IMPORTANT

- **Mandatory Lab participation** is every two weeks (next mandatory Lab: February 11)
- **Optional Lab participation**: We will be in the Lab weekly (since time is allocated in the official calendar!) – so you have the option to consult with us and get help on your Lab Assignments

Optional Lectures on Brahms

- As of Tuesday February 2, every Tuesday and Thursday from 1-3PM the Brahms developer, Dr. Maarten Sierhuis – Professor at Carnegie Mellon University, will remotely lecture for us and for students at Delft University of Technology.
- The lectures will be transmitted out of my Adaptive Risk Management Lab located in room H 212.
- I strongly encourage you to attend these unique lectures, as you will have an opportunity to interact and team-up with students from Holland and do joint project for our class together with them using Brahms.

Where are we?

- We covered:
- What is AI and Why
- AI as in ‘agents that **behave** rationally’

Today we begin with the **HOW**

Investigate the fundamental AI techniques we can use to implement rational agents

Knowledge Representation

DELIBERATIVE ARCHITECTURES

Expert Systems

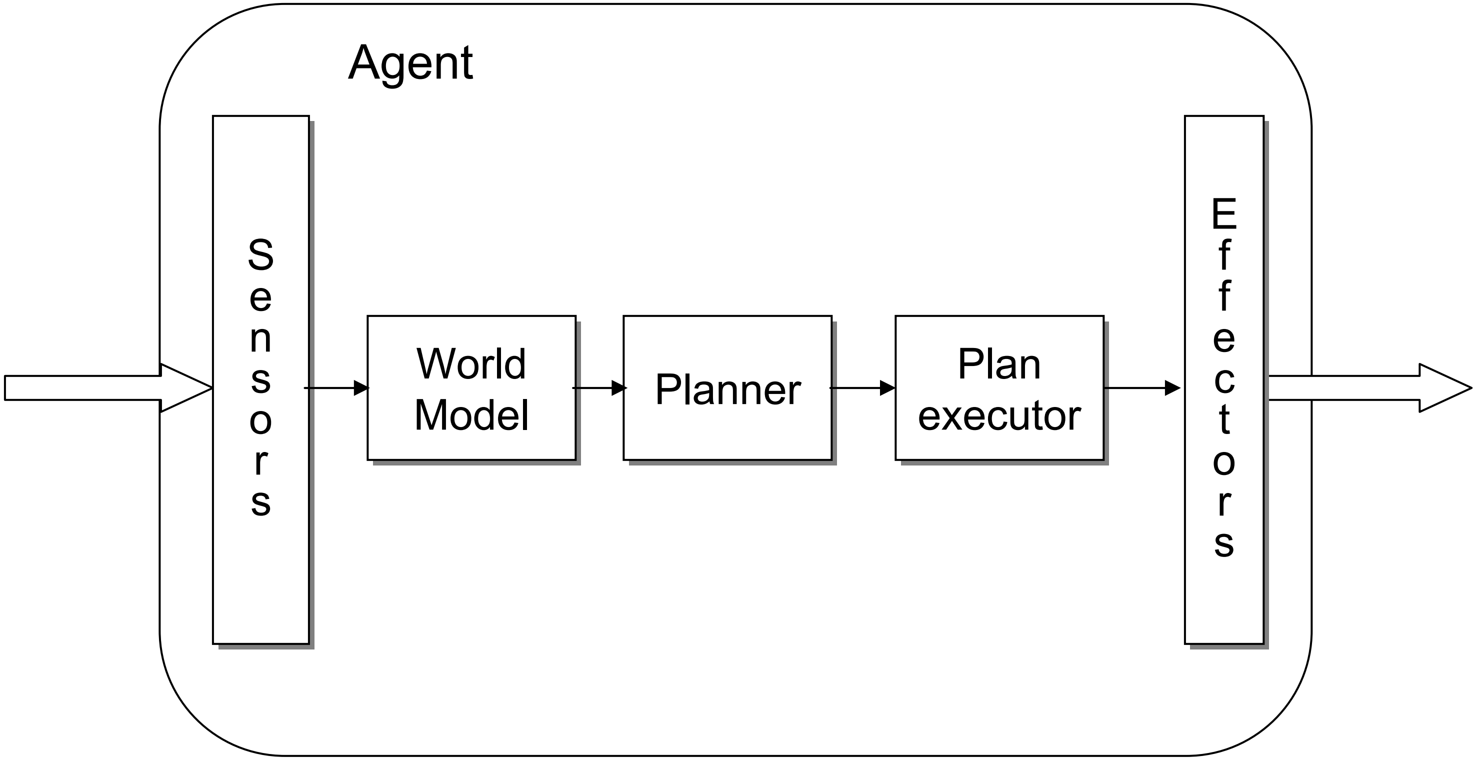
Rule-Based/Knowledge-Based

Logical Agents

Deliberative Architectures

Planning Agent

Main technique: Problem-Solving by Search

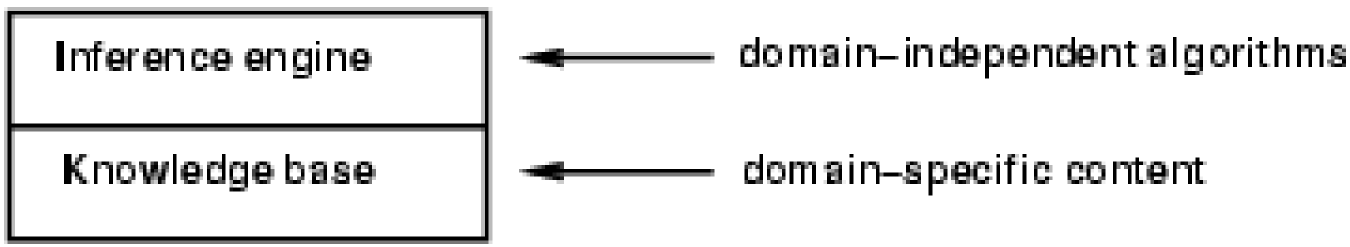


Goal-Based Agents and Utility-Based Agents

Knowledge-Based Agents

- **Knowledge Representation**
- **Requirements for KR**
- **Logic as a KR language**
- Logic in general
- Propositional (Boolean) logic
- Equivalence, validity, satisfiability
- Inference rules and theorem proving
  - forward chaining
  - backward chaining
  - resolution

