Fundamental Issues in MFKB Endeavour

- How to **construct** (and maintain) large body of knowledge
- How to facilitate reuse (**long shelf-life**)
- How to generate suitable **'views' or abstractions** of knowledge for question-answering
- How to answer **novel** (unanticipated) questions

Notion of **components** is central to all these issues.

Motivation for Component-Based Approach

I: The Knowledge Base

- Reusing the top levels of the KB
- Building the KB; powerful KA tool

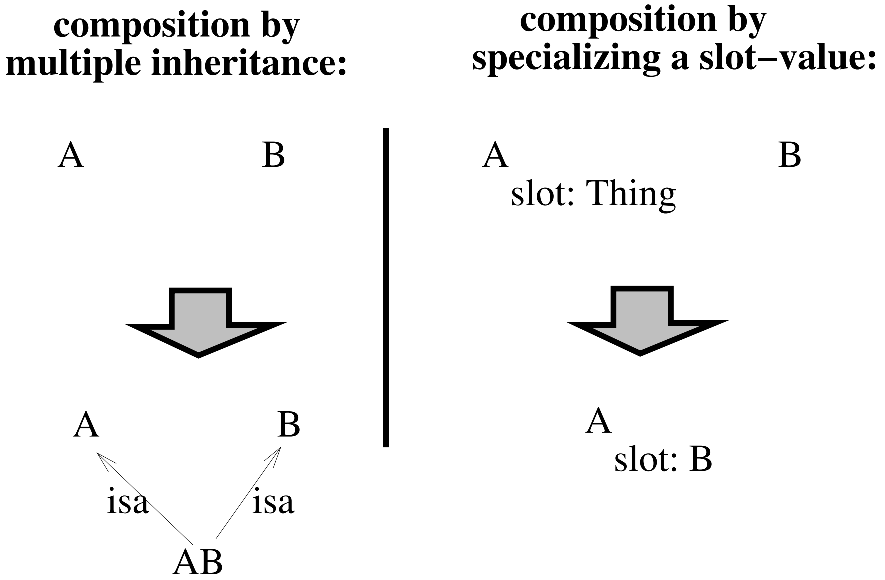
II: The Application Systems

- Automatically constructing scenario to "understand" a user's problem in DCE
- "Teasing apart" / abstraction of concepts

Concept Composition

A fundamental theme in AI: Many concepts are "compound".

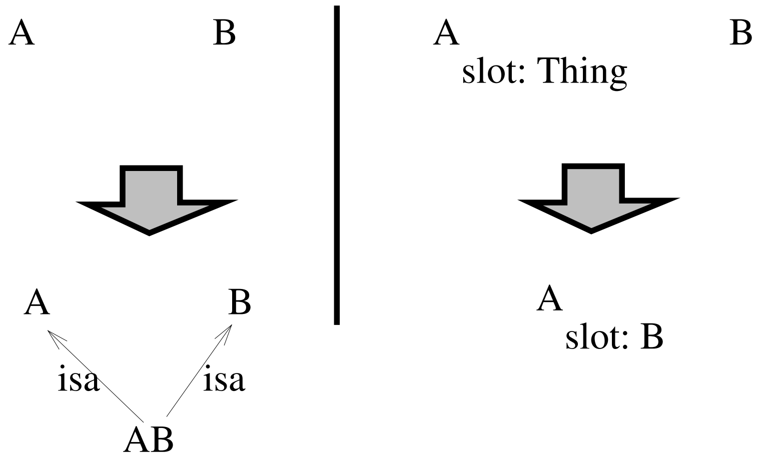
- Two standard KB-based methods are
  - multiple inheritance
  - slot-value specialization



- BUT: Doesn't take account interactions of composed concepts.
- Recognized as valid but limited notion of composition in s/w engineering

Addendum Comments

composition by multiple inheritance: **composition by specializing a slot−value:**



Examples of the above composition methods are:

- we'd like to construct 'pet fish' from 'pet' and 'fish' (eg. via multiple inheritance), and be able to conclude things such as it's probably colourful, and it lives in a domestic water container (eg. a bowl).
- we'd like to construct 'fish bowl' from 'bowl' and 'fish' (eg. via slot-value specialization), and be able to conclude things such as it's probably transparent, and it contains water.

