Form and Content In Knowledge-Intensive Development Environments

Reid G. Smith

Schlumberger Palo Alto Research
3340 Hillview Ave.
Palo Alto, CA 94304
USA

Eric Schoen

Stanford University
Stanford, CA 94305
USA

Future Knowledge Engineering Environments Panel
IEEE CompCon, February 1987

The central problem in knowledge-based system construction is knowledge acquisition—moving real-world knowledge into a software system—by whatever means—and *making it work*. This is not simply a problem encountered in the initial stages of design—although, even if it were, it would be a serious enough problem. Rather, it is an omnipresent prob lem that extends over the complete lifetime of a KBS—during initial design, continuing extension of the knowledge base, integration with other systems, and application to new problems.

One of the ways by which we are attempting to impact the knowledge acquisition bottleneck is through construction of *knowledge-intensive development* environments—highly interactive environments *that* allow *direct* interaction by both KBS developers and domain specialists.

A knowledge-intensive development environment *has* two essential, interrelated components; *a representation and reasoning substrate,* and an *interaction substrate. The rep*resentation and reasoning substrate, *which* we assume to be object-oriented, integrates distinct problem-solving subsystems involving objects, rules, constraints, contexts, and explanation. *This* substrate *has* knowledge of *the use* of *the* individual subsystems, problem-solving methods, and basic knowledge of the domains in which it is applied.

It has been standard to concentrate on representation and reasoning in discussions of the future of development environments—every current KBS shell has at least a rudimentary representation and reasoning substrate. In a sense, this is the *content* part of a development environment. There is, however, another very important part—the *form* part—which we call the interaction substrate.

The interaction substrate provides a set of tools for constructing interactive user interfaces to knowledge-based systems. It must support three different perspectives, corresponding to three different types of user: *(i)* the developer/maintainer; *(ii)* the domain specialist; and *(iii)* the end user. The substrate must provide a reactive environment for developer/maintainers. It must allow domain specialists to focus on the encoded domain knowledge, hiding the underlying representational mechanisms, and provide direct expression and interaction in the *natural* terms of the domain. Finally, the substrate must permit the construction of transparent and "easy-to-use" interfaces for end users. Our recent work on Impulse-86 [3] indicates that it is possible to construct an interaction substrate that meets the needs of all three user groups.

We have concentrated on the interaction substrate over the past year for two main rea sons. First, a powerful interaction substrate simplifies knowledge acquisition. The domain specialist may interact directly with the evolving system in order to examine the knowledge base for gaps, weaknesses, and misunderstandings; and expand, refine, and correct the encoded knowledge. This direct interaction is further simplified when the specialist can articulate his approaches to solving problems via familiar modes of expression (*e.g.*, pictures or equations), in the natural terms and notation of his domain. When direct interaction is possible, the bandwidth of the communication between domain specialist and knowledge engineer is increased. They can concentrate on the required domain knowledge and problem-solving methodology—as opposed to the underlying programming mechanics. Interactive graphics—supported by the interaction substrate—plays a large part both in articulation of methodology and in explanation of system operation.

Second, it is by now well-understood that good interfaces account for a sizable percentage of the overall code in many systems. In addition, 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 [2]. Many of our systems appear to a user as sophisticated graphics terminals—albeit with a strong representation and reasoning component. In our domains of interest, the problems are such that a symbiotic man-machine interac tion is very attractive, and may be essential to success. Hence a KBS shell which supports only the representation and reasoning parts of an application—and provides no assistance for the interaction part—leaves a sizeable problem for application developers. Therefore, we feel compelled to provide a shell that supports interaction as well as problem-solving.

Serendipitously, the representation and reasoning substrate already contains tools well-suited to user interface design. The object-oriented paradigm so useful in domain knowledge base construction is equally viable for encoding and organizing interface constructs like editors, windows, menus, and views. Furthermore, reasoning mechanisms already in place can be brought to bear on the management of user interaction. For example, constraint systems which support problem solving can be used to help maintain consistency during user interaction. Rules can be used to infer missing or dependent information. The same browsing techniques used by the developer to explore the system code can support graphical explanation—for developers, domain specialists, and end users alike.

