First generation knowledge-based systems (KBSs) typically used a single representational paradigm—such as production rules—to meet all requirements. More ambitious goals have led largely to the abandonment of this simple model. These goals include: (i) construction of systems that encapsulate more aspects of human expert behavior; and (ii) reduction in the cost of system construction and maintenance; and (iii) improvement of time/space performance. As a result, current systems tend to use multiple representations—such as objects, rules, constraints, tasks, and procedures—each tuned to handle specialized classes of information and situations. A number of designers have found it advantageous to integrate the different paradigms via an object-oriented foundation, or substrate.

To understand the utility of an object-oriented substrate, we consider the process of KBS development. In current practice the transfer of expertise from a domain specialist to a knowledge-based system involves a computer scientist intermediary—or knowledge engineer. The specialist and the engineer discuss the domain in a series of interactions. During each interaction, the engineer gathers some understanding of a portion of the specialist's knowledge, encodes it in the evolving system, discusses the encoding and the results of its application with the specialist, and refines the encoded knowledge. The process is a painstaking one—expensive and tedious. As a result, one of the foremost problems that has been identified for KBSs is the knowledge acquisition bottleneck.

For our purposes, we can observe that two sorts of software tools and methods are crucial to a smooth KBS development scenario: (i) those that simplify the process of incremental system refinement; and (ii) those that allow the domain specialist to interact directly to some extent with the evolving system in order to: examine the knowledge base for gaps, weaknesses, and misunderstandings; and expand, refine, and correct the encoded knowledge. Such tools have the potential to increase the bandwidth of the interaction between domain specialist, knowledge engineer, and system by allowing attention to be focused on the required domain knowledge and problem-solving methodology—without being bogged down in programming mechanics.

Object-oriented programming provides powerful assistance for the KBS developer. It has been found to be an effective aid to the exploratory programming style that characterizes the incremental refinement process of KBS development. One of the primary problems facing the KBS developer is management of complexity in a dynamic environment. This is increasingly true as KBSs are scaled-up to meet the demands of real-world problems. A powerful aid to managing complexity is a clear conceptual model of the evolving system—a kind of computational skeleton. In constructing a clear model, it is helpful to organize computation around programming constructs whose internal structure and interrelationships explicitly reflect
those of constructs in the physical world in which the resultant systems are to operate. It is in
constructing such models that the object-oriented style excels. Furthermore, the object-
oriented style encourages modular code with well-defined interfaces through its use of
messages as an invocation form. Inheritance of properties simplifies code-sharing. This leads
to more space-efficient code and to ease of maintenance and alteration. In addition—and of
primary importance—the use of messages allows many traditional KBS representational
paradigms (e.g., rules, constraints) to be encoded as objects and invoked (via messages) in a
uniform manner.

A clear model of the evolving system is as important to the domain specialist as it is to the KBS
developer. Objects appear to be a natural and understandable knowledge organizing
mechanism to humans not normally involved in computation. The concept of a prototype for
encapsulating information—both data and procedures—is well-understood and used by humans,
as are the concepts of classes and instances, taxonomies—and taxonomic inheritance of
properties (along with a number of other forms of inheritance). In addition, a number of
powerful systems have been developed for easy graphical examination and extension of
knowledge bases. A keynote to these systems is that the domain specialist is able to express
himself and interact in the natural terms and notation of his domain.

In the first part of the presentation, we review our experience with an example of an integrated
knowledge-based system development environment based on an object-oriented substrate
and demonstrate the advantages of the approach.

We have been discussing the utility of an object-oriented substrate for reducing the knowledge
acquisition bottleneck by increasing the bandwidth of interactions between domain specialist,
knowledge engineer, and knowledge-based system. There are two intended effects: (i) a
reduction in the time/cost required for KBS construction and maintenance; and (ii) an increase
in the robustness and power of the resultant systems. There is a third potential payoff that
might accrue from the use of objects as a representational substrate. We are searching for a
representational substrate that will allow knowledge to be encoded in such a way as to permit
knowledge bases developed in the context of one application to be used almost as is in related
applications in the same domain. Indeed some aspects of the encoded knowledge bases
should be usable across domains. In the short term, this would enable amortization of the cost
of knowledge acquisition. In the longer term, it could lead to knowledge-sharing between
specialists—a kind of computational publication. Unlike the knowledge involved in traditional
journal publication, however, this knowledge will be computationally described, directly usable
knowledge. Such a development could have enormous implications and could vastly increase
the leverage of an individual domain specialist. It could greatly shorten the time delays
currently involved in making a new development the widely used standard practice.