These conclusions can't be drawn automatically using the above composition methods. As a result we have to add these features in manually. And as a result of that we lose all the benefits of compositionality which we were seeking, and pay a heavy price:

- Every domain concept has to be explicitly created by the knowledge engineer.
- The top-level concepts in the KB have a tendency to become near-vacuous, as composition can't fully exploit them.
- The knowledge base becomes vast and unweildy, and contains large numbers of specific concepts.
- The KB becomes difficult for others to maintain or extend.
- Novel concepts can't be created or handled at run-time.
- As we don't know the components of concepts, we've no mechanism for easily generating coherent abstractions – an essential requirement for question-answering software.

Example: File Server

```
Server
------
  host: Machine
  provides: Service
  interacts-with: Source
                  ------
                  source-of: (Self provides Service)
                  access-script: Script
  software: Software
            --------
            controls: (Self interacts-with Source)
  endpoint: Endpoint
  ....
```

Some questions we'd like to be able to answer from the composition:

```
file-server = ?
size of software of file-server?
access-script of file-server?
h/w requirements of file-server?
```

Simply changing `Service` to `File` doesn't answer these questions.

An Extended Notion of Composition

1. Composition as **unification**
2. Specialization permits **elaboration**

Unification: we merge the graphs representing the component concepts:

- Frame represents a graph of relations
- Compose 2 concepts = merge 2 graphs
  1. Specify nodes to match
  2. Match graphs spreading out from those nodes

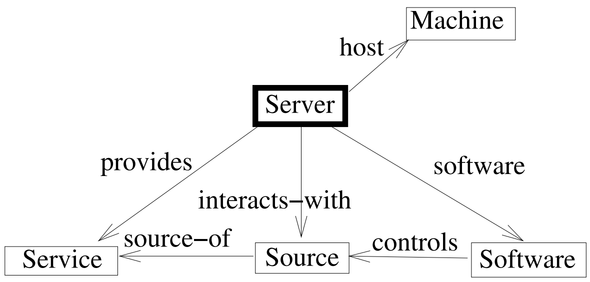
Elaboration: Specialized structures may correspond to other frames in the KB.

- May 'recognize' subgraphs in a graph
  $\Rightarrow$ can elaborate.

Composition as Unification: Example

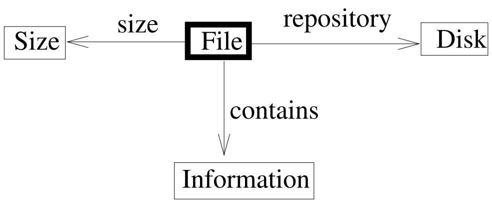
Concept 1:

Server
host: Machine
provides: Service
interacts−with: Source
source−of: (Self provides Service)
software: Software
controls: (Self interact−with Source)
endpoint: Endpoint
...

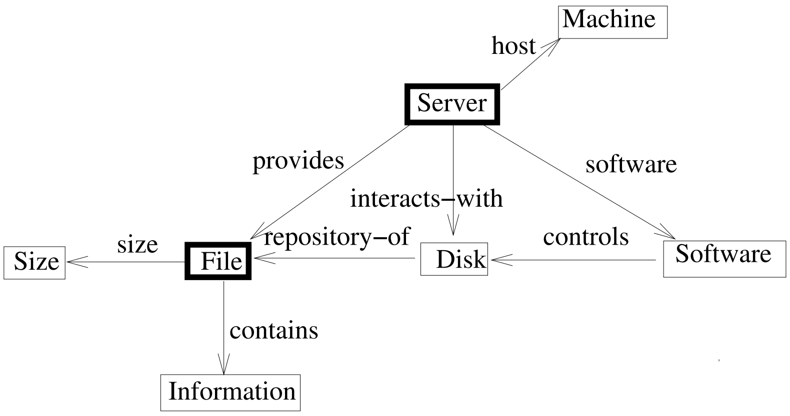


Concept 2:

File
size: Size
contains: Information
repository: Disk

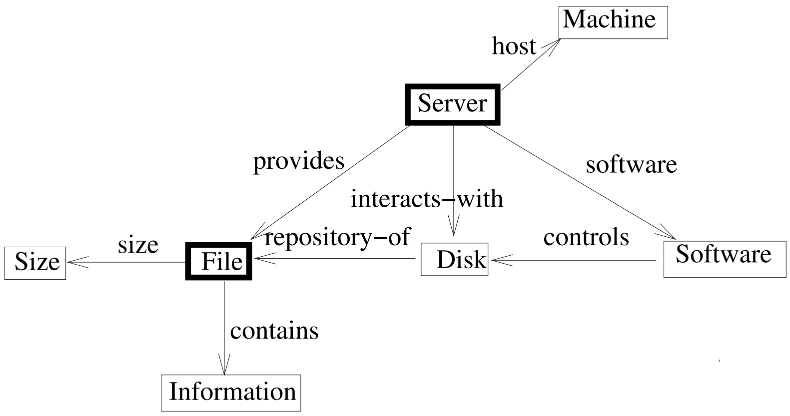


Composition:

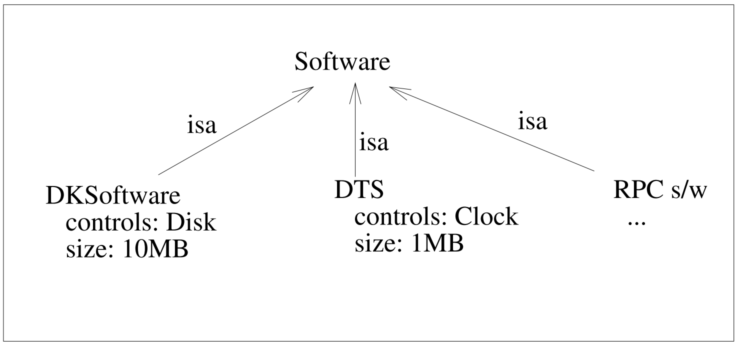


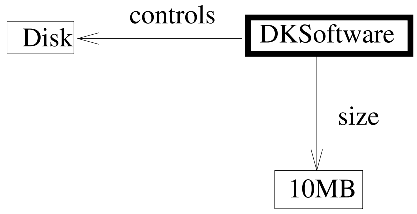
Concept Elaboration: Example

Specialized Graph:

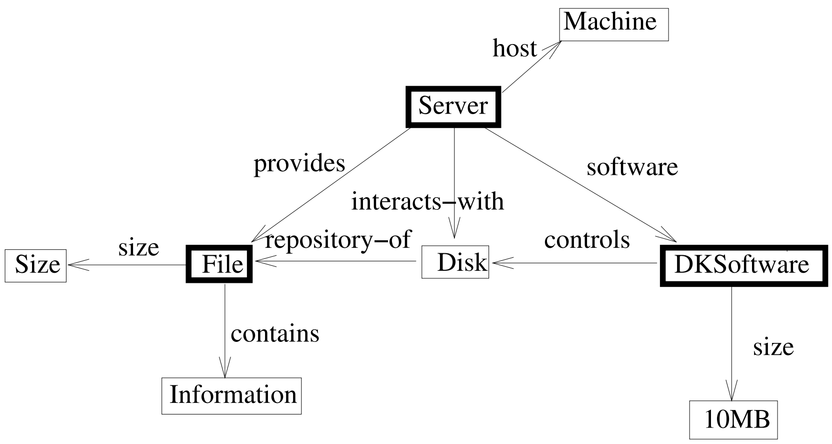


Domain Knowledge:





==> Elaborated Graph:

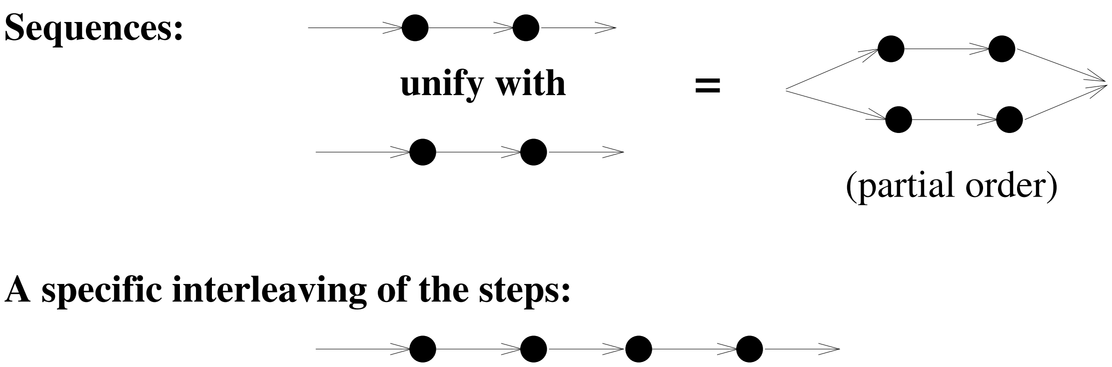


Automatic Composition

- Interactions between concepts propogate
- Automatically realizes "knock-on effects"