In retrospect, this comes as no surprise. We can view human-computer interaction—especially in the context of knowledge-based systems—as a form of *discourse*, one of the most knowledge-intensive activities in which we engage [1]. For effective communication, each participant must be aware of the preconceptions, intentions, and specialized vocabulary of the other. Further, much is often left unstated during discourse, to be inferred by the participants. Analogies to discourse can be drawn in a knowledge-intensive development environment. The interaction substrate must reason about the background, sophistication, and intentions of its user, as well as the available domain knowledge, to construct specialized views. It also must often infer information left implicit by the user. In fact, the interaction substrate is itself a knowledge-based system; thus, it is only natural that its needs are well matched to the services offered by the representation and reasoning substrate.

The bottom line, then, for future development environments is that attention should be paid to *both* form and content.

Acknowledgements: These ideas have been developed through many interactions with members of the Knowledge-Based Systems group at Schlumberger-Doll Research, especially Robert L. Young.

References

[1] Michael Brady and Robert C. Berwick, editors. *Computational Models of Discourse.* MIT Press, Cambridge, MA, 1983.

[2] J. Seely Brown. *From Cognitive to Social Ergonomics and Beyond*, pages 457-486. Lawrence Erlbaum Associates, Hillsdale, N. J., 1986.

[3] R. G. Smith, R. Dinitz, and P. Barth. Impulse-86: A Substrate for Object-Oriented Interface Design. In *Proceedings of the First ACM Conference on Object Oriented Systems, Languages, and Applications*, pages 167-176, September 1986.

The form IS The Content

Slogan[1]

[1] Each panelist was asked to generate a comparatively polarizing and challenging **slogan** as the central theme of his presentation—to highlight differences in perspective and stimulate a vigorous discussion.

The form IS The Content

a very important part of

KNOWLEDGE ACQUISITION IS THE OMNIPRESENT PROBLEM

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.

Knowledge acquisition and explanation are two aspects of a dialogue between a user and a system.

User interfaces are of critical importance during knowledge-based system development and use.

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**

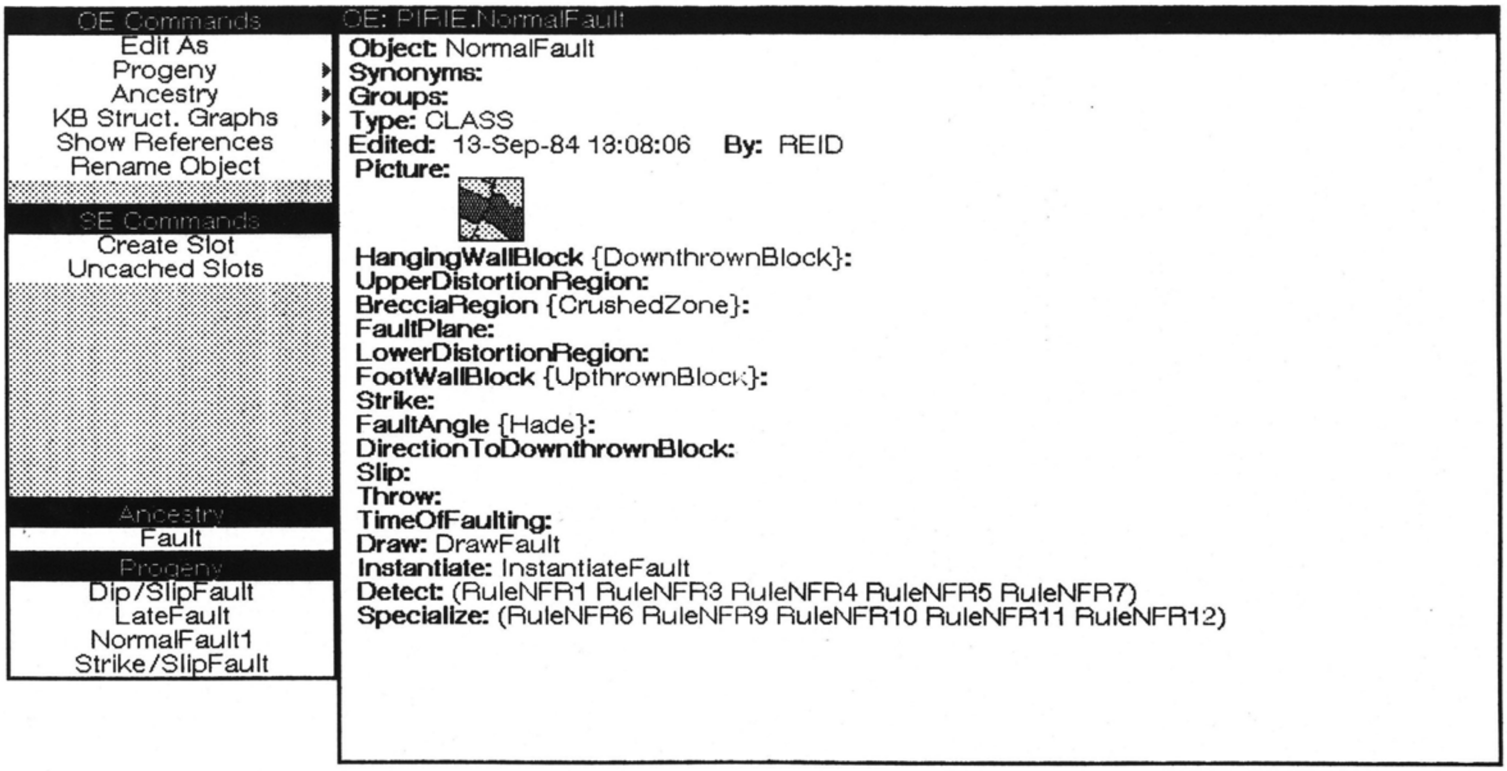
Knowledge-Based Systems ⇨ User Interfaces