In order to obtain the desired effect, we believe that architectural principles of knowledge-
based system design must emerge. The current practice in the field will not lead to the
advantages we seek regarding robustness, power, and cost of construction and maintenance.
Below, we list some candidate principles.

• A clear, expressive domain model is central.
• It is desirable to encode knowledge through abstract relations that are usable across
domains (e.g., part/whole, generalization/specialization, task/subtask).
• Knowledge should be partitioned wherever possible (e.g., into domain-independent,
domain-specific, and task-specific packets).
• Assumptions about the context in which knowledge will be used should be minimized.
• Control should be explicitly represented (i.e., strategy knowledge, problem-solving state, and problem-solving history).
• Specialized representations are essential (e.g., objects, rules, constraints, tasks, procedures) for high performance. They must be integrated.
• User interaction should be considered as an integrated component.

In the second part of the presentation, we discuss these principles and the utility of object-oriented programming in following them.
Representation in Knowledge-Based Systems

Reid G. Smith

Schlumberger-Doll Research
Old Quarry Road
Ridgefield CT, 06877-4108
USA
Themes

• State of the art knowledge-based systems orchestrate multiple representational paradigms

• An object-oriented substrate offers useful tools for integration of the different paradigms

• Architectural principles of knowledge-based systems are emerging

• An object-oriented substrate helps developers follow the principles
Impacting the Knowledge Acquisition Bottleneck

 KNOWLEDGE ENGINEERING TOOLS

• Assist the Knowledge Engineer and Domain Specialist to Focus on the Domain Knowledge and Problem-Solving Methodology

• Simplify the Incremental Refinement Process

• Allow Direct Interaction by the Domain Specialist

  ... Expression and Interaction in Domain Terms

• Amortize KB Construction Cost via Knowledge Sharing

  ... Architectural Principles

MACHINE LEARNING TECHNIQUES

• Learning Apprentice Systems
Structured Object Representation

- Modularity: Encapsulation of Properties and Behavior
- Specialized Procedure Invocation via Messages
- Abstraction: Taxonomic Hierarchies...
  - Inheritance of Properties
- Event-Driven Procedure Invocation

... Useful for Managing Complexity

- Prototypes: a Natural and Understandable Knowledge Organizing Mechanism
- Computational Skeleton: Structure for the Reasoning System
- Messages: a Universal Invocation Mechanism
Graph of PROGENY for Lithology in GEOLOGY

- Lithology
  - SedimentaryRock
    - Sylvite
    - SiliciclasticRock
    - Salt
    - MixedLithology
    - MiscellaneousRock
    - Halite
    - Gypsum
    - Evaporite
    - CarbonateRock
    - Anhydrite
  - MetamorphicRock
    - IgneousRock
      - IntrusiveRock
      - ExtrusiveRock
Graph of PARTS for DeltaicPlain in GEOLOGY

DeltaicPlain
  \arrow{UpperDeltaicPlain}
  \arrow{LowerDeltaicPlain}

Swamp
OverBankLevee
Lake
RiverChannel
CrevasseFan
Marsh

CrevasseChannel
CrevasseSplay
MEDICAL TAXONOMY

Bacterium
  G-Rod
  G + Rod
  G + Coccus

G-Rod
  Enterobacteriacum
    E.Coli
    Klebsiella
    Proteus
  Hemophilus
  Diptheroids

G-Coccus
  Neisserium
    Gonococcus
    Meningococcus
  Staphylococcus
  Streptococcus
## Hearsay II

<table>
<thead>
<tr>
<th>LEVELS</th>
<th>KNOWLEDGE SOURCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE INTERFACE</td>
<td>![Diagram]</td>
</tr>
<tr>
<td>PHRASE</td>
<td></td>
</tr>
<tr>
<td>WORD SEQUENCE</td>
<td>![Diagram]</td>
</tr>
<tr>
<td>WORD</td>
<td></td>
</tr>
<tr>
<td>SYLLABLE</td>
<td></td>
</tr>
<tr>
<td>SEGMENT</td>
<td></td>
</tr>
<tr>
<td>PARAMETER</td>
<td></td>
</tr>
</tbody>
</table>
Strobe Datatypes

Object

Datum-Print:
Value: Sys/ObjectPrint
Datatype: Lisp

Datum-Edit:
Value: Sys/ObjectEdit
Datatype: Lisp

Log

Well:
Value:
Datatype: Well

Index:
Value:
Datatype: Index

Display:
Value: DisplayLog
Datatype: Lisp

Well

Company:
Value:
Datatype: Text

Logs:
Value:
Datatype: Log

Gr-3

Well:
Value: Well-1
Datatype: Well

Index:
Value: Depth-10
Datatype: Depth

(MESSAGE 'Gr-3 'Well 'Edit)
received by
Sys/ObjectEdit
Strobe

Well
- Company:
  - Value: Text
  - Datatype: Text
- Logs:
  - Value: Log
  - Datatype: Log

