Knowledge-Intensive Development Environments

Reid G. Smith

KNOWLEDGE ACQUISITION IS THE OMNIPRESENT PROBLEM[1]

The problem is moving real-world knowledge into a software system—by whatever means—*and making it work.* It extends over the complete lifetime of a system—during initial design, continuing extension of the knowledge base, integration with other systems, and application to new problems.

We have two main ways to impact the knowledge acquisition bottleneck:

➪ Knowledge-Intensive Development Environments

Learning Apprentice Systems

Adherence to a set of architectural principles of knowledge-based system design simplifies the task of knowledge acquisition and reuse.

[1] **Slides from:** Reid. G. Smith. Knowledge-Intensive Development Environments. Presented at the *Fifth National Conference on Artificial Intelligence (AAAI-86) Panel on Directions for Expert Systems,* August 14, 1986, Philadelphia, PA. (Panelists: Janice S. Aikins (Chair), Frederick Hayes-Roth, John P. McDermott, Herbert Schorr, Reid G. Smith).

KNOWLEDGE-INTENSIVE DEVELOPMENT ENVIRONMENTS

Representation Substrate **(*e.g.,* object-oriented)**

**Integration of Objects, Procedures, Rules,
Constraints, Dependencies,
Contexts, Explanation, …**

**Knowledge of Use of Components,
Problem-Solving Methods,
Generic Domains, …**

Interaction Substrate

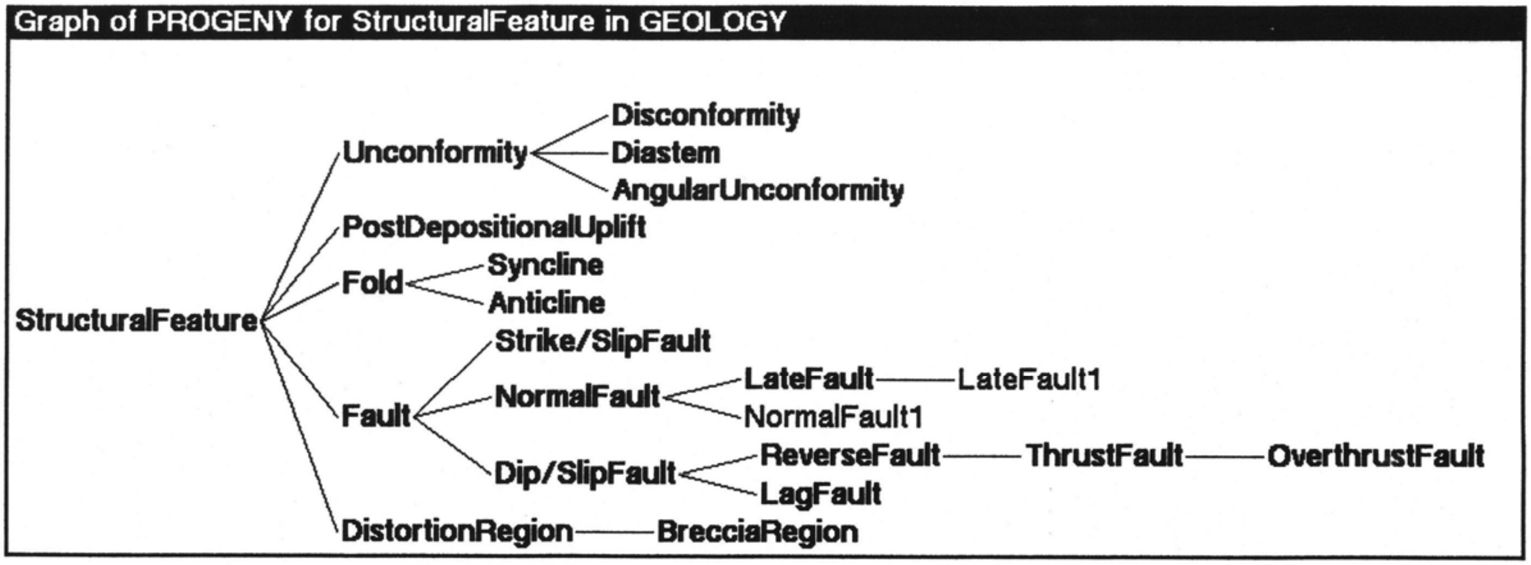
Clients: Developer/Maintainer
Domain Specialist
End User

