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.

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.

---

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
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
The Form IS The Content

∧

a very important part of
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