IAI : Expert Systems

© John A. Bullinaria, 2005

What is an Expert System?

Jackson (1999) provides us with the following definition:

> An ***expert system*** is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice.

To solve expert-level problems, expert systems will need efficient access to a substantial domain ***knowledge base***, and a ***reasoning mechanism*** to apply the knowledge to the problems they are given. Usually they will also need to be able to explain, to the users who rely on them, how they have reached their decisions.

They will generally build upon the ideas of knowledge representation, production rules, search, and so on, that we have already covered.

Often we use an ***expert system shell*** which is an existing knowledge independent framework into which domain knowledge can be inserted to produce a working expert system. We can thus avoid having to program each new system from scratch.

Typical Tasks for Expert Systems

There are no fundamental limits on what problem domains an expert system can be built to deal with. Some typical existing expert system tasks include:

1. The interpretation of data

   Such as sonar data or geophysical measurements
2. Diagnosis of malfunctions

   Such as equipment faults or human diseases
3. Structural analysis or configuration of complex objects

   Such as chemical compounds or computer systems
4. Planning sequences of actions

   Such as might be performed by robots
5. Predicting the future

   Such as weather, share prices, exchange rates

However, these days, “conventional” computer systems can also do some of these things.

Characteristics of Expert Systems

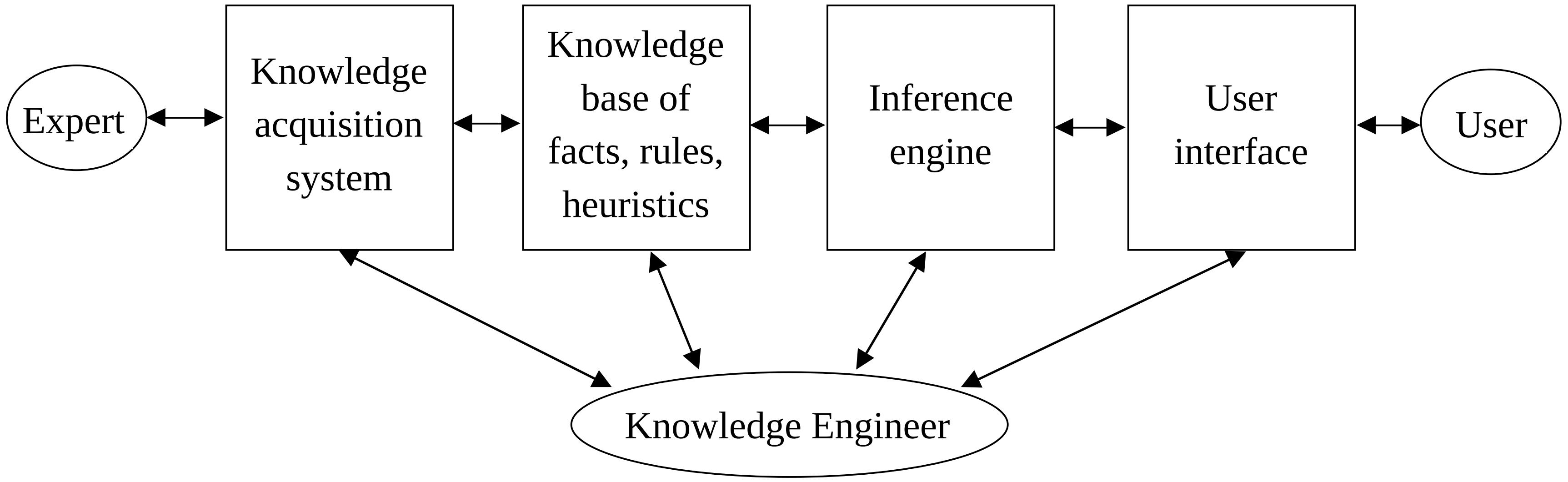
Expert systems can be distinguished from conventional computer systems in that:

1. They ***simulate human reasoning*** about the problem domain, rather than simulating the domain itself.
2. They perform ***reasoning*** over ***representations of human knowledge***, in addition to doing numerical calculations or data retrieval. They have corresponding distinct modules referred to as the ***inference engine*** and the ***knowledge base***.
3. Problems tend to be solved using ***heuristics*** (rules of thumb) or ***approximate methods*** or ***probabilistic methods*** which, unlike algorithmic solutions, are not guaranteed to result in a correct or optimal solution.
4. They usually have to provide ***explanations*** and ***justifications*** of their solutions or recommendations in order to convince the user that their reasoning is correct.

Note that the term Intelligent Knowledge Based System (IKBS) is sometimes used as a synonym for Expert System.

The Architecture of Expert Systems

The process of building expert systems is often called ***knowledge engineering***. The ***knowledge engineer*** is involved with all components of an expert system:


Expert
Knowledge acquisition system
Knowledge base of facts, rules, heuristics
Inference engine
User interface
User
Knowledge Engineer

Building expert systems is generally an iterative process. The components and their interaction will be refined over the course of numerous meetings of the knowledge engineer with the experts and users. We shall look in turn at the various components.