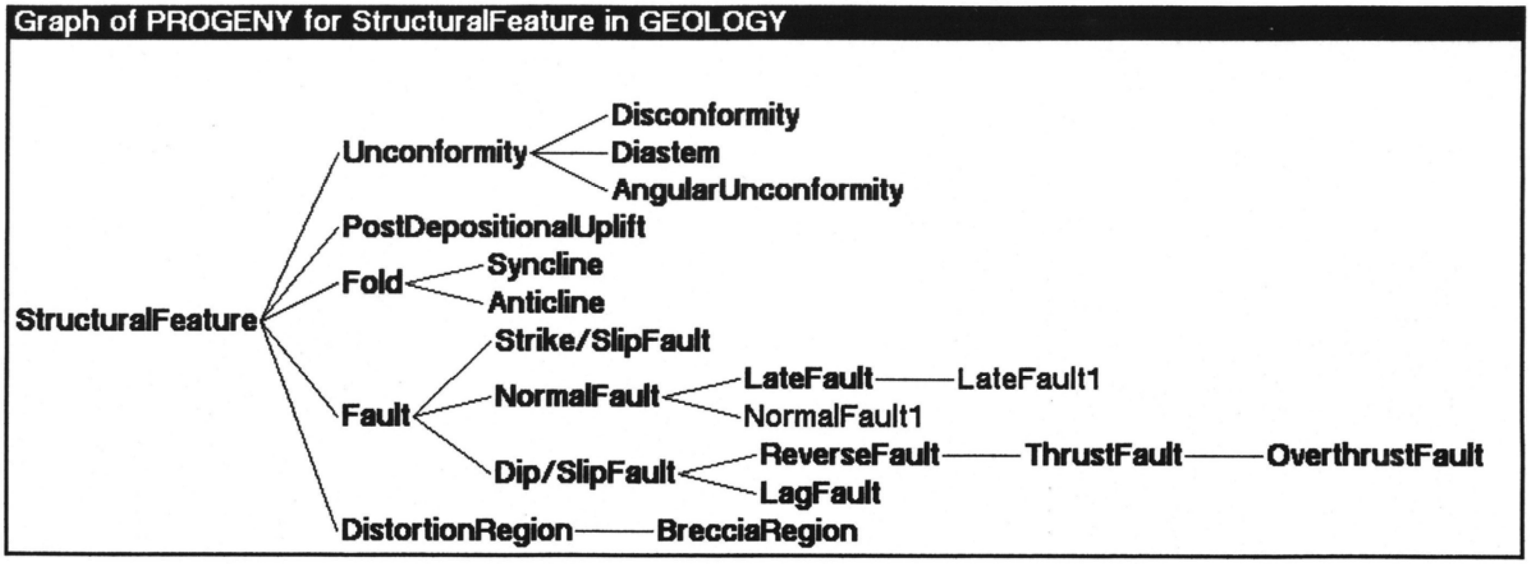
More than 50% of KBS code may support the user interface. If a KBS toolkit serves only to build the representation & inference parts of an application, a sizeable problem remains for developers.