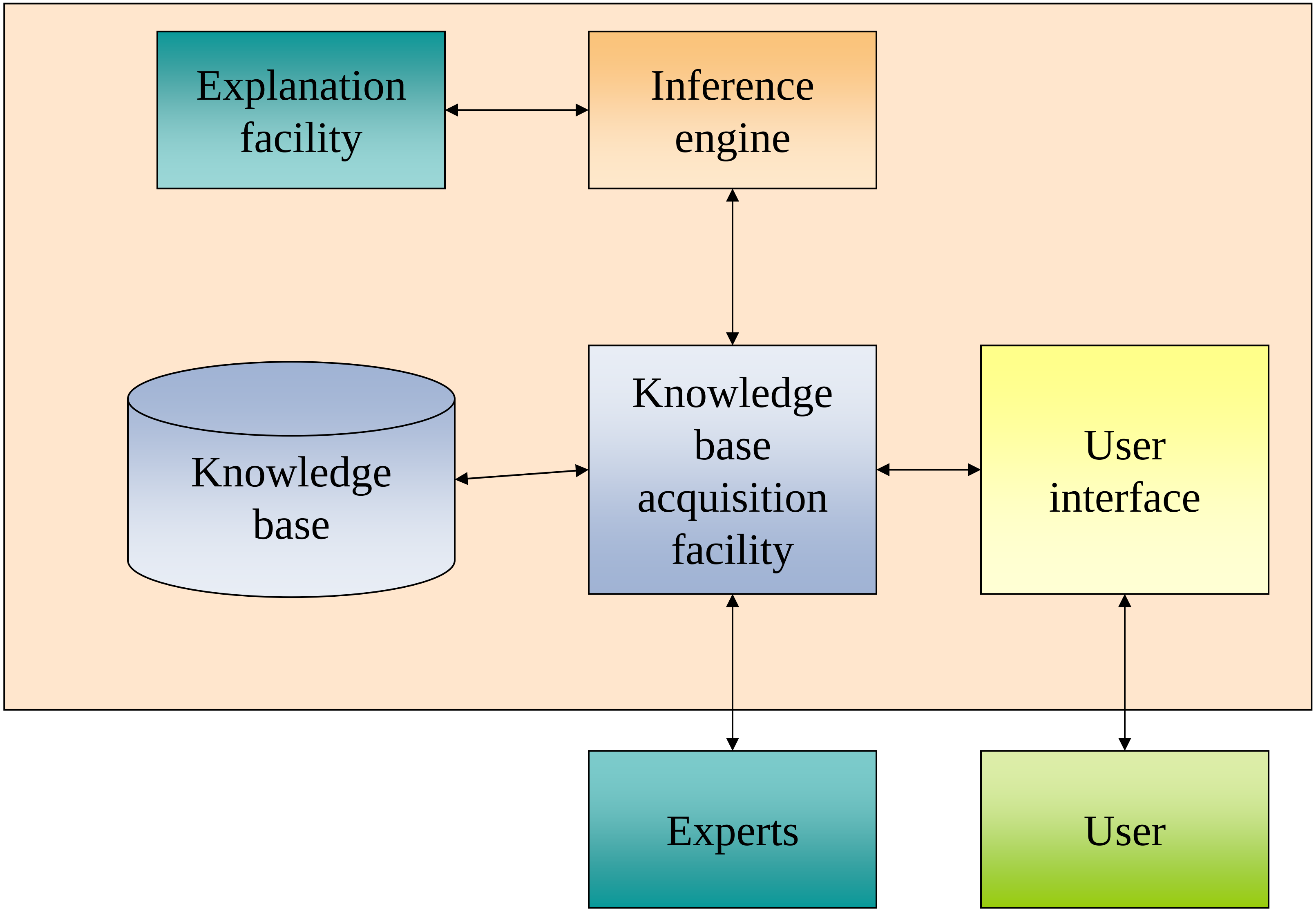
Knowledge bases



- Knowledge base = set of sentences in a formal language

- Declarative approach to building an agent (or other system):
  - `Tell` it what it needs to know
- Then it can `Ask` itself what to do - answers should follow from the KB

EXPERT SYSTEMS

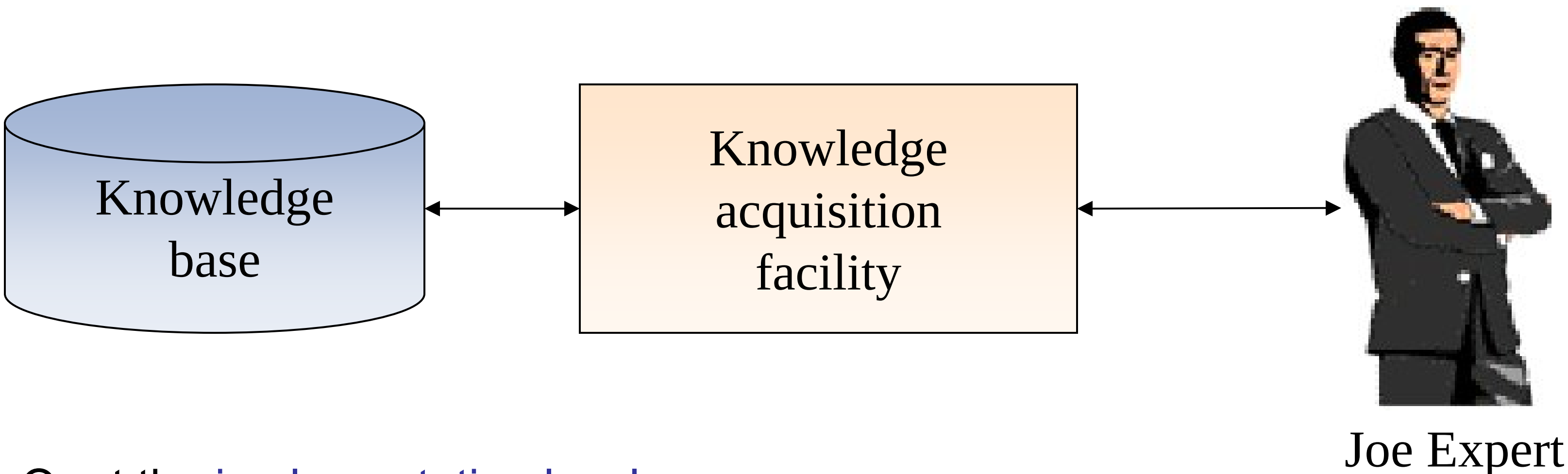


Agents can be viewed at the knowledge level

