Knowledge Level Engineering: Ontological Analysis

James H. Alexander, Michael J. Freiling, Sheryl J. Shulman,
Jeffrey L. Staley, Steven Rehfuss and Steven L. Messick

Computer Research Laboratory
Tektronix Laboratories

ABSTRACT

Knowledge engineering suffers from a lack of formal tools for understanding domains of interest. Current practice relies on an intuitive, informal approach for collecting expert knowledge and formulating it into a representation scheme adequate for symbolic processing. Implicit in this process, the knowledge engineer formulates a model of the domain, and creates formal data structures (knowledge base) and procedures (inference engine) to solve the task at hand. Newell (1982) has proposed that there should be a *knowledge level* analysis to aid the development of AI systems in general and knowledge-based expert systems in particular. This paper describes a methodology, called **ontological analysis**, which provides this level of analysis. The methodology consists of an analysis tool and its principles of use that result in a formal specification of the knowledge elements in a task domain.

1. Knowledge Engineering needs a methodology.

Traditionally, knowledge engineering has been a difficult process. Neophyte knowledge engineers often "don't know where to start." The difficulty in getting started is related to confusions over how to encode or classify relevant knowledge items from the task domain. Clancey (1985) provides a typical example from MYCIN:

> Perhaps one of the most perplexing difficulties we encounter is distinguishing between subtype and cause, and between state and process ... For example, a physician might speak of a brain-tumor as a kind of brain-mass lesion. It is certainly a kind of brain-mass, but it causes a lesion (cut); it is not a kind of lesion. Thus, the concept bundles cause with effect and location: a *lesion* in the *brain* caused by a *mass* of some kind is a brain-mass-lesion. (pg. 311)

This experience is familiar to any knowledge engineer. Misunderstandings about the knowledge elements in a system often pervade mature systems and cause endless problems. In response to this problem Newell (1982) has suggested that there should be a *knowledge level analysis* of domains which would guide knowledge-based systems development.

In this paper we discuss a methodology for analyzing problem domains we call *ontological analysis*. Most problems encountered in knowledge-based systems derive from *ad hoc* design of the knowledge structures. Often, knowledge is collected by writing rules or frames in a language-specific syntax, without a systematic consideration of the underlying structure of knowledge elements. Ontological analysis focuses attention on the elements of knowledge in their own right, independent of implementation techniques. An ontological analysis is distinctly different from knowledge representation languages in that it presents only a high level description of a problem's knowledge structure. Ontological analysis is used to identify and construct an adequate knowledge representation for a problem.

2. Ontological Analysis.

To philosophers, ontology is the branch of metaphysics concerned with the nature of existence, and the cataloguing of existent entities (Quine, 1980). The role of ontology in AI has been noted previously (Hayes, 1985; Hobbs, 1985; McCarthy, J., 1980). We use the term to emphasize that a knowledge-based system is best designed by careful attention to the step-by-step composition of knowledge structures. An ontology is a collection of abstract objects, relationships and transformations that represent the physical and cognitive entities necessary for accomplishing some task. Our experience indicates that complex ontologies are most easily constructed in a three step process that concentrates first on the (static) physical objects and relationships, then on the (dynamic) operations that can change the task world, and finally on the (epistemic) knowledge structures that guide the selection and use of these operations.

Any useful methodology must contain both *formal tools* for constructing an analysis, and informal *principles of practice* to guide application of the formal tools. Our research has indicated that several different formal tools are useful for extracting and defining ontologies. We are developing a family of languages collectively called SPOONS (SPecification Of ONtological Structure) to encompass tools based respectively on *domain equations*, *equational logic*, and *semantic grammars.*

2.1 Domain Equations in Ontological Analysis

The most useful and concise of these languages is SUPE-SPOONS (SUPErstructure SPOONS), which is based on the domain equations of denotational semantics (Gordon 1979; Stoy 1977) and algebraic specification (Guttag and Horning, 1980). Because of the rich ontologies found in most knowledge engineering problems, domain equations provide a concise and reasonably abstract characterization of the necessary knowledge structures. Furthermore, they do not encourage the knowledge engineer to get prematurely involved in details.

2.1.1 SUPE-SPOONS Syntax and Semantics

SUPE-SPOONS consists of two basic statement types:

- **Domain equations: Site = Building × Campus**. These statements define domains, or *types* of knowledge structures.*
- **Domain element declarations: add_meeting: Meeting → [Meetings → Meetings]** These statements declare the type of specific domain elements.