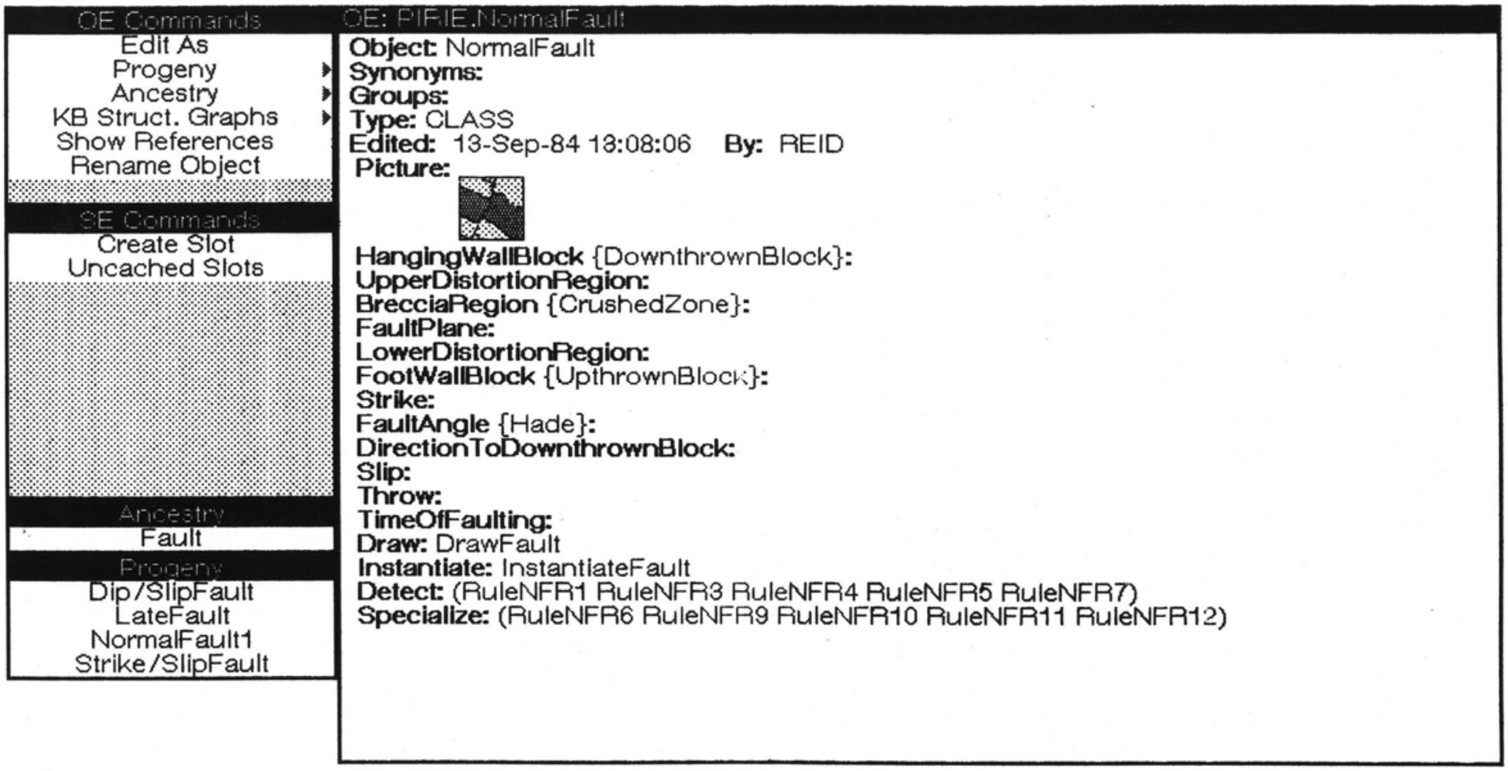
The needs of all client types can be met with a single extensible substrate