i.e., what they know, regardless of how implemented

Knowledge Acquisition Facility

- Knowledge acquisition facility
  - Provides a convenient and efficient means of capturing and storing all components of the knowledge base



Or at the implementation level

i.e., data structures in KB and algorithms that manipulate them

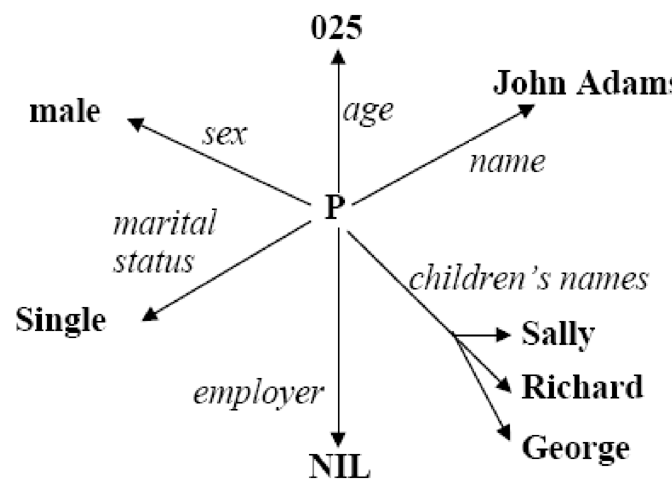
How about databases?