The right hand side of statements can be composed of one or more domains or constant elements with operators relating these elements. Four primitive domains, STRING, BOOLEAN, INTEGER and NUMBER, are always assumed to be defined. Other primitive domains can be defined by explicit enumeration of their elements, or by open assignment to some collection of atomic elements.

*For most purposes, it suffices to think of domains as sets. A more complex semantics is needed if domains are defined recursively (Stoy 1977) or with multiple equations.

Table 1
Static Ontology Fragment for IEC

Meeting = <atomic>
Project = <atomic>
Department = <atomic>
Person = <atomic>
Scheduled_Meeting = (Meeting × Person)
Meeting_Purpose = One_Time_Meeting_Purpose + Repetitive_Meeting_Purpose
One_Time_Meeting_Purpose = {discuss, plan} × Project
Repetitive_Meeting_Purpose = {staff, project} × Department
Location_Of_Meeting = [Meeting → Location_Description]
Time_Of_Meeting = [Meeting → Time_Description]
Purpose_Of_Meeting = [Meeting → Meeting_Description]
Participants_In_Meeting = [Meeting → Person_Description]
Owner_of_Meeting = [Scheduled_Meeting → Person]

Table 2
Dynamic Ontology for IEC

State = Meetings × Purposes × Required_Participations × [Meeting → Arbitrator] × [Meeting → Reviewer] × [Meeting → Meeting_Plan] × [Person → Schedule] × [Room → Schedule]
Operation = Heuristic_Operation + Algorithmic_Operation + Autonomous_Operation + { schedule_new_meeting }
Heuristic_Operation = {select_arbitrator,select_reviewer}
Algorithmic_Operation = {create_new_meeting,assimilate}
Autonomous_Operation = { signoff_or_propose, assent, arbitrate,initial_proposal}
schedule_meeting = (Purpose × Required_Participation) →[State → State]
create_new_meeting : (Purpose× Required_Participation) → Meeting
select_arbitrator : Meeting× Purpose → Arbitrator
select_meeting_to_act_on : State → Meeting
select_reviewer : Meeting → [State → Reviewer]
initial_proposal : [Arbitrator → Meeting_Plan]
signoff_or_propose : [Reviewer→[Old_Meeting_Plan→New_Meeting_Plan]
arbitrate : (Old_Meeting_Plan × New_Meeting_Plan) → (Meeting_Plan × Continue)
Continue = BOOLEAN
assent : Meeting_Plan → [(Person + Room) → Schedule]
assimilate : ((Meeting × Meeting_Plan) + ((Person + Room) × Schedule) + (Meeting × Arbitrator) + (Meeting × Reviewer)) → [State → State]

- **Explicit enumeration**: Meeting_Room_Accessory = {blackboard, screen, ...}
- **Open assignment:** Meeting = <atomic>

The operators in the domain equations are of five types:

- **Discriminated Union: D + E.** Discriminated union of two domains defines that domain composed of each member of D and E, where original domain identity is preserved.
- **Cross Product: D × E.** Cross product of two domains describes a domain composed of all ordered pairs whose first element is a member of domain D and second element is a member of domain E.
- **Domain mapping: D → E.** Mapping of one domain onto another creates a domain consisting of all functions which map domain D onto domain E.
- **Collection of Sets: 2**D.** Defines the domain consisting of all subsets of D.
- **Collection of Ordered Sets: D*.** Defines the domain of all ordered sequences of D.

2.2 Building an Ontology

We will illustrate our method by using SUPE-SPOONS to build an ontology for the task of scheduling meetings in an Intelligent Electronic Calendar (IEC: Staley, in press). The task of the IEC is to schedule meetings in a semi-automated fashion, and to support the negotiations necessary for determining who will meet when and where. A partial ontology for this task is found in Appendix A. It should be noted that the IEC is in the early stages of design, and our analysis here is for expository purposes only. A complete ontology for a knowledge-based system that has been implemented and runs can be found in Freiling *et al.* (1986).

2.2.1 A Static Ontology for the IEC

Ontological Analysis begins by enumerating the physical objects in the problem domain and identifying their inherent properties and relationships. At the level of the static ontology, the analysis performed is quite similar to the entity-relationship model of Chen (1976).