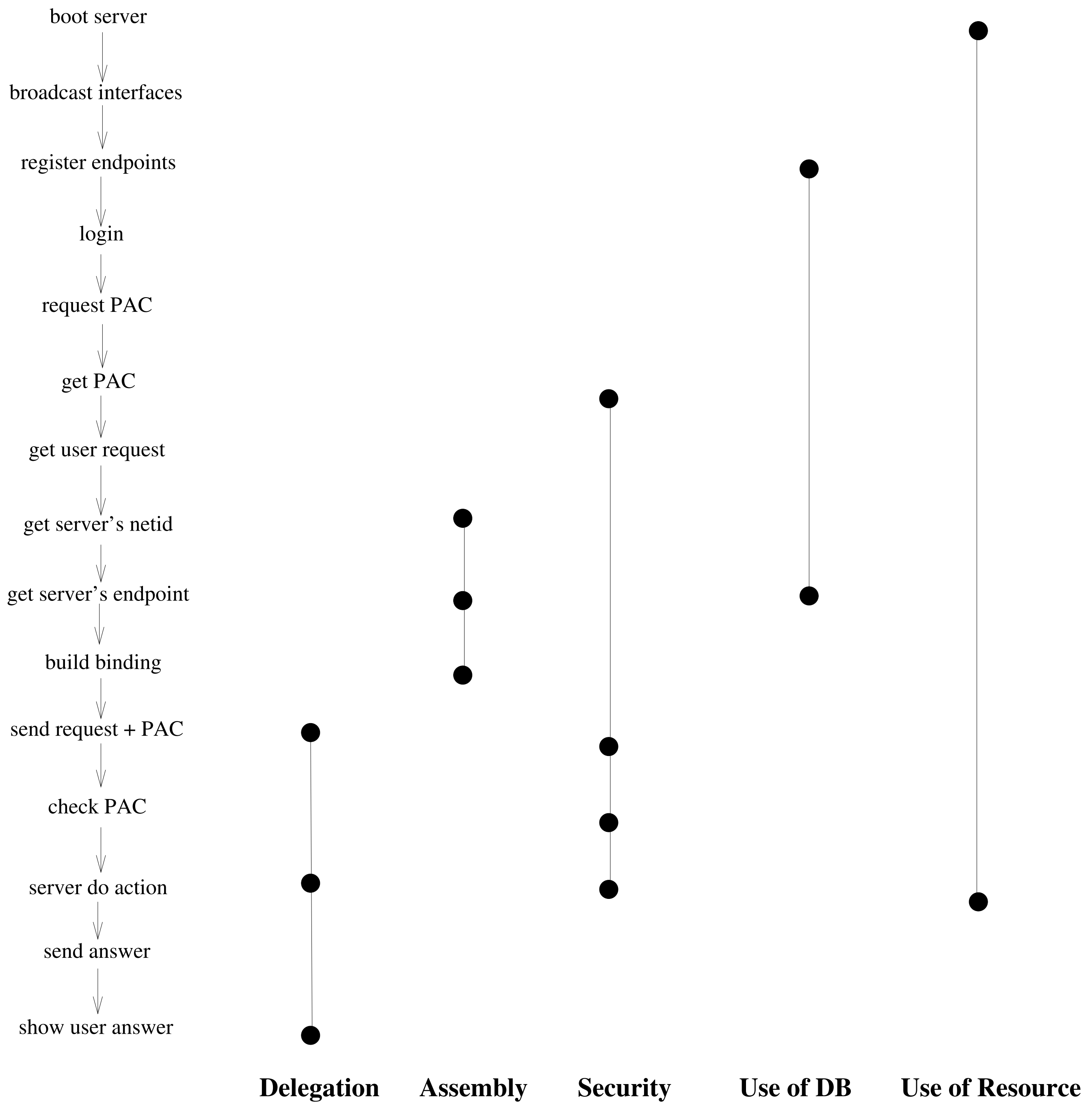
Scripts

Can apply a similar notion to unification of sequences.



Script = composition of scriptal components.

DCE Script + Component Threads



III: Components of Actions

Actions:

- Describe dynamic aspects of domain.
- Again, can be viewed as unification of components.

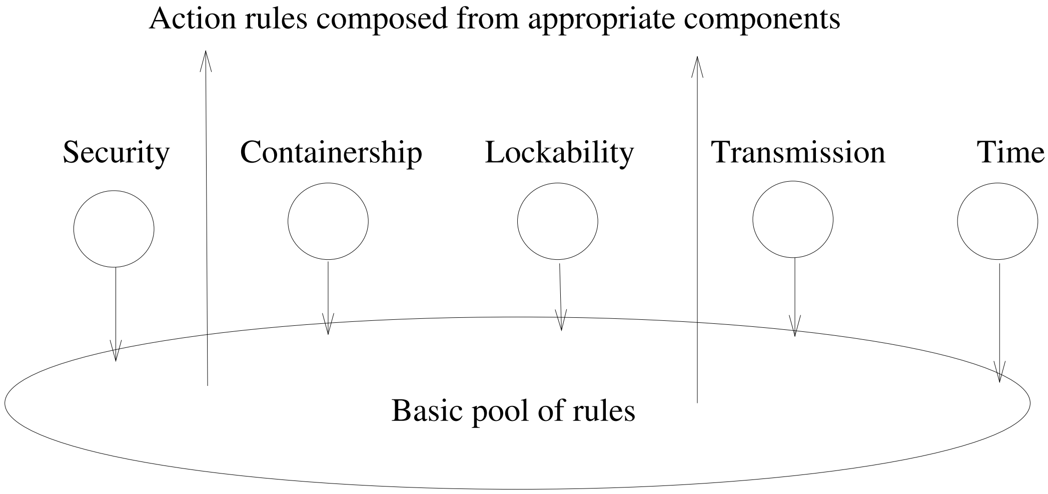
```
put(Agent,Data,DB) ::                     % SOURCE:
    PC  authorized(Agent, DB)             % authorization
        not locked(DB)                    % lockability
        free_space(DB, FreeSpace)         % volume
        size(Data, Size)                  % volume
        Size =< FreeSpace                 % volume
    DEL free_space(DB, FreeSpace)         % volume
    ADD free_space(DB, FreeSpace-Size)    % volume
        in(Data,DB)                       % container
```