Simple databases are commonly used to good effect in Computer Science. They can be use to store and manipulate virtually any kind of information.

A Record Structure

John Adams
025
male
single
Sally
Richard
George
NIL

A Directed Graph



But storage and display are not enough — we also need to *manipulate* the knowledge

Databases as knowledge representation

Advantages

- Databases are well suited to efficiently representing and processing large amounts of data (and derivation from a database is virtually independent of its size).
- We can build on traditional database systems to process more complex and more powerful representational devices (e.g. frames).

Disadvantages

- Only simple aspects of the problem domain can be accommodated.
- We can represent entities, and relationships between entities, but not much more.
- Reasoning is very simple. Basically the only reasoning possible is simple lookup, and we usually need more sophisticated processing than that.

Inferential Adequacy

- Representing knowledge not very interesting unless you can use it to make inferences:
  - Draw new conclusions from existing facts.
    - “If its raining John never goes out” + “It’s raining today” so..
  - Come up with solutions to complex problems, using the represented knowledge.
- Inferential adequacy refers to how easy it is to draw inferences using represented knowledge.
- Representing everything as natural language strings has good representational adequacy and naturalness, but very poor inferential adequacy.

KR Approaches

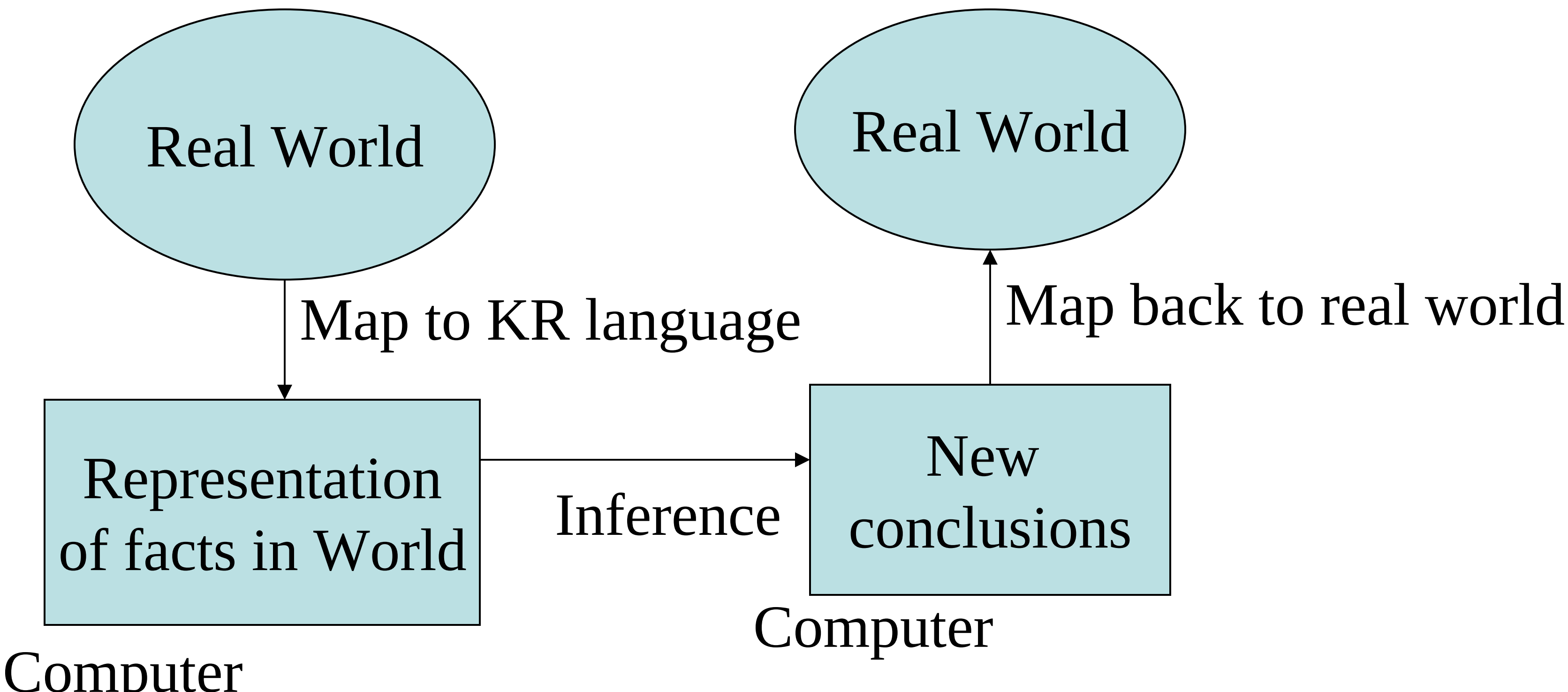
- So, neither natural languages nor traditional computer formalisms are not good enough for KR. . .
- Rule-based systems (AKA *production systems*)
- Expert systems
- Semantic networks and frames
- Graphical representation convenient for *knowledge engineers*
- Logic
- Formal semantics