Table 1 presents a subset of the IEC static ontology. Only those equations relating to the domain `Meeting` are presented. This domain consists of abstract tokens, or surrogates (Codd 1979), each of which represents a single meeting. Another domain, `Scheduled_Meeting`, contains elements which indicate that a meeting has been entered on someone's calendar. A number of mappings are also defined that identify the salient properties of meetings, such as `Time_Description`, `Meeting_Description` and `Person_Description`. The domain `Meeting_Purpose` represents various reasons why a meeting should be held, and embodies the analytical decision that one time meetings are project oriented, while repetitive meetings revolve around organizational units or departments. This is a simplification of course, but what is important is the ease with which such decisions can be expressed.

The IEC static ontology also addresses time scales, meeting room characteristics, and individual membership in projects and departments (see Appendix A for the details). In total the static ontology gives us a picture of all important elements within the domain.

2.2.2 A Dynamic Ontology for the IEC.

Problem solving is often characterized as search through a state space (Simon, 1981; Newell and Simon, 1972). Solution of a problem consists of selecting operators whpse application transforms the current state into another. The dynamic ontology defines a problem space in terms of configurations of elements from the static ontology, and then defines problem operators as transformations built on the domain of problem states.

The dynamic ontology defines which knowledge is unchanged throughout the problem solving process (*i.e.*, organizational charts, see Appendix A) and which knowledge changes as the problem is solved (*i.e.*, schedules and meeting plans). In the IEC, the problem state includes pending meetings, their purposes and required participations, room schedules, individual schedules, and state information recording the negotiation process.

Table 2 shows a sample dynamic ontology for the IEC. To define the dynamic ontology, we must commit to a particular model for the negotiation process. Negotiation is an extremely subtle and complex problem (Davis and Smith, 1981); for this example, we will use a very simplistic model.

The task of scheduling a meeting is called `schedule_meeting`. Significant subtasks include choosing an arbitrator, choosing reviewers, producing proposals, reviewing them, and reserving space on calendars. These operations are divided up into algorithmic operations (no rules or heuristics required), heuristic operations (driven from a rule base to be specified in the epistemic

Table 3
Epistemic Ontology for the IEC

Arbitrator_Selection_Rules =
2 ** (Purpose × Person_Description)
Meeting_Selection_Rules =
2 ** Meeting_Plan_Pattern
Reviewer_Selection_Rules =
2 ** (Meeting_Plan_Pattern × Person_Description)

ontology), and autonomous processes (which do not run in the IEC at all, but represent the behavior of independent actors.)

Our negotiation model proceeds as follows. First, the IEC selects an arbitrator (`select_arbitrator`), who proposes an initial meeting plan (`initial_proposal`). The meeting plan consists of proposals about time, location, and participants of the meeting, coupled with signoffs from participants. A plan is considered complete when the signoff list for each part of the plan is the same as the list of participants.

Once a plan for the meeting is on the table, a reviewer is selected to look at the plan (`select_reviewer`) and either approves it or makes a counter-proposal (`signoff_or_propose`). If a counter-proposal is made the arbitrator selects either the old proposal or the counter-proposal as the standing proposal (`arbitrate`) then selects a new reviewer (`select_reviewer`). After each decision, the arbitrator also decides whether to continue the negotiations (`arbitrate`). If negotiations are halted, all participants are required to accept the current version of the plan (`assent`).

The IEC divides its time between coordinating the negotiation processes for several meetings. It makes use of a heuristic process (`select_meeting_to_act_on`) to decide which meeting to work on at any point in time.

2.2.3 The Epistemic Ontology.

The dynamic ontology defines the operations available to the IEC for performing its task. The epistemic ontology defines knowledge structures to guide the selection and use of these operations. Table 3 shows the epistemic ontology for the IEC.

The epistemic ontology usually contains two different types of knowledge structures. One knowledge type is used to select which operation should be performed. The other knowledge type controls the actual performance of operations. Because our simple negotiation model is so rigid, none of the former appear. In fact, the only knowledge structures that do appear are those needed to guide the operations classified as **heuristic operations** in the dynamic ontology: `select_meeting_to_act_on`, `select_arbitrator`, and `select_reviewer`.

For `select_arbitrator`, we assume that the purpose of the meeting dictates who a likely arbitrator should be (e.g. the ranking manager of the group calling the meeting), so that arbitrator selection rules need only associate certain purpose patterns with descriptions of likely persons.

Rules to select meetings to work on and reviewers to continue negotiation depend on the current version of the meeting plan. Thus, these rules require the definition of a pattern that can successfully match meeting plans. For meeting selection rules, all that is required is that the meeting plan match the pattern -- that will make it a candidate for selection. For reviewer selection rules, the meeting pattern must be associated with some suggestion regarding a reviewer.

