

KBS/ES Design and Implementation

So far, though we have considered a mature AI architecture and its application, we have not addressed in detail the issue of **how** we construct such systems.

Needless to say, this is quite important...!!!

This week, we will have a look at the following issues:

- Knowledge acquisition and techniques.
- Knowledge Formulation
- KBS Implementation and ESTA
- Verification and Validation.

The KA Bottleneck

It (usually) turns out that the most difficult issue we have to tackle is that of ***Knowledge Acquisition*** (KA) - eliciting the knowledge required by the KBS in a structured manner.

This issue is often termed the ***Knowledge Acquisition Bottleneck***.

An overview of some KA methods is given here - this is not exhaustive (more detail in Requirements Engineering).

Turban "Expert Systems and Applied AI", and Hart "Knowledge Acquisition" are good further reading (useful for projects?)..

Difficulties in KA

Knowledge acquisition is difficult for the following reasons:

- ***Knowledge expression*** - introspection and tacit knowledge.
- ***Transfer to a machine*** - AI systems require a finer-grained KR than humans usually provide.
- ***Number of participants*** - often a team of people (e.g. multiple experts) are involved which complicates matters.
- ***Structuring the knowledge*** - we have to provide a structure to what the expert gives us.
- ***Experts unwilling to cooperate*** - politics, FUD!
- ***Communication issues*** - interpersonal chemistry!
- ***Irrelevant data*** - finding a needle in a haystack!

Types of KA Method

KA methods can be categorised as follows:

- **Manual Methods**, e.g.
 - **Interviews** - structured/unstructured
 - **Tracking Methods** - e.g. protocol analysis
- **Semi-automatic Methods**
 - Supporting the expert.
 - Supporting the knowledge engineer.
- **Automatic Methods**
 - **Machine Learning/Data Mining** - e.g. rule induction.

Interviews

These are the most commonly used from of KA and involve a dialogue between expert and knowledge engineer (KE).

There are a number of types, e.g. ***unstructured interviews***. The expert is allowed to lead the discussions ('tell me everything you know!') with the expert interrupting for clarifications, etc.

- Seldom gives the whole picture, but can be useful for a starting point to formulate questions for later interviews.
- Often difficult to interpret what is being said.
- Can make the KE seem unprepared!
- Problem of separating out the relevant information.

Interviews (Continued)

It is therefore often better to perform **structured interviews**, which allows a more systematic and goal-orientated process. There are a number of ways of doing this:

- The expert to talking through a case study (**walkthrough**).
- Having a list of specific questions to ask the expert - some KBS methodologies make use of prepared sets of questions (the **probing questions** method).
- Making the goals of the session explicit to the expert.
- The KE actively directs the interview (**directional control**).

In summary, interviews can be useful, but they need planning and their results must be carefully checked.

Tracking Methods

These attempt to *track* the reasoning process of an expert, accessing their ‘train of thought’ and their cognitive tasks.

Protocol analysis records the expert’s step-by-step information processing/decision making behaviour. The expert is asked to ‘think aloud’, and a transcript is analysed.

- This makes the expert be explicit about what he is doing, and allows the KE access to this.
- *Problems include*: the need for the expert to be aware of what he is doing and why, be able to articulate this, be aware of the alternatives, subjectivity, and that the expert may be mistaken!

Other Manual Methods

- **Case analysis** - expert critiques case studies, a more selective version of this is **critical incident analysis**.
- **Brainstorming** - useful for multiple experts.
- **Conceptual graphs** - this allows the domain to be visualised.
- **Prototyping** - allowing the expert to critique a prototype can yield a lot of information.
- **Clustering/Grouping methods** - good for seeing how domain objects and concepts are related to each other.
- **Simulation** - try to emulate the expert!

There are plenty more - the above should suffice....

Expert Driven Methods

You could try to get the expert to do all of the work by an open-ended questionnaire, organised report, activity logs, or giving a presentation. This is good for getting overviews, however:

- Experts are not trained as KEs - so they won't be as good!
- Expert reports tend to be biased.
- Experts may speculate, rather than sticking to the facts.
- Experts lose interest (time constraints!)
- Experts may forget to tell you things!

If the expert is hard to get hold of, these can be useful.

Repertory Grid Analysis

This has the following stages.

- Identify the important **objects** in the domain of expertise
- Identify the important **attributes** for decision making.
- Establish a bi-polar scale for each attribute.
- The expert then looks at 3 objects and asks what distinguishes any two of them from the third.
- This is then used to assign ratings to elements in an object-attribute grid (see next slide).