User Interfaces ⇨ Knowledge-Based Systems

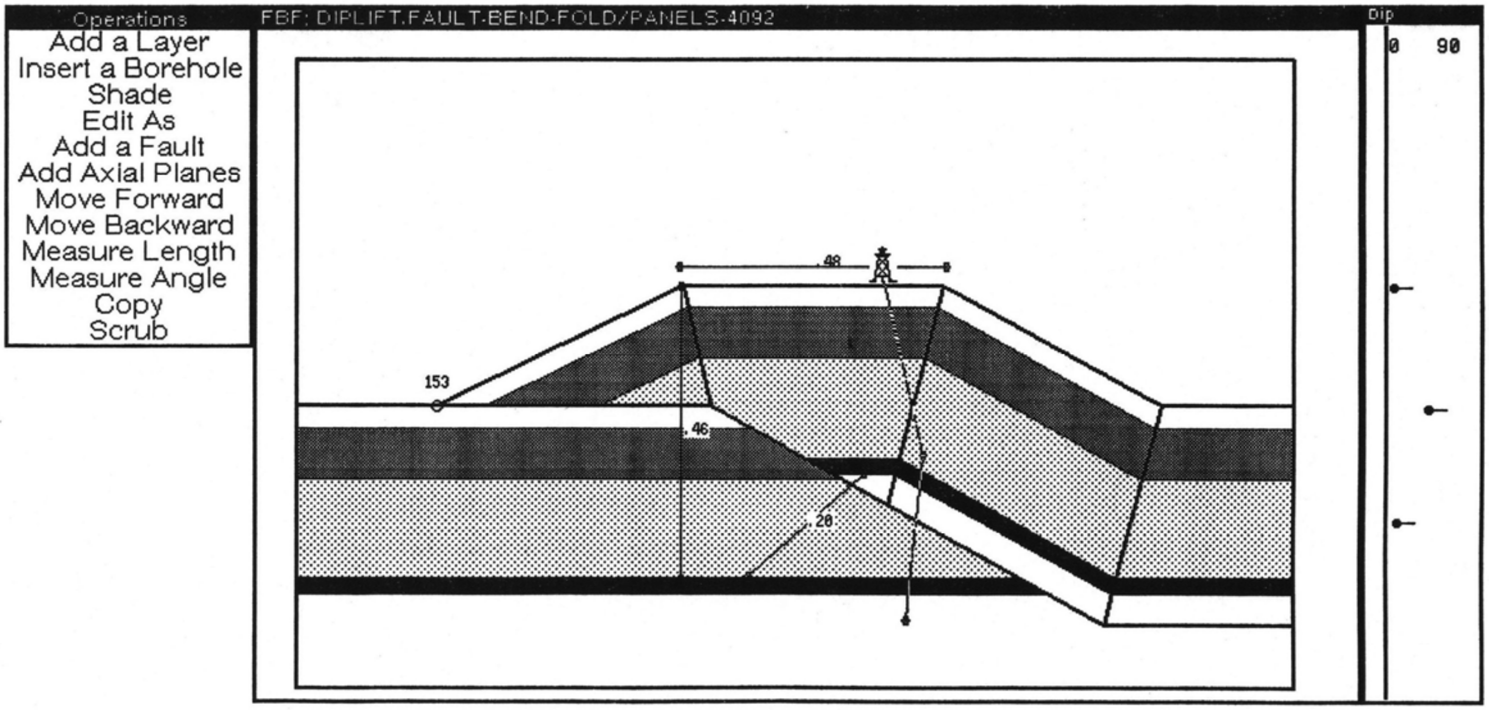
During the incremental refinement process that typifies KBS development, high quality user interfaces are essential.