Knowledge Acquisition

The knowledge acquisition component allows the expert to enter their knowledge or expertise into the expert system, and to refine it later as and when required.

Historically, the knowledge engineer played a major role in this process, but automated systems that allow the expert to interact directly with the system are becoming increasingly common.

The knowledge acquisition process is usually comprised of three principal stages:

1. ***Knowledge elicitation*** is the interaction between the expert and the knowledge engineer/program to elicit the expert knowledge in some systematic way.
2. The knowledge thus obtained is usually stored in some form of human friendly ***intermediate representation***.
3. The intermediate representation of the knowledge is then compiled into an ***executable form*** (e.g. production rules) that the inference engine can process.

In practice, many iterations through these three stages are usually required!

Knowledge Elicitation

The knowledge elicitation process itself usually consists of several stages:

1. Find as much as possible about the problem and domain from books, manuals, etc. In particular, become familiar with any specialist terminology and jargon.

2. Try to characterise the types of reasoning and problem solving tasks that the system will be required to perform.

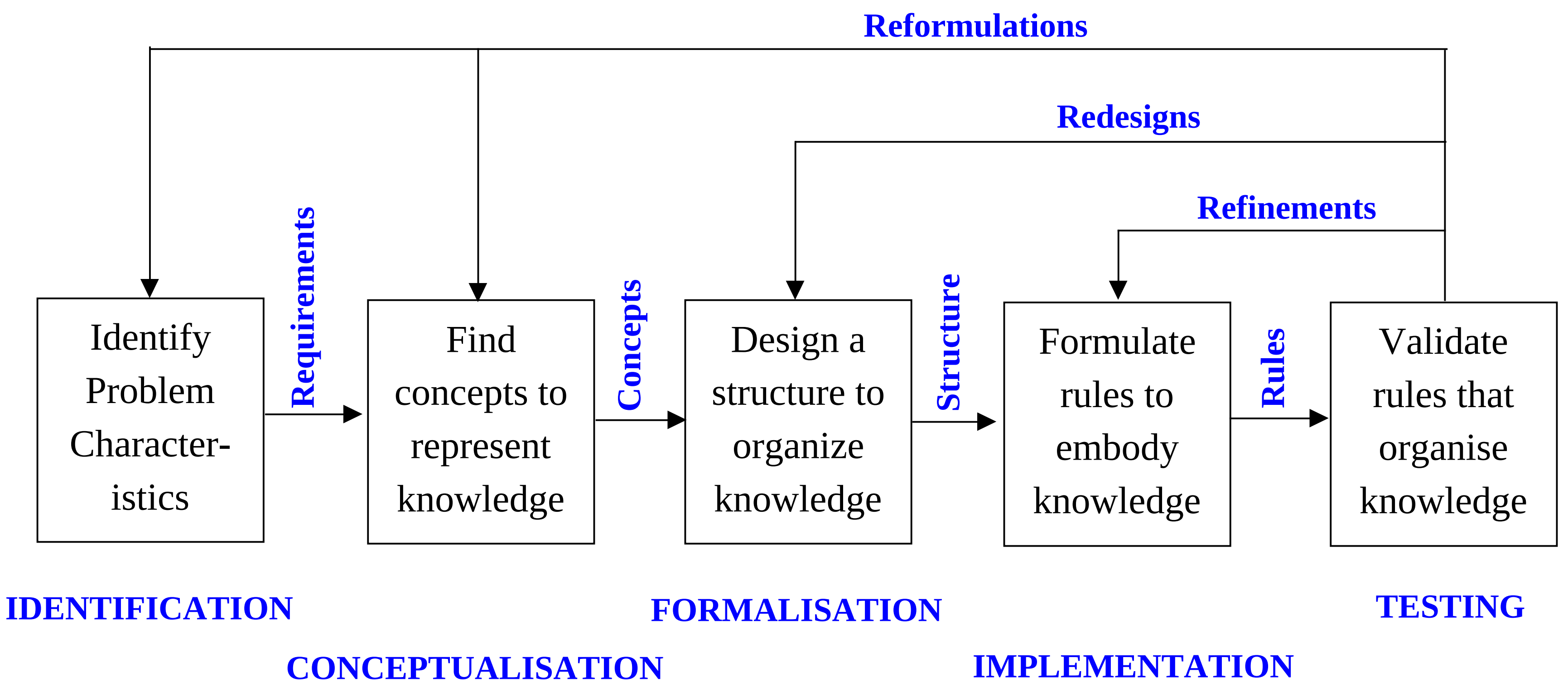
3. Find an expert (or set of experts) that is willing to collaborate on the project. Sometimes experts are frightened of being replaced by a computer system!

4. Interview the expert (usually many times during the course of building the system). Find out how they solve the problems your system will be expected to solve. Have them check and refine your intermediate knowledge representation.

This is a time intensive process, and ***automated knowledge elicitation*** and ***machine learning*** techniques are increasingly common modern alternatives.