The expert can then revise his ratings, and the grid can be used to make suggestions - ***this can easily be made computer aided!***

Repertory Grid Example

<u>Attribute</u>	<u>Orientation</u>	<u>Ease of Use</u>
<u>Trait</u>	<u>Symbolic (3)</u>	<u>High (3)</u>
<u>Opposite</u>	<u>Numeric (1)</u>	<u>Low (1)</u>
LISP	3	2
PROLOG	3	2
C	2	2
COBOL	2	3
FORTRAN	1	1

- In this case two **attributes** were considered, *orientation* and *ease of use*, with bipolar traits.
- **Values** were then assigned to each **object** (programming language) to produce the grid.
- This can be used to answer questions, e.g. if numeric orientation is very important, then use FORTRAN.

KA Tools

Several types of tools have been developed for supporting KA, which fall under the following types:

- ***Editors and interfaces*** - minimise errors, maximise usability.
- ***Explanation*** - gives trace of reasoning for debugging (also useful for the user).
- ***KB revision aids*** - check for mistakes both in syntax and semantics (e.g. inconsistency)
- ***Pictorial KA (PIKA)*** - assists in visualisation.

E.g., the TEIRESIAS system for EMYCIN, which identified inconsistencies, rule conflicts, and other inadequacies.

Rule Induction

Now often known under its 'sexier' moniker - ***data mining***.

If a large enough database of cases is available, there are methods (e.g. ID3, C4.5) for extracting rules (or decision trees) for this data. This could save a *lot* of effort!

However....

- Rules generated may be hard to interpret.
- Expert still has to select the attributes.
- There are a range of algorithms, with different capabilities.
- Only good for rule/classification-based applications.
- Combinatorial explosion - limits number of attributes.
- Garbage in, garbage out...

Selecting KA Tools/Methods

Some (ideal world) criteria are as follows:

- Applicability to a wide range of problem domains
- Easy for experts to use (e.g. tutorial facilities)
- Able to analyse work in progress and detect inconsistencies and gaps in knowledge.
- Can handle multiple sources of knowledge (e.g. experts, textual records, rule induction).
- Automates manual methods where possible.
- Can interface with a number of ES shells.

Also take into account the (dis)advantages discussed earlier.

Knowledge Formulation

- This is the intermediate stage between KA and KR. Once knowledge is obtained it needs to be structured, before a KR formalism is selected and applied.
- One important stage is to build an **ontology**, which in its simplest form is a set of definitions of the domain objects and concepts (more later on in the module).
- Other useful formulation tasks are:
 - identification of rules and methods of inference;
 - identification of categories/similarities;
 - construction of causal models.

KBS Implementation Options

- Of course the KBS needs to be implemented. Though conventional languages could be used, most KBS developers opt for one of the following:
 - AI programming languages (e.g. PROLOG, LISP) which are *symbolically* orientated and based on the logic/functional programming paradigms.
 - KBS/ES shells, which allow development ***directly*** in a given KR formalism.

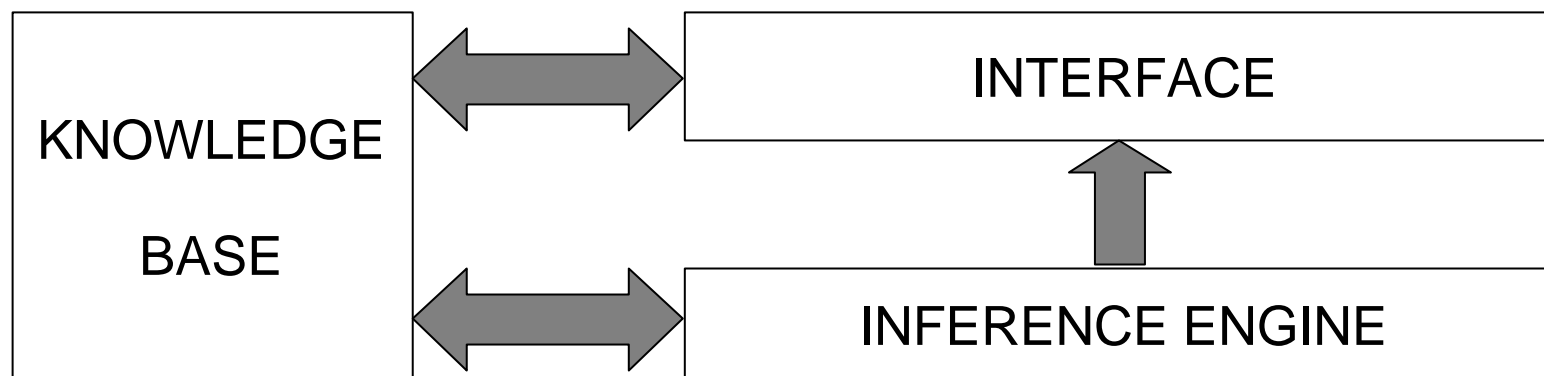
The earlier stages & project requirements determine the implementation platform, not the other way round!