Log
- Well:
  - Value: Well
  - Datatype: Well
- Index:
  - Value: Index
  - Datatype: Index
- Display:
  - Value: DisplayLog
  - Datatype: Lisp

Well-1
- Company:
  - Value: Japex
  - Datatype: Text
- Logs:
  - Value: Gr-3, Sp-2
  - Datatype: Log

Gr-3
- Well:
  - Value: Well-1
  - Datatype: Well
- Index:
  - Value: Depth-10
  - Datatype: Depth

Knowledge Base
Strobe Summary

- Knowledge Bases, Objects, Slots, Facets
- Tangled Taxonomic Hierarchies
  - Class/Instance Distinction
  - Flexible Slot/Facet Inheritance with Caching
- Procedural Attachment: Invocation via Message
  - Datatype Forwarding
- Event-Driven Procedure Invocation
  - Object/Slot Creation/Deletion
  - Slot Value Access/Alteration [before/after]
- Groups: Arbitrary Object Collections
- Multiple Resident Knowledge Bases
- Multiple Machine/Language Operation
- Object/Slot Synonyms
- Description Object Instantiation
Individual Rule Object

**Object:** NormalFault9  
**Generalizations:** Rule  
**If:** (Condition1 Condition2)  
**Then:** (Action1 Action2)  
**Condition1:** (ThereExists z NormalFaultZone)  
**Condition2:** (ThereExists p RedPattern  
\( $< \text{p.Length RedLength} \)  
\( $\text{Above p z} \)  
\( $\text{Perpendicular p.Azimuth z.Strike} \))  
**Action1:** ($Specialize z 'LateFaultZone)  
**Action2:** ($Assign 'DirectionToDownthrownBlock z  
\text{p.Azimuth})  
**Translation:**  
If  
1. there exists an instance of the class NormalFaultZone (z), and  
2. there exists an instance of the class RedPattern (p) such that the Length of p < RedLength, and such that p is above z, and such that the Azimuth of p is perpendicular to the Strike of z  
Then  
1. specialize z to be a LateFaultZone  
2. the DirectionToDownthrownBlock of z \( \leftarrow \) the Azimuth of p
Individual Rule Object

Object: Tidal1
Generalizations: Rule

... Translation:

If

1. there exists a Transition/InnerShelfZone \( (z) \) such that the influence of \( z \) is Wave/Tide, and
2. there exists a set of BluePatterns \( (p) \) such that each new element of \( p \) is within \( z \), and such that each new element of \( p \) is below the last element of \( p \), and such that the Azimuth of each new element of \( p \) is opposite to the Azimuth the last element of \( p \), and such that the size of the set \( p > 1 \)

Then

1. create a TidalFlatZone \( (tz) \)
2. the Top of \( tz \) ← the Depth of the first element of \( p \)
3. the Bottom of \( tz \) ← the Depth of the last element of \( p \)
4. the Axis of \( tz \) ← the Azimuth of the last element of \( p \)
Rule Class Object

**Object:** Rule

**Generalizations:** Object

- If:
- Then:

**Ruleset:**

**Translation:**

**Apply:** ApplyRule

**Match:** MatchRule

**MatchAll:** MatchRuleAll

**Execute:** ExecuteRule

**Translate:** TranslateRule
Individual Ruleset Object

**Object:** DeepMarineRuleset

**Generalizations:** Ruleset
  
  **NormalRules:** Marine-20 Marine-21 Marine-22
  
  **ControlStrategy:** ReStartAfterFiring
Ruleset Class Object

**Object:** Ruleset

**Generalizations:** Object

- NormalRules:
- FireOnceRules:
- FireAlwaysRules:
- ControlStrategy:
- Termination Condition:
- KnowledgeBase:

**Apply:** ApplyRuleset

**ApplyInProcess:** ApplyRulesetInProcess
Constrained Object

Object: WellLocation
Type: Class
Generalizations: Object

Well:
TOWN: {Township}
STAT: {State Province}
NATI: {Nation Country}
CONT: {Continent}

Value:
Datatype: Expr
Candidates: (Europe North-America South-America
Asia Africa Australia)
Constraints: (MembershipConstraint)
Single Slot Constraint Object

Object: MembershipConstraint
Generalizations: SingleSlotConstraint

If {Condition}: (Condition1 )
Then {Correction}: (Action1 )
Condition1: (NOT (MEMBER Value Candidates))
Action1: (ERROR Value "is not one of:" Candidates)

SetOrElementConstraint: Element
Datatype Object for Integrity

**Object:** Datatype

**Generalizations:** Root

**Datum-Get:** StandardGetHandler

... 