Main KR Approaches

- Logic
- Frames/Semantic Netorks/Objects
- Rule-based systems

Well-defined syntax/semantics

- You must know exactly what an expression means in terms of objects in the real world.



A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
   static: KB, a knowledge base
           t, a counter, initially 0, indicating time

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

- The agent must be able to:
  - Represent states, actions, etc.
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

Inferential Adequacy

- KR scheme must allow us to make new *inferences* from old knowledge (reason on that knowledge, drawing new conclusions)

- It must make inferences that are:

sound — the new knowledge actually does follow from the old knowledge;

complete — it should make all the right inferences.

- Soundness usually easy; completeness very hard!

Example

- Given knowledge. . .

Michael is a man.

All men are mortal.

- the inference

Simon is mortal.

- is **not sound**, whereas

Michael is mortal.

- is **sound**.

Expert Systems

- **Expert systems** are commercially the most successful domain in Artificial Intelligence.
  - These programs mimic the experts in whatever field.

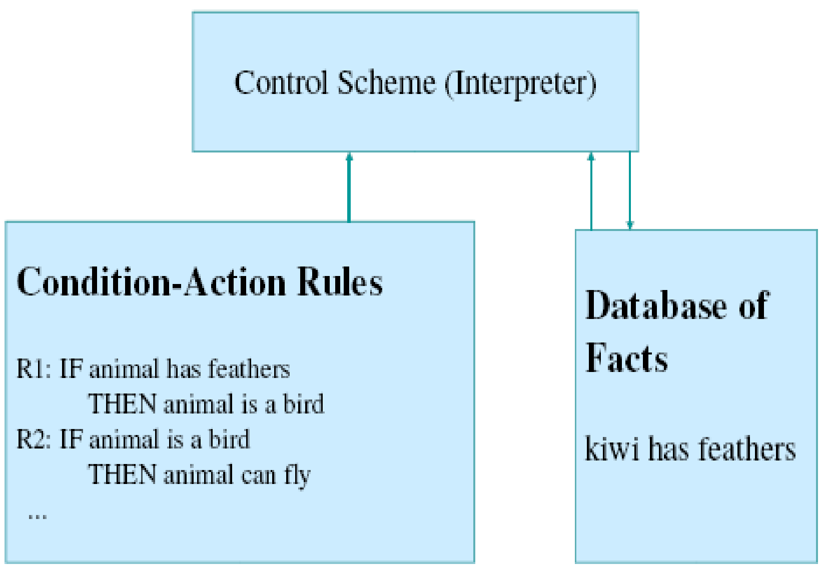
Auto mechanic	Telephone networking
Cardiologist	Delivery routing
Organic compounds	Professional auditor
Mineral prospecting	Manufacturing
Infectious diseases	Pulmonary function
Diagnostic internal medicine	Weather forecasting
VAX computer configuration	Battlefield tactician
Engineering structural analysis	Space-station life support
Audiologist	Civil law

Rule-Based Systems

- Knowledge is specified as a collection of *rules*.
- Each rule has the form
- condition −! action
- which may be read if condition then action.
- The condition (antecedent) is a *pattern*.
- The *action* (consequent) is an operation to be
- performed if rule *fires*.
- Knowledge is applied to *facts*—unconditional
- statements that are assumed to be correct (at the time
- they are used).

Rule-Based Systems Architecture

- A collection of rules
- A collection of facts
- A rule *fires* if a fact *matches* the *condition* of the rule
  - Mechanism that fires rules is *inference engine*.