Stages of Knowledge Acquisition

The iterative nature of the knowledge acquisition process can be represented in the following diagram (from Jackson, Section 10.1):


Reformulations
Redesigns
Refinements
Identify Problem Character-istics
Requirements
Find concepts to represent knowledge
Concepts
Design a structure to organize knowledge
Structure
Formulate rules to embody knowledge
Rules
Validate rules that organise knowledge
IDENTIFICATION
CONCEPTUALISATION
FORMALISATION
IMPLEMENTATION
TESTING

Note: Virtually every book/paper/lecturer will represent this in a slightly different way!

Levels of Knowledge Analysis

Knowledge identification: Use in depth interviews in which the knowledge engineer encourages the expert to talk about how they do what they do. The knowledge engineer should understand the domain well enough to know which objects and facts need talking about.

Knowledge conceptualization: Find the primitive concepts and conceptual relations of the problem domain.

Epistemological analysis: Uncover the structural properties of the conceptual knowledge, such as taxonomic relations (classifications).

Logical analysis: Decide how to perform reasoning in the problem domain. This kind of knowledge can be particularly hard to acquire.

Implementational analysis: Work out systematic procedures for implementing and testing the system.

Capturing Tacit/Implicit Knowledge

One problem that knowledge engineers often encounter is that the human experts use tacit/implicit knowledge (e.g. procedural knowledge) that is difficult to capture.

There are several useful techniques for acquiring this knowledge:

1. **Protocol analysis:** Tape-record the expert thinking aloud while performing their role and later analyse this. Break down the their protocol/account into the smallest atomic units of thought, and let these become operators.

2. **Participant observation**: The knowledge engineer acquires tacit knowledge through practical domain experience with the expert.

3. **Machine induction**: This is useful when the experts are able to supply examples of the results of their decision making, even if they are unable to articulate the underlying knowledge or reasoning process.

Which is/are best to use will generally depend on the problem domain and the expert.

Representing the Knowledge

We have already looked at various types of knowledge representation. In general, the knowledge acquired from our expert will be formulated in two ways:

1. **Intermediate representation** – a structured knowledge representation that the knowledge engineer and expert can both work with efficiently.
2. **Production system** – a formulation that the expert system's inference engine can process efficiently.

It is important to distinguish between:

1. **Domain knowledge** – the expert's knowledge which might be expressed in the form of rules, general/default values, and so on.
2. **Case knowledge** – specific facts/knowledge about particular cases, including any derived knowledge about the particular cases.

The system will have the domain knowledge built in, and will have to integrate this with the different case knowledge that will become available each time the system is used.

The Inference Engine

We have already looked at production systems, and how they can be used to generate new information and solve problems.

Recall the steps in the basic Recognize Act Cycle:

1. ***Match*** the premise patterns of the rules against elements in the working memory. Generally the rules will be domain knowledge built into the system, and the working memory will contain the case based facts entered into the system, plus any new facts that have been derived from them.
2. If there is more than one rule that can be applied, use a ***conflict resolution*** strategy to choose one to apply. Stop if no further rules are applicable.
3. ***Activate*** the chosen rule, which generally means adding/deleting an item to/from working memory. Stop if a terminating condition is reached, or return to step 1.

Early production systems spent over 90% of their time doing pattern matching, but there is now a solution to this efficiency problem:

The Rete-Algorithm

The naïve approach to the recognize act cycle tries to match all E elements in working memory against all P premises in all R rules, so it is necessary to check all $E \times P \times R$ possible matches in each cycle.

However, the rules will generally have parts of their conditions in common (***structural similarity***), and the application of any one rule will generally only slightly change the working memory (***temporal redundancy***).

These facts are used by the ***Rete Algorithm*** to improve efficiency ('rete' is Latin for 'net'). The condition parts of the rules are structured into a network. For the first cycle, the match algorithm generates the initial conflict set by processing the network for all the initial facts. Thereafter, only the changed elements in working memory need to be fed through the network to determine the changes to the conflict set.

For an explicit example see Russell & Norvig (1995) Section 10.5:

The User Interface

The Expert System user interface usually comprises of two basic components:

1. The Interviewer Component

This controls the dialog with the user and/or allows any measured data to be read into the system. For example, it might ask the user a series of questions, or it might read a file containing a series of test results.

2. The Explanation Component

This gives the system's solution, and also makes the system's operation transparent by providing the user with information about its reasoning process. For example, it might output the conclusion, and also the sequence of rules that was used to come to that conclusion. It might instead explain why it could not reach a conclusion.

So that is how we go about building expert systems. In the next two weeks we shall see how they can handle uncertainty and be improved by incorporating machine learning.

Overview and Reading

1. We began by considering what exactly Expert Systems are, their general architectures, and some of the typical tasks that they can deal with.
2. We then looked in turn at each of the principal Expert System components (the knowledge acquisition system, the knowledge base, the inference engine, and the user interface) and studied the role of the knowledge engineer in the knowledge elicitation process and in building the expert system.

Reading

1. Jackson: Chapters 1, 10
2. Negnevitsky: Section 1.2.4, Chapter 2
3. Rich & Knight: Chapter 20
4. Russell & Norvig: Sections 8.4, 16.7
5. Nilsson: Chapter 17