To complete the epistemic ontology, the details of patterns and the matching process need to be defined for meeting plans and purposes. The interested reader is referred to Appendix A for some sample definitions.

3. Principles of practice

We have performed ontological analyses on a wide range of domains including a system troubleshooting Tektronix Oscilloscopes (Alexander *et al.*, 1985; Rehfuss *et al.*, 1985; Freiling *et al.*, 1986), MYCIN's medical knowledge (Shortliffe, 1984), design rule checking for nMOS circuitry (Lob 1984), oscilloscope operation and parts of the IEC (Staley, in press). Each analysis has improved our understanding of the problem domain, and the use of SUPE-SPOONS to sketch out designs has helped us clear up many confusing situations.

Through this experience we have built up a series of principles for constructing ontologies with SUPE-SPOONS. A methodology is more than just a formal notation, it also requires guidelines of proper practice. We have identified seven guidelines so far.

1. Begin with physical entities, proceed to their properties and relationships from there. The most accessible elements of any task domain are usually the physical objects and relationships that must be manipulated. Formalizing these provides an easy way to get started.

The recommended procedure for extracting an ontology is to begin by analyzing a paper knowledge base (Freiling *et al.*, 1985) that describes the task domain in English. The paper knowledge base may come from verbal protocols (Ericsson and Simon, 1984), a textbook, or a training manual. The technical vocabulary used in the paper knowledge base provides the initial elements of the static ontology. After these vocabulary elements have been defined is the time to examine the more esoteric dynamic and epistemic realms.

2. The static, dynamic and epistemic ontologies are not strict boundaries, use them loosely. The placement of ontological elements into categories has no formal effect on the semantics of the ontology, the levels only provide a conceptual framework. Arguing about whether a knowledge structure is actually dynamic or epistemic is of little value.

3. Clearly establish the distinction between objects and what they are intended to represent. Two types of object appear in an ontology: first-class and second-class objects. First-class objects (*surrogates* in database terminology; Codd, 1979) cannot be individuated by their properties. Instead they are individuated by identifying tokens (`<atomic>` in SUPE-SPOONS). In the IEC, meetings are first-class objects:

Meeting = <atomic>
Time_Of_Meeting = [Meeting → Time_Description]

Properties of first class objects are expressed via functions that map the objects into their property values.

Second-class objects refer to those elements of an ontology that represent aggregations of other elements. Second-class objects are individuated solely on the basis of common components. For example, consider the following second-class object:

Gregorian_Time_Point=Year×Month×Day×Hour×{00,15,30,45}

Any two calendar dates are equal if they consist of the same year, the same month, hour and quartile. Only when their composite attributes are identical are the elements themselves identical.

The usefulness of this distinction usually does not appear until implementation, and has to do with issues of representing identity and partial knowledge about elements that are beyond the scope of this paper.

4. Understand and separate intensional and extensional entities. There are many cases in knowledge engineering where it is important to distinguish between representatives for the physical objects, and for descriptions or viewpoints of those objects. For the IEC, it is necessary to define (extensional) units of absolute time, and relate them to (intensional) descriptions of time units with respect to one calendar or another. Only then is it possible to represent the fact that descriptions like 1986 (Gregorian) and Showa 60 (Japanese) refer to the same time interval.

A common way to achieve the distinction between extensional representatives of real world objects and intensional representatives of descriptions or classes of such objects is define representatives for the extensional objects with only the bare minimum of structure. In the IEC, for instance,

Real_Time_Point = INTEGER
Real_Time_Interval = (Real_Time_Point × Real_Time_Point)

Here we define the primitive points in time as integers. We associate point 0 with 12 midnight on 1/1/1901, by the Gregorian calendar. The points are 15 minutes apart. Intervals of time can then be represented simply as a pair of points. (There is actually a bit more complexity for dealing with unbounded intervals, see Appendix A).

Intensional descriptions with respect to various calendars can be constructed as necessary from different parts of the description.

Intensional_Time_Point =
    Gregorian_Time_Point +
    Japanese_Imperial_Reign_Time_Point
Gregorian_Time_Point =
    Year × Month × Day × Hour × { 00, 15, 30, 45 }
Japanese_Imperial_Reign_Time_Point =
    Era × Year × Month × Day × Hour × { 00, 15, 30, 45 }

Finally, intensional description can be related to extensional descriptions by the use of various interpretation functions:

interpret : [Intensional_Time_Point → Real_Time_Point]

Extensional identity of descriptions of varying sorts can then be defined as equality of the image under the relevant interpretation functions.