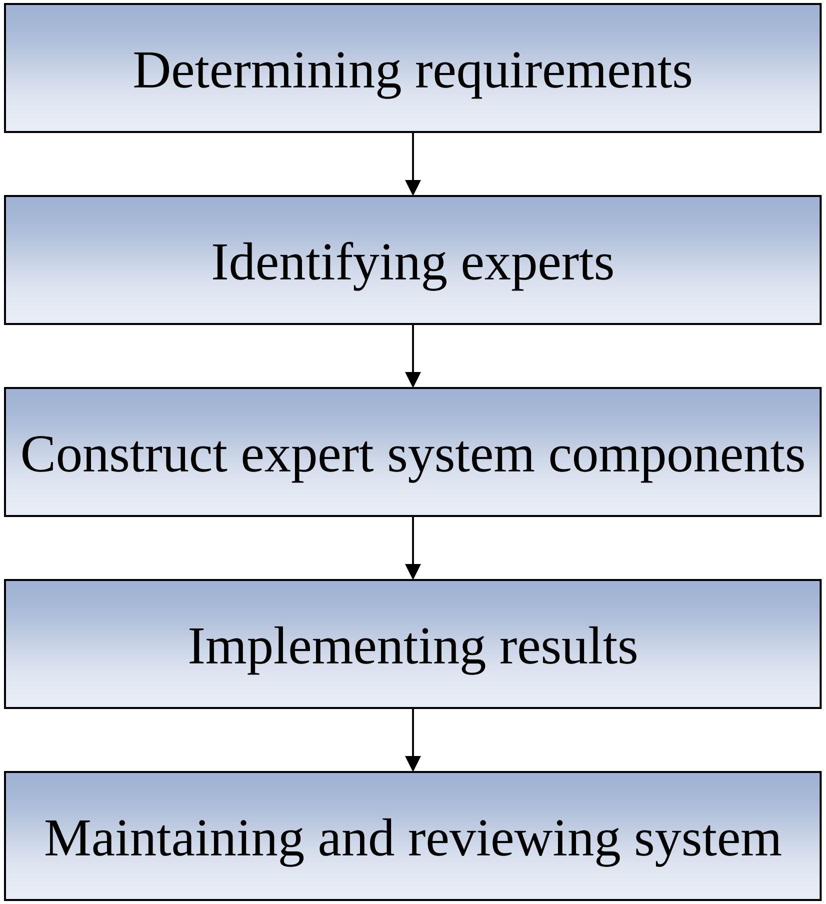
Expert Systems

- Expert systems are also called Rule-based systems.
  - Expert's expertise is built into the program through a collection of rules.
  - The desired program functions at the same level as the human expert.
  - The rules are typically of the form:
    - If (some condition) then (some action)
    - Example: If (gas near empty AND going on long trip) then (stop at gas station AND fill the gas tank AND check the oil).

Expert Systems

- Two major parts of an expert system:
  - The **knowledge base**: The collection of rules that make up the expert system.
  - The **inference engine**: A program that uses the rules by making several passes over them.
    - On each pass, the inference engine looks for all rules whose condition is satisfied (*if* part).
    - It then takes the action (*then* part) and makes another pass over all the rules looking for matching condition.
    - This goes on until no rules' conditions are matched.
    - The results are all those action parts left.