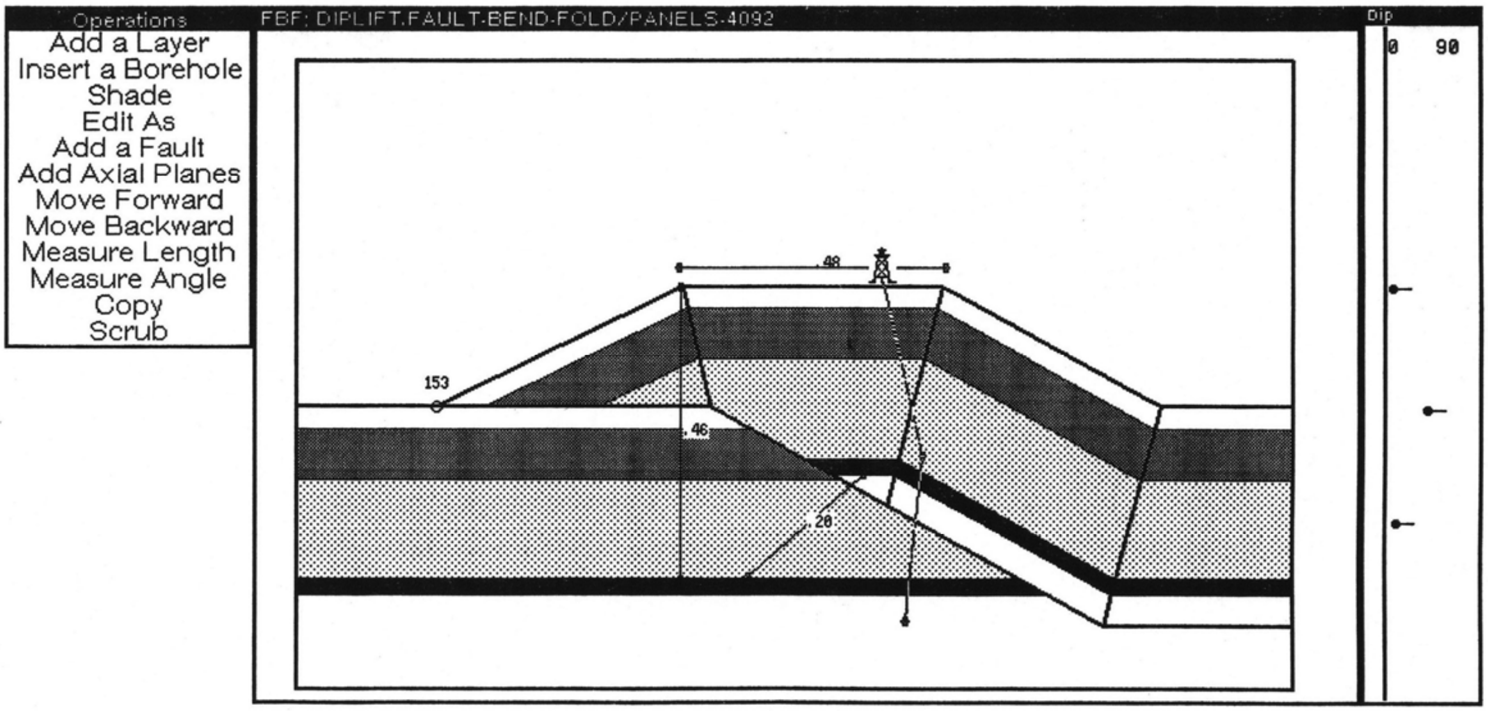
The user interface is often the critical module. It is what people see—end users and domain specialists alike. It provides the data from which users form mental models of how the overall system operates, and hypotheses about its behavior in new situations.