KBS/ES Architecture

A KBS/ES is composed of (at least) the following:

- A **knowledge base (KB)**, contains the system's knowledge;
- An **interface**, allows use of the system and updates the KB;
- An **inference engine**, which does the reasoning;

Diagrammatically, this can be viewed as:



What is a KBS/ES Shell?

A KBS/ES shell is, at its simplest, a KBS/expert system with an empty knowledge base which can be filled to tackle a new problem.

Example: EMYCIN (derived from MYCIN) produced PUFF.

In practice, in order for shells to be useful, they need to allow the designer to edit/view the KB, and assess/repair faults.

It is **important** to choose a shell which adopts the reasoning mechanisms and KR methods **appropriate** to the problem.

We will look at ESTA which is used in your coursework.

Introduction to ESTA

ESTA is built in the Visual Prolog language. For our purposes it has three main advantages: it is ***simple***, it is ***easy to use***, and (most importantly) it is ***free***.

ESTA is a rule based shell that can handle small-scale ES applications, use the MS Windows GUI, interface with its applications, and has its own development environment.

It also (in its own fashion) provides:

- Both forward and backward chaining mechanisms.
- A range of variable types (e.g. Boolean, number, category).
- The ability to evaluate mathematical expressions.

ESTA: Getting Started

- Run ESTA and use **File ? Open** to load Hodge .kb into it.
- Now use **Consult ? Begin Consultation** see an example session, noting the following:
 - Clicking **why** will make ESTA generate a simple explanation from the KS, and **explain** produces a canned text explanation.
 - There are a range of input types - strings, numbers, etc.
- The **Consult** menu has options to view and check the KB.
- You need the **Parameter, Section, Edit,** and **Title** menus.

The *Title* and *Edit* Menus

The ***Title*** is rather basic and optional - it defines what appears in a pop-up window when the KB is loaded.

The ***Title*** menu has options that allow you to edit, delete, display, etc. the title and create a new one.

The ***Edit*** menu has the usual options you are familiar with for any MS Windows application so that you can cut and paste text.

For all for the menu options, an explanation is available for ESTA's help facility.....

Sections

Sections in ESTA are the main way of implementing a forward-chaining ES. They are edited via the Section menu and are of the form given in the example below:

```
section start : 'the first section to be executed'  
  if car_color = 'red'  
    ( advice 'Your car is red, try to sell it to the fire brigade ',  
      call sound(200,100) )  
  if car_color <> 'red' and car_color <> 'blue'  
    advice 'Your car is not one of our favourite colors !'  
  
advice 'That's all folks!'
```

There are a number of different actions, they are explained in ESTA's help facility. I would just stick with: **advice**, **assign**, **do**, **do section of**, **stop**, and **exit**.

Parameters I

ESTA uses conventional variables called ***Parameters***. Their values make up, in effect, the working memory. Parameters can be edited via the corresponding menu.

Parameters can be of the following types:

- **Boolean** - true/false
- **Text** - a string
- **Number** - self explanatory!
- **Category** - one from a set of values.

Rules can assign values, or questions can be asked.

Parameters II

Two examples of parameter definitions are:

```
parameter name 'the name of the user'  
type text  
question 'What is your name ?'
```

This defines the type of the parameter, and its question.

```
parameter pc_colour 'The user''s favourite screen colour'  
type category  
options  
  red,  
  blue,  
  green.  
question 'Which screen colour do you prefer when using an editor?'
```

We need to specify allowed values for category parameters.

Some Coursework Advice (Build)

- Do a design ***first*** - decide on the rules.
- Explicitly address & report all of the stages of the design.
 - i.e. KA ? Formulation ? KR ? Implementation.
- ESTA requires practice - so look at the example KB's.
- Adopt forward chaining, and use sections, as they help break down the reasoning into more sensible chunks (take advantage of this in the design).
- Do not try to use ESTA's advanced features (e.g. pictures), you'll waste time for no credit.
- Start early - don't leave it to the last minute.
- Write a good report and remember to test the system!

Good ESTA Examples

I advise you to look at the following KB's for the coursework - the others can confuse as they deal with advanced features!

- Car.kb - a fairly large fault diagnosis system.
- Cinema.kb - a small forward-chaining system.
- Danger.kb - a large first aid system.
- Easyesta.kb - a *really* simple example!
- Hodge.kb - a medium-size medical system.
- Safe.kb - another first aid system.
- Sanocor.kb - a large health assesment system.

Good Luck - and use the ESTA guide I've provided!

Evaluation, V & V

Of course the system needs to be **evaluated** to assess its *overall value in the context where it is to be applied*.

Part of this is **validation** of the system against the expert's knowledge and performance so that it does the 'right' things. The KBS also needs to be **verified** against its specification (whether the KB has been implemented correctly).

This process often has to be repeated several times in the life-cycle, due to the prototyping approach taken in KBS development, though SE approaches are largely applicable.

Turban is a starting point for further reading (projects?).