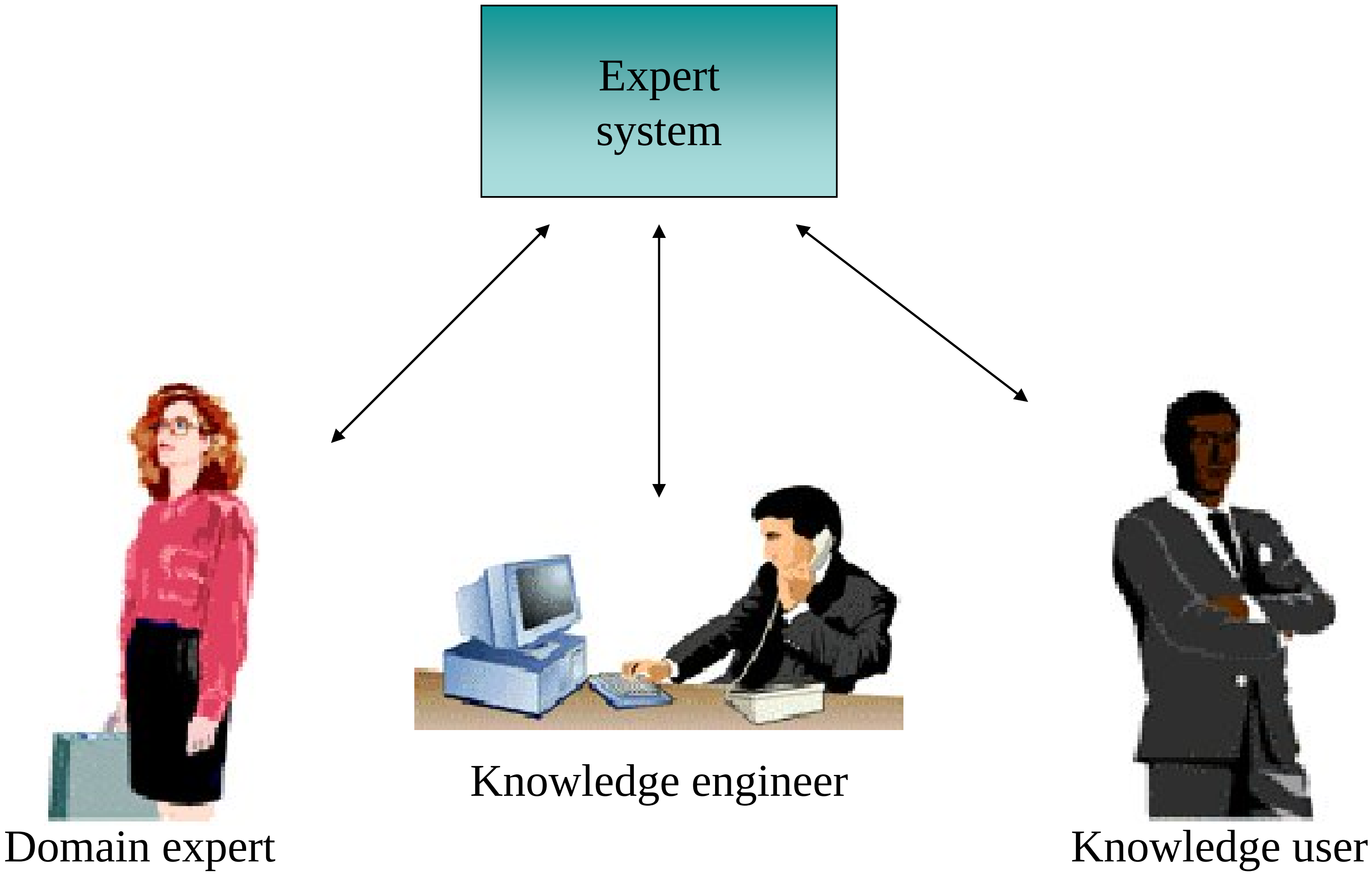
Expert Systems Development



Domain

- The area of knowledge addressed by the expert system.

KNOWLEDGE-BASED SYSTEM



KNOWLEDGE ELICITATION

Participants in Expert Systems Development and Use

- Domain expert
  - The individual or group whose expertise and knowledge is captured for use in an expert system
- Knowledge user
  - The individual or group who uses and benefits from the expert system
- Knowledge engineer
  - Someone trained or experienced in the design, development, implementation, and maintenance of an expert system

Schematic

Agents vs expert systems

- Expert systems which started appearing in the 1970s were heralded as the first intelligent pieces of software

Are they agents?

- Although intelligent, they have no control over their actions
- They only provide domain-specific information and usually do not adapt
- No true social ability: limited interaction with humans or other (expert) systems
- They are decoupled from their environment

Knowledge in AI

- In order to solve the complex problems encountered in AI, one generally needs a large amount of knowledge, and suitable mechanisms for representing and manipulating all that knowledge.
- ". . . power exhibited . . . is primarily a consequence of the specialist knowledge employed by the agent and only very secondarily related to . . . the power of the [computer]. Our agents must be knowledge rich, even if they are methods poor."
- [Ed Feigenbaum]

The Role of Knowledge

- Knowledge about a domain allows problem solving to be *focused* — not necessary to exhaustively search.
- *Explicit* representations of knowledge allow a *domain expert* to understand the knowledge a system has, add to it, edit it, and so on.
- ***Knowledge engineering***.
- Comparatively *simple* algorithms can be used to *reason* with the knowledge and derive *new* knowledge.

What is Knowledge?

- Knowledge is *information* about some domain or subject area, or about how to *do* something.
- Knowledge can take many forms. Some simple
- examples are:
- Eve is a female, Adam is a male.
- Females with children are mothers.
- Mothers are females, fathers are males.
- etc.

How to *Represent* Knowledge?

- Why don’t we use **natural languages** (e.g. English) to represent Knowledge
- Natural language is certainly expressive enough!
- But it is also too ambiguous for automated reasoning.
- No clear semantics.
- Semantic ambiguities
- “bank” is “river bank” or “financial bank”
- “model”
- What about ‘computer’ languages?
- You must know exactly what an expression means in terms of objects in the real world.

Requirements for KR

- So, how we *represent* knowledge in a form amenable to computer manipulation?
- **Desirable features** of KR scheme:
- *representational adequacy*;
- *inferential adequacy*;
- *inferential efficiency*;
- *well-defined syntax & semantics*;
- *naturalness*.