- **Expression and Interaction in Domain Terms**
- **Direct Interaction by Domain Specialist**
- **Focus of Knowledge Engineer and Domain Specialist on Domain Knowledge and Problem-Solving Methodology**
- **Explanation and Debugging**
- **Interactive Graphics**


Graph of PROGENY for StructuralFeature in GEOLOGY
StructuralFeature
Unconformity
Disconformity
Diastem
AngularUnconformity
PostDepositionalUplift
Fold
Syncline
Anticline
Fault
Strike/SlipFault
NormalFault
LateFault
LateFault1
NormalFault1
Dip/SlipFault
ReverseFault
ThrustFault
OverthrustFault
LagFault
DistortionRegion
BrecciaRegion



```
Object: NormalFault
Synonyms:
Groups:
Type: CLASS
Edited: 13-Sep-84 13:08:06    By: REID
Picture:
HangingWallBlock {DownthrownBlock}:
UpperDistortionRegion:
BrecciaRegion {CrushedZone}:
FaultPlane:
LowerDistortionRegion:
FootWallBlock {UpthrownBlock}:
Strike:
FaultAngle {Hade}:
DirectionToDownthrownBlock:
Slip:
Throw:
TimeOfFaulting:
Draw: DrawFault
Instantiate: InstantiateFault
Detect: (RuleNFR1 RuleNFR3 RuleNFR4 RuleNFR5 RuleNFR7)
Specialize: (RuleNFR6 RuleNFR9 RuleNFR10 RuleNFR11 RuleNFR12)
```


Operations
Add a Layer
Insert a Borehole
Shade
Edit As
Add a Fault
Add Axial Planes
Move Forward
Move Backward
Measure Length
Measure Angle
Copy
Scrub
FBF: DIPLIFT.FAULT-BEND-FOLD/PANELS-4092
Dip
0 90
48
153
46
20

Knowledge-Based Systems ⇨ User Interfaces

More than 50% of KBS code may support the user interface. If a KBS toolkit serves only to build the representation & inference parts of an application, a sizeable problem remains for developers.

The user interface is often the critical module. It is what people see—end users and domain specialists alike. It provides the data from which users form mental models of how the overall system operates, and hypotheses about its behavior in new situations.

The representation substrate already contains tools well-suited to user interface design.

- **Object-Oriented Encoding of Interface Constructs**
- **Interpretation of Knowledge Base to Specialize Views and Interaction Methods**
- **Constraints to Maintain Consistency**
- **Rules to Infer Missing or Dependent Information**

Slogan

The Form IS The Content

The Form IS The Content
∧
a very important part of

RGS /August 1986

The following slides were not used.

KNOWLEDGE BASED SYSTEM
ARCHITECTURAL PRINCIPLES

Separate the Inference Engine and Knowledge Base

Keep the Inference Engine Simple/Understandable

Orchestrate Multiple Representations

Design around a Clear, Expressive Domain Model

Partition Knowledge Wherever Possible

Represent Problem Solving/Control Explicitly

Avoid Assumptions about Context of Use

Design for Explanation

Consider User Interaction as an Integrated Component

Exploit Redundancy

KNOWLEDGE BASED SYSTEM
ARCHITECTURAL PRINCIPLES

Separate the Inference Engine and Knowledge Base

Keep the Inference Engine Simple

Orchestrate Multiple Representations

- **Objects, Rules, Constraints, Procedures … *Messages***
- **Uniformity Simplifies Task of Inference Engine Design**

Design around a Clear, Expressive Domain Model

- ***Static* Concept Knowledge … Abstract Relationships**
- ***Dynamic* Action Knowledge (Tasks)**
  … Linked to Static Knowledge
- **Structure Readily Understandable by Domain Specialists**

Partition Knowledge Wherever Possible

- **By Domain [Geology, Problem Solving/Control, Interaction]**
- **By Task**

Represent Problem Solving/Control Explicitly

- **Strategy Knowledge & Problem-Solving State**

Avoid Assumptions about Context of Use

- **Problems Change Over Time**
- **Knowledge Is Applied in New Contexts & Different Systems**

Design for Explanation

… For End Users, Domain Specialists, and Programs

Consider User Interaction as an Integrated Component

Exploit Redundancy

RGS /August 1986