5. Build relevant abstractions through the use of generalization and aggregation. Generalization and aggregation (Smith and Smith) are common techniques for building large knowledge structures. It is interesting to note that generalization and aggregation steps have a direct manifestation as discriminated unions and Cartesian Products:

GENERALIZATION: Car = (Compact + Luxury_Car + Truck)
AGGREGATION: Car_Assembly = (Engine × Chassis × Body × Drive _ Train)

An ontology may also contain many implicit generalization and aggregation relations. Even the properties of a first class object are implicitly aggregated through the fact that some particular car defines values for each:

Car = <atomic>
Type = [Car → { Compact, Luxury_Car, Truck }]
Has_Engine = [Car → Engine]
Has_Chassis = [Car → Chassis]
Has_Body = [Car → Body]
Has_Drive_Train = [Car → Drive_Train]

Note also that generalizations are implicit in the properties of objects as well. Instances of the domain Car, for example, can be decomposed into Compacts, etc. on the basis of common images under the Type function.

6. Encode rules as simple associations, and heuristic steps as mappings between domains. Novices at Ontological Analysis are tempted to define rules in a form like;

Gate_Recognition_Rules = [Circuit → Gate_Type]

This mapping (from an Ontological Analysis of RUBICC (Lob 1984)) to choose a gate type from a circuit fragment really describes the heuristic task that uses rules, rather than of the rules themselves. We prefer to analyze rules as simple aggregations, because this makes it easier to spot multiple uses for the same rule structure:

Gate_Recognition_Rules = 2 ** (Transistor_Pattern ×Gate_Type)
recognize : Gate_Recognition_Rules → [Circuit → Gate_Type]
synthesize : Gate_Recognition_Rules → [Gate_Type → Circuit]

Another advantage of separating the rules from the heuristic task is that it focuses explicitly on the need to define classes of patterns and matching criteria.

Table 4
Glib Fragment

<signal value> ::= 'HIGH' | 'LOW'
<signal> ::= 'SIGNAL-'<integer>
<atomic signal predicate> ::= <signal> IS <signal value>
<signal predicate> ::= <atomic signal predicate>
| <atomic signal predicate> 'when' <atomic signal predicate>

7. Ensure the compositionality of elements. This case illustrates vividly the usefulness of our methodology. We first encountered this problem in the process of building a semantic grammar (GLIB) in order to extract the ontology of electronic instrument behavior (Freiling *et al.*, 1984). Table 4 shows a fragment of GLIB that can generate the following atomic signal predicate.

SIGNAL-3 IS HIGH

Initially we assumed that this signal predicate would map signals into Boolean values. However, the semantics of two such statements combined with the connective `when` was not at all clear. If `when` was assumed to produce a Boolean itself, then the result would be returned by one of the 16 truth function of two Boolean values, clearly not what we had intended.

SIGNAL-3 IS HIGH when SIGNAL-4 IS LOW

Using domain equations to analyze the problem,

Signal = [Time → Value]
Signal_Predicate =
  [(Signal × Signal_Value) → BOOLEAN]
  ≡ [([Time → Value] × Signal_Value) → BOOLEAN]

we discovered that our signal predicate as defined was dropping the temporal information and performing a global comparison with the threshold value. This problem was solved by creating a more appropriate definition for `Signal_Predicate`, which follows:

Signal_Predicate =
  [(Signal × Signal_Value) → [Time → BOOLEAN]]
  ≡ [([Time → Value] × Signal_Value)
      → [Time → BOOLEAN]]
when: [[[Time → BOOLEAN] × [Time → BOOLEAN]]
      → [Time → BOOLEAN]]

Thus, the comparison made by the `Signal_Predicate` is made *at each instant of time*, so that the result is not a single truth value computed from the whole signal, but a truth value for every time unit of the signal. This makes it possible for when to preserve its `when` functional character, since the truth function (logical and) is now applied on a point by point basis. The compositional analysis of this type of problem is common to researchers familiar with the techniques of semantics and model theory (Allen, 1981). Our hope is that a language like SUPE-SPOONS can make such techniques available to practitioners as well.