**Datum-Put:** IntegrityPutHandler

**Datum-Add:** IntegrityAddHandler

**Datum-Remove:** IntegrityRemoveHandler

...
Impulse: An Extensible and Interactive Knowledge Base Editor

- Reactive Environment
- Customized Presentation and Interaction
- Flexible Display of Knowledge Base Structure
**Rule Commands**

- Exit
- Order LHS
- Order RHS
- Rename This Rule
- Translate Rule
- Delete This Rule

**SE Commands**

- Edit Facets
- Inspect Value
- Message
- Set Value
- Delete this Slot

**Clause Commands**

- New LHS
- New RHS
- Rename Clause
- Delete Clause

**Ancestry**

- Report Example Rule

---

**Rule Editor: NormalFault9 in KB TESTRULES**

**Object:** NormalFault9

**Synonyms:**

**Groups:**

**Type:** INDIVIDUAL

**Edited:** 19-Jan-85 16:35:07  By: KEIO

**IF:**

**Condition1:** (THERE-EXISTS :Y NORMAL-FAULT-PROJECTION)

**Condition2:** (THERE-EXISTS :Z RED-PATTERN

\[ \langle \text{THE LENGTH} :Z \rangle \]

\[ \text{RedLength} \]

\[ \langle \text{ABOVE} :Z :Y \text{ Overlap Gap} \rangle \]

\[ \langle \text{PERPENDICULAR} \ (\text{THE AZIMUTH} :Z) \ (\text{THE STRIKE} :Y) \ (\text{Tolerance}) \rangle \]

**THEN:**

**Action1:** ($\text{SPECIALIZE} :Y \text{ to be a LATE-FAULT-PROJECTION})

**Action2:** ($\text{ASSIGN DIRECTION-TO-DOWNTHROWN-BLOCK of} :Y \text{ to be} (\text{THE AZIMUTH} :Z))

**Rule Slots:**

- RULESET(\(\uparrow\)):
- SOURCE(\(\uparrow\)):
- AUTHOR(\(\uparrow\)):
- BREAK(\(\uparrow\)):
- TRANSLATION:

**IF:**

1. There exists an instance of the class NORMAL-FAULT-PROJECTION (:Y)
2. There exists an instance of the class RED-PATTERN (:Z)
   - such that the LENGTH of :Z < RedLength, and
   - such that :Z is above :Y within [Overlap, Gap], and
   - such that the AZIMUTH of :Z is perpendicular to the STRIKE of :Y within Tolerance

**THEN:**

1. Specialize :Y to be a LATE-FAULT-PROJECTION
2. the DIRECTION-TO-DOWNTHROWN-BLOCK of :Y + the AZIMUTH of :Z
SHADING: DIAGONAL -> BREAK = BREAK, VERTICAL -> BREAK = T, UNSHADED -> BREAK = NIL
MODULAR INCREMENTS TO STROBE

- **Impulse**
  - Strobe-based KB editor
  - Customizable

- **Rule Interpreter**
  - Rule Debugging Package
  - Rule Editor (Impulse-based)

- **Constraint Package**
  - Semantic Integrity Management within and among objects

- **Object File Management (KBMS)**
  - Indexed non-resident object files

- **Declarative Task Representation**
  - Control and data flow among modules
  - User-interface information
KBS Architectural Principles

(Davis, 1982)

Separate the Inference Engine and Knowledge Base
  • What Is True vs How to Use It

Use as Uniform a Representation as Possible
  • Specialization Is Often Worth the Cost of Translation

Keep the Inference Engine Simple

Exploit Redundancy
KBS Architectural Principles

A Clear, Expressive Domain Model is Central
• Objects
• Abstract Relations

Partition Knowledge Wherever Possible
• Domain-Independent Knowledge
• Domain-Specific Knowledge
• Task-Specific Knowledge

Avoid Assumptions about Context of Use

Represent Control Explicitly
• Strategy Knowledge
• Problem-Solving State

Orchestrated Multiple Representations
• Objects, Rules, Procedures
• Invocation via Message

Consider User Interaction as an Integrated Component
Encoding Abstract Relations

• Generalization
  Specialization
  Subsumption

• Class Membership

• SuperPart    Task    Abstraction
  Part        Subtask    Expansion
  Port        Port ...

• Cause
  Effect
  Manifestation

• Suggests

• Precondition
  Postcondition

• Ordering ...