Building An Action from Components

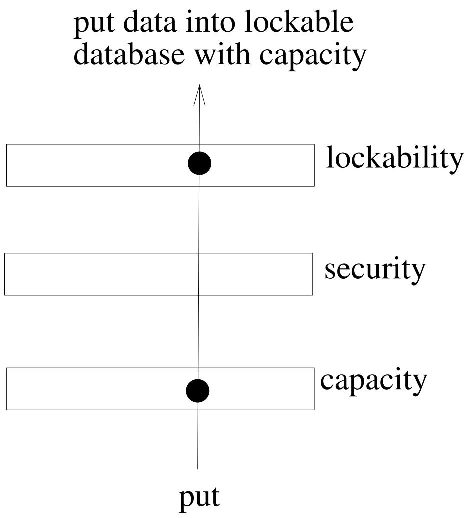
		CONTAINER		CAPACITY		LOCK		SECURITY
an authorized agent putting data in a volumetric, lockable container	=	put	+	insertion into volume	+	access of lockable item	+	authorized activity
		PC – del – add in(Data,DB)		free_space(...)		not locked(Obj) – –		authorized(Agent) – –

Composition of Actions from Components

1. Components contribute basic actions
2. Components modify (specialize) basic actions as appropriate for particular situations



Can view components as "layers" which transform (specialize) basic rules:



Pulling it All Together: Building a KB From Components

1. Define the **vocabulary** to use. This:
   - introduces concepts ("things" and "actions") to be used.
   - handles combinatorics of their interactions
2. Define the **ontology** of the domain, eg.

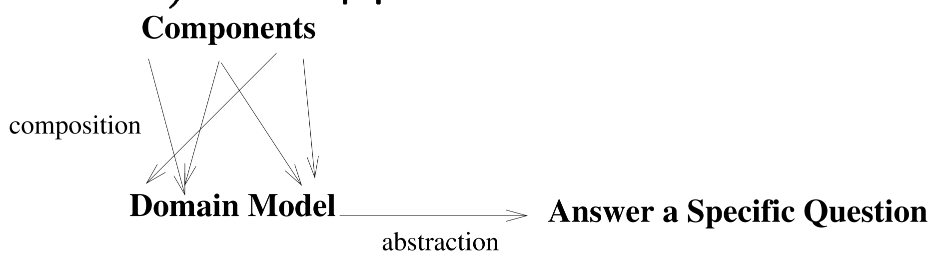
```
database = secure lockable container
server = software agent
machine = server's host
cell = collection of machines
disk = file repository
destroy(file) = delete
binding = server address
ticket = time-limited token of identity
```

3. Define **scripts**

Application Systems

Composition for:

1. Constructing KB
2. Generating abstractions (by removing components) for applications



Ability to selectively include/ignore aspects of the domain is crucial to question-answering.

Abstraction in Question Answering: Example

```
Trying to start the cdsadv manually,            SCENARIO
I see the following:
# cdsadv -e\*
Can't initialize shared memory cache            HINT
Cannot auto-enable clerk

Any suggestions on what can cause this?

...

You couldn't communicate with the CDS server    SCAN SCRIPT /
for some reason. The most likely explanation       COMMUNICATION
is that the network masks on your system aren't    ABSTRACTION
set right to permit the CDS broadcast protocol
to work.

....

When running the Starter Kit on a minimal       SCAN SCRIPT /
system I found that my swapfile was way too         CAPACITY
small. I went to 60mg for the swapfile and          ABSTRACTION
was able to run CDS.
```

Summary

- Component approach a fundamental requirement of KBs
- Traditional methods of multiple inheritance / slot-value specialization have limited compositional power
- Use unification as a basis for more powerful notion of composition
- Can be applied to facts, scripts (sequences) and actions (knowledge about facts)
- Automatic composition essential for both tractibly building the KB, and automatically generating abstractions of concepts for question-answering