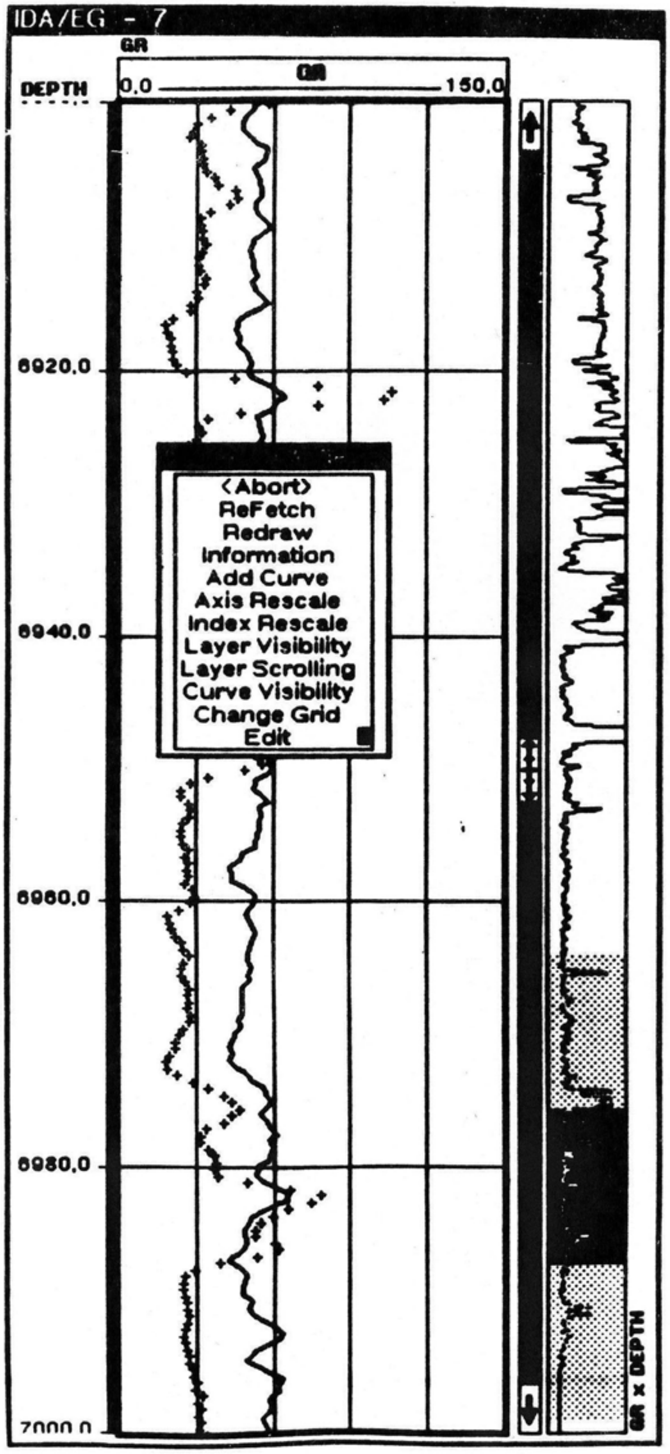
The representation substrate already contains tools well-suited to user interface design.