• Attribute (obligatory, optional, ...)
  (necessary, sufficient, ...)
Control Rule

Create tasks to determine any unknown attributes of the focus of the current task (one task per attribute).

If

There exists an Attribute (:att) of the Focus (:f) of the Current Task
such that the :att of :f is unknown, and
such that there are Detectors (:dtr) of :att of :f

Then

Create a Sibling Task (:st) of the Current Task
To Detect :att of :f using :dtr
Append [Rules ReElaborate Apply] to the SuccessfulInvocationProcessors of :sbt
Abstraction Example
(Szolovits & Clancey, 1985)

If Mary has a fever,
then Mary has an infection.

Mary is a patient

If the patient has a fever,
then the patient has an infection.

Fever is a symptom. Infection is a disease. Fever is a symptom of infection.

If the patient has a symptom of a disease,
then the patient has the disease.

A symptom is a feature. A disease is a class.

If there is evidence for a feature of a class,
then there is evidence for the class.
Interpretation Rule

Recognize the existence of a **CrevasseSplay**.

**If**

There exists a **RockUnit** (:ru) in the **Current Context** such that the **DepositionalEnvironment** (:de) of :ru is a descendant of **CrevasseFan**, and  
There exists an element of the **Patterns** (:bp) of :ru such that :bp is an instance of **BluePattern**, and  
such that at least 0.8 of :bp is within :ru, and  
There does not exist a co-constituent of :ru (:ru2) such that the **DepositionalEnvironment** of :ru2 is a descendant of **ChannelLag**, and  
such that :ru2 is below :ru

**Then**

Alter :de of :ru to be **CrevasseSplay**
Interpretation Rule

Recognize the existence of a **CrevasseSplay**.

**If**

- There exists a **RockUnit** (:ru) in the **Current Context** such that the **DepositionalEnvironment** (:de) of :ru is a descendant of **CrevasseFan**, and
- There exists an element of the **SedimentaryFeatures** (:cb) of :ru such that :sf is an instance of **CurrentBedding**, and such that at least 0.8 of :cb is within :ru, and
- There does not exist a co-constituent of :ru (:ru2) such that the **DepositionalEnvironment** of :ru2 is a descendant of **ChannelLag**, and such that :ru2 is below :ru

**Then**

- Alter :de of :ru to be **CrevasseSplay**
NEOMYCIN DIAGNOSTIC STRATEGY

Consult

MakeDiagnosis  PrintResults

IdentifyProblem  CollectInformation

ForwardReason  GenerateQuestions

ClarifyFinding

ProcessFinding/Hypothesis

EstablishHypothesisSpace  ProcessHardData

Group&Differentiate  Explore&Refine  AskGeneralQuestions

PursueHypothesis

TestHypothesis  RefineHypothesis  RefineComplexHypothesis
Examples:

IF the hypothesis being focused upon has a child that has not been pursued
THEN pursue that child

IF there is a datum that can be requested that is a characterizing feature of the recent finding that is currently being considered
THEN find out about the datum

IF the desired finding is a subtype of a class of findings, and the class of findings is not present in this case
THEN conclude that the desired finding is not present
Control Rule

Create tasks to elaborate the attribute values of the focus of the current task (one task per attribute value).

If

There exists a local Attribute (:att) of the Focus (:f) of the Current Task such that :att is known and elaborable

Then

Create an Offspring Task (:ost) of the Current Task To Elaborate :att using [Rules ReElaborate Apply] Append [Rules DeActivate Apply] to the SuccessfulInvocationProcessors of :ost
Control Rule

Create a task to refine the focus of the current task.

If

There does not exist a Sibling Task (:tsk) of the Current Task such that the Task of :tsk = Refine, and
There exists a Focus (:f) of the Current Task such that the Refiners (:r) of :f are known

Then

Create a Sibling Task (:sbt) of the Current Task To Refine :f using :r
Append [Rules ReEstablishRefine Apply] to the SuccessfulInvocationProcessors of :sbt
Interpretation Rule

Establish the existence of any *EnergyIndications*.

If

There exists a RockUnit (:ru) in the **Current Context**
such that the *Energyindications* of :ru are unknown,
such that the Well (:wl) of :ru is known,
such that the LogData (:Id) of :wl is known,
such that Message to
[Detectors, EnergyIndicators, Detect] (:en)
    is successful

Then
Assign *EnergyIndications* of :ru to be :en
Advantages:

- Transparency
- Explainability
- Maintainability and Extensibility
  - Simplified Addition of New Facts and Relations
  - Increased Robustness
- Storage
- Reusability of KBs in Related Domains
  - Lower Cost of Construction
- Freedom to Select Optimal Representation For Special Cases