4. Future Work

There are a number of weaknesses with the ontological analysis technique as currently defined. Even so, we have found the methodology useful for conceptualizing a knowledge engineering problem, and creating a forum for cogent discussion. Consequently, we actively use the ontological technique on a day to day basis.

Simultaneously, we are defining the theoretical foundations of the methodology. Our goal is to create a formal mathematical system for ontological analysis of problem solving domains. Formal systems allow the creation of tools for automatically checking and organizing the resulting analysis, automating the creation of some components of the ontological systems.

We feel that SUPE-SPOONS provides a valuable tool that enables knowledge engineers to sketch out solutions to knowledge engineering problems at a fairly high level of abstraction. The limitations of domain equations prevent a premature attention to the low-level details of a domain. Eventually, however, those details do need to be addressed. We feel that this is best accomplished with a separate language, and are actively working on another member of the SPOONS family (T-SPOONS) that uses equational logic to define and constrain actual domain elements beyond simply naming their types.

In practice, we have found it hard to get consistent analyses from different knowledge engineers. Only experience will show us the proper formulation of the methodology. Presently, there is no standard concept of a *virtual machine* to be assumed when the analysis is being performed. Denotational semantics, for example has implicitly assumed concepts such as *stack* that form the virtual machine for programming language analysis. We are working to establish a standard to serve as a basis for the methodology.

Finally, we are working to connect our work with other theoretical work on the nature of the knowledge level. Specifically, we see two connections with Clancey's (1985) recent analysis of classification problem solving. First, his notions of generalization, aggregation and heuristics have a more formal description in our formalism. Second, Clancey suggests that problem solving techniques compose to form larger knowledge-based systems. Ontological analysis can provide a means to highlight this composition process. For both of these concepts, we hope eventually to be able to build demonstrations that connect these higher level tasks with the primitive ontological elements of the problem domain.

5. Summary

We have presented a technique, ontological analysis, that has much promise as a knowledge engineering methodology. Methodologies of this type will release the discipline from *ad hoc* descriptions of knowledge and provide a principled means for a knowledge engineer and expert to analyze the elements of a problem domain and communicate the analysis to others. The abstract level at which domain equations characterize the semantics of structures and procedures, not specifying too much detail, help in this regard.

The effectiveness of a technique depends critically on the formulation of more and better principles to guide its use. Such principles only come painfully with much practice. We invite other knowledge engineers to try this approach, and relate their experiences.

6. References

Alexander, J.H., M.J. Freiling, S.L. Messick & S. Rehfuss. Efficient Expert System Development through Domain-Specific Tools. *Fifth International Workshop on Expert Systems and their Application,* Agence de l'Informatique Etablissement Public National, Avignon, France, May, 1985.

Alexander, J.H. & M.J. Freiling. Smalltalk-80® Aids Troubleshooting System Development. *Systems and Software, 4, 4,* April, 1985.

Allen, J.F. An Interval-Based Representation of Temporal Knowledge. In *Proc. IJCAI-1981,* Vancouver, British Columbia, Canada, August, 1981.

Chen, P.P. The Entity-Relationship Model -- Toward a Unified View of Data. *ACM Transactions on Database Systems, 1, 1,* March, 1976.

Clancey, W.J. Heuristic Classification. *Artificial Intelligence, 27, 289-350, 1985.*

Codd, E.F. Extending the Database Relational Model to Capture More Meaning. *ACM TODS 4:4,* December 1979, 397-434.

Davis, R. & R.G. Smith. *Negotiation as a Metaphor for Distributed Problem Solving.* Artificial Intelligence Laboratory Memo, 624, MIT, May 1981.

Ericsson, K.A. & H.A. Simon. *Protocol Analysis.* MIT Press; Cambridge, MA, 1984.

Freiling, M.J., J.H. Alexander, S.L. Messick, S. Rehfuss & S. Shulman. Starting a Knowledge Engineering Project - A Step-by-Step Approach. *A.I. Magazine, 6, 3,* Fall, 1985.

Smalltalk-80 is a registered trademark of Xerox corporation.

Freiling, M.J. & J.H. Alexander. Diagrams and Grammars: Tools for the Mass Production of Expert Systems. *First Conference on Artificial Intelligence Applications,* IEEE Computer Society, Denver, Colorado, December, 1984.

Freiling, M.J., J.H. Alexander, D. Feucht & D. Stubbs. GLIB - A Language for Describing the Behavior of Electronic Devices. Applied Research Technical Report, CR-84-12, April 6, 1984; Tektronix, Inc., Beaverton, OR.