Representational Adequacy

- A KR scheme must be able to actually represent the knowledge appropriate to our problem.
- Some KR schemes are better at some sorts of knowledge than others.
- *There is no one ideal KR scheme!*

Representational adequacy

- Consider the following facts:
  - John believes no-one likes him.
  - Most children believe in Santa.
  - John will have to finish his assignment before he can start working on his project.
- Can all be represented as a string! But hard then to manipulate and draw conclusions.
- **How do we represent these formally in a way that can be manipulated in a computer program?**
- Some notations/languages only allow you to represent certain things.
  - Time, beliefs, uncertainty, all hard to represent.

Symbolic Representation

Assumption of (traditional) AI work is that:

- Knowledge may be represented as “symbol structures” (essentially, complex data structures) representing bits of knowledge (objects, concepts, facts, rules, strategies..).
  - E.g., “red” represents colour red.
  - “car1” represents my car.
  - red(car1) represents fact that my car is red.
- Intelligent behaviour can be achieved through manipulation of symbol structures

KR Languages

- Knowledge representation languages have been designed to facilitate this.
- Rather than use general C++/Java data structures, use special purpose formalisms.
- A KR language should allow you to:
  - represent adequately the knowledge you need for your problem (*representational adequacy*)
  - do it in a clear, precise and “natural” way.
  - allow you to reason on that knowledge, drawing new conclusions.

Well defined syntax/semantics

- Suppose we have decided that “red1” refers to a dark red colour, “car1” is my car, car2 is another..
- **Syntax of language will tell you which of following is legal**: `red1(car1), red1 car1, car1(red1), red1(car1 & car2)?`
- **Semantics of language tells you exactly what an expression means** - e.g., `Pred(Arg)` means that the property refered to by `Pred` applies to the object refered to by `Arg`. E.g., property “dark red” applies to my car.

Syntax and Semantics

- It should be possible to tell:

- whether any construction is “grammatically correct”.
- how to read any particular construction — no *ambiguity*.

- Thus KR scheme should have *well defined syntax*.
- It should be possible to precisely determine, for any given construction, exactly what its meaning is.
- Thus KR scheme should have *well defined semantics*.
- *Syntax is easy, semantics is hard!*

A natural representation scheme?

- Also helpful if our representation scheme is quite intuitive and natural for human readers!
- Could represent the fact that my car is red using the notation:
  - “`xyzzy ! Zing`”
  - where `xyzzy` refers to redness, `Zing` refers to my car, and ! used in some way to assign properties.
- But this wouldn’t be very helpful..

Naturalness

- Ideally, KR scheme should closely correspond to our way of thinking, reading, and writing.
- Allow *knowledge engineer* to read & check *knowledge base*.
- Again, *more general* a KR scheme is, less likely it is to be readable & understandable.

Inferential Adequacy

- Representing knowledge not very interesting unless you can use it to make inferences:
  - Draw new conclusions from existing facts.
    - “If its raining John never goes out” + “It’s raining today” so..
  - Come up with solutions to complex problems, using the represented knowledge.
- Inferential adequacy refers to how easy it is to draw inferences using represented knowledge.
- Representing everything as natural language strings has good representational adequacy and naturalness, but very poor inferential adequacy.

Inferential Adequacy

- KR scheme must allow us to make new *inferences* from old knowledge (reason on that knowledge, drawing new conclusions)

- It must make inferences that are:

sound — the new knowledge actually does follow from the old knowledge;

complete — it should make all the right inferences.

- Soundness usually easy; completeness very hard!

Example

- Given knowledge. . .

Michael is a man.

All men are mortal.

- the inference

Simon is mortal.

- is not sound, whereas

Michael is mortal.

- is sound.

Inferential Efficiency

- You may be able, in principle, to make complex deductions given knowledge represented in a sophisticated language.
- But it may be just too inefficient.
- Generally the more complex the possible deductions, the less efficient will be the reasoner.
- Need representation and inference system sufficient for the task, without being hopelessly inefficient.

Inferential Efficiency

- A KR scheme should be *tractable* — make inferences in reasonable time.
- Unfortunately, *any* KR scheme with interesting *expressive power* is not going to be efficient.
- Often, the more *general* a KR scheme is, the *less efficient* it is.
- Use KR schemes tailored to problem domain — less general, but more efficient.

Examples

- *Arithmetics*
- The expression A + B > 3 is correct while . . .
- . . . A + B > is not
- A + B > 3 is either the fact “true” or “false”
- depending on the values of A and B
- *Java*
- if(bePolite)
- System.out.println("Good morning");
- else
- System.out.println("I am busy");