- **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**

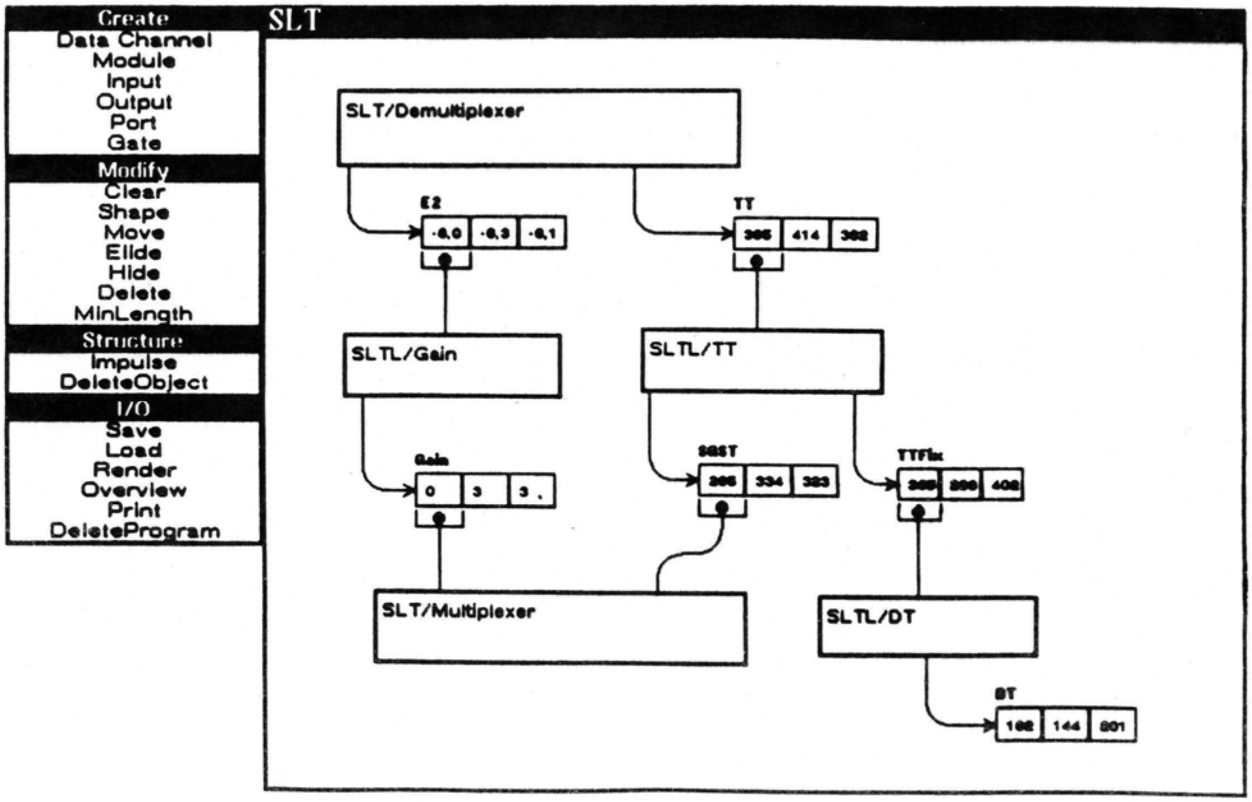

OE Commands
Edit As
Progeny
Ancestry
KB Struct. Graphs
Show References
Rename Object
SE Commands
Create Slot
Uncached Slots
Ancestry
Fault
Progeny
Dip/SlipFault
LateFault
NormalFault1
Strike/SlipFault
OE: PIRIE.NormalFault
Object: NormalFault
Synonyms:
Groups:
Type: CLASS
Edited: 13-Sep-84 13:08:06 By: 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)


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


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



IDA/EG - 7
GR
DEPTH
0.0
GR
150.0
6920.0
6940.0
6960.0
6980.0
7000.0
<Abort>
ReFetch
Redraw
Information
Add Curve
Axis Rescale
Index Rescale
Layer Visibility
Layer Scrolling
Curve Visibility
Change Grid
Edit
GR x DEPTH


Create
Data Channel
Module
Input
Output
Port
Gate
Modify
Clear
Shape
Move
Elide
Hide
Delete
MinLength
Structure
Impulse
DeleteObject
I/O
Save
Load
Render
Overview
Print
DeleteProgram
SLT
SLT/Demultiplexer
E2
TT
414
SLTL/Gain
SLTL/TT
0
3
3
334
TTFix
SLT/Multiplexer
SLTL/DT
144

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