Freiling, M.J., S. Rehfuss, J.H. Alexander, S.L. Messick, & S. Shulman. The Ontological Structure of a Troubleshooting System for Electronic Instruments. *First International Conference on Applications of Artificial Intelligence to Engineering Problems,* Southampton University, U.K., April, 1986.

Gordon, M.J.C. *The Denotational Description of Programming Languages.* Springer Verlag; New York, NY, 1979.

Guttag, J. & J.J. Horning. *Formal Specification as a Design Tool.* Xerox PARC Technical Report, CSL-80-1, January, 1980.

Hayes, P.J. Naive Physics I: Ontology for Liquids. In J.R. Hobbs & R.C. Moore (Eds.), *Formal Theories of the Commonsense World.* Ablex Publishing; Norwood, NJ, 1985.

Hobbs, J.R. Ontological Promiscuity. *23rd Annual Meeting of the ACL,* Chicago, July, 1985.

Lob, C. *RUBICC: A Rule-Based Expert System for VLSI Integrated Circuit Critique.* Electronic Research Laboratory Memo UCB/ERL M84/80, University of California, Berkeley, 1984.

McCarthy, J. Circumscription - A form of non-monotonic reasoning. *Artificial Intelligence, 13,* 1980, 27-39.

Newell, A. The Knowledge Level. *Artificial Intelligence, 18,* pp. 87-127, 1982.

Newell, A. & H.A. Simon. *Human Problem Solving.* Prentice-Hall; Englewood Cliffs, N.J., 1972.

Rehfuss, S., J.H. Alexander, M.J. Freiling, S.L. Messick & S.J. Shulman. *A Troubleshooting Assistant for the Tektronix 2236 Oscilloscope.* Applied Research Technical Report, CR-85-34; Tektronix, Inc.; Beaverton, OR; September 25, 1985.

Simon, H.A. *The Sciences of the Artificial.* The MIT Press; Cambridge, MA; 1981.

Quine, W.V.O. *From a Logical Point of View.* Harvard University Press; Cambridge, MA; 1980.

Shortliffe, E.H. Details of the Consultation System. *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project,* Addison-Wesley; Reading, MA, 1984.

Smith, J.M. & D.C.P. Smith. Database Abstractions: Aggregation and Generalization. *ACM Transactions on Database Systems, 2:2,* June, 1977.

Staley, J.L. An Intelligent Electronic Calendar: A Smalltalk-80® Application. *Tekniques,* in press, Information Display Group, Tektronix, Wilsonville, OR.

7. Appendix A: Ontology for IEC

Static Ontology

```
Person = <atomic>
Persons = 2**Person
Project = <atomic>
Department = <atomic>
Scheduled_Meeting = ( Meeting × Person )
Meeting_Room = <atomic>
Name = <string>
Group = <atomic>

Meeting_Room_Accessory = {blackboard, screen, projector}
Chair_Arrangement_Type = {conference, classroom, auditorium}
Meeting = <atomic>
```

Meeting_Purpose = One_Time_Meeting_Purpose
+ Repetitive_Meeting_Purpose
One_Time_Meeting_Purpose = {discuss, plan, review} × Project
Repetitive_Meeting_Purpose = { staff, project } × Department

Meeting_Proposal =
Time_Proposal + Location_Proposal + Participant_Proposal

Time_Proposal = Time_Description
Location_Proposal = Location_Description
Participant_Proposal = Persons

Reflection = [Scheduled_Meeting → Meeting]

Person_Name = [Person →Name] = [Name → Person]
Person_Attribute =
Name + [Name × Hierarchical_Link]
+ [{rep_of} × Group]
+ [{resp_rep_of} × Group]
+ [{head_of} × Group]
Person_Description = 2**Person_Attribute

Hierarchical_Link = {boss_of, subordinate_of}*
Organization_relation_of_Person = Hierarchical_Link

Concession_Type = {time, location, ...}
Owes_Concession_To = [(Person × Person) → Concession_Type*]
negotiating_points :
[(Person × Person × Concession_Type*
× Organization_Relation_of_Person) → INTEGER

Group_Contained_By = [Group → Group]
Member_Of_Group = [Person → Group]
Project_Name = [Project → <string>]

Location_Description = {Room_Capacity} × INTEGER
× {blackboard, no_board}
Room_Has = Meeting_Room → 2**Meeting_Room_Accessory
Room_Capacity = Meeting_Room → INTEGER
Chair_Arrangement_In_Room
= Meeting_Room → Chair_Arrangement_Type
Building = <atomic>
Campus = <atomic>
Site = Building × Campus
At = [Meeting_Room → Site]

