [BC75].

[WIRS82]
Wirsing, M., "Structured Algebraic Specifications," *Proc. AFCET Symposium for Computer Science*, Paris, France, March 1982.

[WIRT71]
Wirth, N., "Program Development by Stepwise Refinement," *Communications of the ACM,* Vol. 14, No. 4, April 1971, pp. 221-227.

[WIRT73]
Wirth, N., *Systematic Programming, An Introduction,* Prentice-Hall, Englewood Cliffs, N.J., 1973.

[WIRT77]
Wirth, N., "Modula: A Language for Modular Programming," *Software—Practice & Experience,* Vol. 7, No. 1, January 1977, pp. 3-35.

[WKP80]
Walker, B. J., R. A. Kemmerer, and G. J. Popek, "Specification and Verificaton of the UCLA Security Kernel," *Communications of the ACM,* Vol. 23, No. 2, February 1980, pp. 118-131.

[WLGL78]
Wensley, J. H., L. Lamport, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," *Proc. IEEE,* Vol. 66, No. 10, October 1978, pp. 1240-1255.

[WLS76]
Wulf, W. A., R. L. London, and M. Shaw, "An Introduction to the Construction and Verification of Alphard Programs," *IEEE Transactions on Software Engineering,* Vol. SE-2, No. 4, December 1976, pp. 253-265.

[WM77]
Wong, H. K. T., and J. Mylopoulos, "Two Views of Data Semantics: Data Models in Artificial Intelligence and Database Management," *INFOR,* Vol. 15, No. 3, 1977.

[WM81]
Weinreb, D., and D. Moon, "LISP Machine Manual," MIT Laboratory for Artificial Intelligence, March 1981.

[WONG81]
Wong, H. K. T., "Design and Verification of Interactive Information Systems Using TAXIS," Technical Report CSRG-129, CSRG, Univ. of Toronto, April 1981.

[WOOD75]
Woods, W. A., "What's in a Link: Foundations for Semantic Networks," in [BC75], pp. 35-82

[WPPD83]
Wirsing, M., P. Pepper, H. Partsch, W. Dosch, and M. Broy, "On Hierarchies of Abstract Data Types," *Acta Informatica,* 1983 (to appear).

[WS73]
Wulf, W. A., and M. Shaw, "Global Variables Considered Harmful," *SIGPLAN Notices,* Vol. 8, No. 2, February 1973, pp. 28-34.

[WSH77]
Welsh, J., M. J. Sneeringer, and C. A. R. Hoare, "Ambiguities and Insecurities in PASCAL," *Software — Practice & Experience,* Vol. 7, No. 6, 1977, pp. 685-696.

[WSHF81]
Wulf, W. A., M. Shaw, P. N. Hilfinger, and L. Flon, *Fundamental Structures of Computer Science,* Addison-Wesley, Reading, Mass., 1981.

[YANN82]
Yannakakis, M., "A Theory of Safe Locking Policies in Database Systems," *Journal of the ACM,* Vol. 29, No. 3, July 1982.

[YZ80]
Yeh, R. T., and P. Zave, "Specifying Software Requirements," *Proc. IEEE,* Vol. 68, No. 9, September 1980.

[ZANI77]
Zaniolo, C., "Relational Views in a Database System; Support for Queries," *Proc. IEEE Computer Applications and Software Conference,* Chicago, Ill., November 1977, pp. 267-275.

[ZILL80]
Zilles, S. N., "An Introduction to Data Algebras," in D. Bjoerner (ed.), *Abstract Software Specifications, Lecture Notes in Computer Science,* No. 86, Springer-Verlag, New York, 1980, pp. 248-272.

[ZISM78]
Zisman, M., "Use of Production Systems for Modelling Asynchronous Parallel Processes," in [WH78].

[ZLT82]
Zilles, S. N., P. Lucas, and J. W. Thatcher, "A Look at Algebraic Specification," submitted for publication.