Quarter = { 00 , 15 , 30 , 45 }
Hour = {0..24}
Date = {1..31}
Month = {1..12}
Year = {-BB .. +BB}
Cycle = {-BB' .. +BB'}
Year' = {000, 100, ..., 900}
Ap = {1..13}
Day = {1..28}
Identified_Time_Interval = [Real_Time_Point → INTEGER]
Calendar = [Real_Time_Interval → Calendar_Interval]
Real_Time_Interval =
[Real_Time_Point × Real_Time_Point]
+ [Real_Time_Point × {unbounded}]
+ [{unbounded} × Real_Time_Point]
+ [{unbounded} × {unbounded}]
Event_Description = Interval_Description × Meeting_Description
Interval_Description =
[{between} × Calendar_Point × Calendar_Point]
+ [{before} × Interval_Description]
+ [{after} × Interval_Description]
+ [{before, after, during} × Event_Description]
Calendar_Region = <atomic>
Calendar_Point = Gregorian_Point + Japanese_Point
Calendar_Interval = Calendar_Point × Calendar_Point
Point_Description =
Calendar_Point + [{after} × Calendar_Point]
+ [{before} × Calendar_Point]
+ [{within} × Interval_Description]

Gregorian_Point = Year × Month × Day × Hour × Quarter
Japanese_Point = Era × Year × Month × Day × Hour × Quarter
express_as : [Calendar → [Real_Time_Point → Calendar_Point]
interpret_as : [Calendar → [Calendar_Point → Real_Time_Point]

Event = Scheduled_Meeting + Block_Schedule
Events = 2**Event
Assignments = [Event → Real_Time_Interval]
Schedule = Events × Assignments
Block_Schedule = {read, errand, fill_out_form} × Time_Quantum

Dynamic Ontology

Meeting_Plan = [Meeting_Proposal → Signoffs]
Signoffs = Persons
Arbitrator = Person
Reviewer = Person
Participant = Person
Old_Meeting_Plan = Meeting_Plan
New_Meeting_Plan = Meeting_Plan

State = Meetings × Purposes × Required_Participations
× [Meeting → Arbitrator] × [Meeting → Reviewer]
× [Meeting → Meeting_Plan] × [Person → Schedule]
× [Room → Schedule]
Operation = Heuristic_Operation + Algorithmic_Operation
+ Autonomous_Operation + { schedule_new_meeting }
Heuristic_Operation = { select_arbitrator , select_reviewer ,
select_meeting_to_act_on }
Algorithmic_Operation = { create_new_meeting, reserve}
Autonomous_Operation = { signoff_or_propose , assent , arbitrate ,
initial_proposal }

schedule_meeting = (Purpose ×Required_Participation)
→[State → State]
create_new_meeting : (Purpose×Required_Participation)→Meeting
select_arbitrator : Meeting×Purpose → Arbitrator
select_meeting_to_act_on : State → Meeting
select_reviewer : Meeting → [State → Reviewer]

initial_proposal : [Arbitrator → Meeting_Plan]
signoff_or_propose : [Reviewer → [Old_Meeting_Plan
→ New_Meeting_Plan]
arbitrate : (Old_Meeting_Plan ×New_Meeting_Plan)
Continue = BOOLEAN
assent : Meeting_Plan → [(Person + Room) → Schedule]
assimilate :
((Meeting ×Meeting_Plan) +
((Person + Room) ×Schedule) +
(Meeting ×Arbitrator) +
(Meeting ×Reviewer)) →
[State → State])

Epistemic Ontology

Arbitrator_Selection_Rules =
2 ** (Purpose ×Person_Description)
Meeting_Selection_Rules =
2 ** Meeting_Plan_Pattern
Reviewer_Selection_Rules =
2 ** (Meeting_Plan_Pattern ×Person_Description)
Meeting_Plan_Pattern =
((Time_Pattern ×Signoff_Pattern)
(Location_Pattern ×Signoff_Pattern)
(Participant_Pattern ×Signoff_Pattern))
Time_Pattern = Time_Description + { anytime }
Location_Pattern = 2 ** Location_Description + { anywhere }
Participant_Pattern = 2 ** Person_Description + { anybody }
Signoff_Pattern = 2 ** Person_Description + { anybody } +
{ nobody_but_proposer }