An Application of CBR in Design System Generation

Task

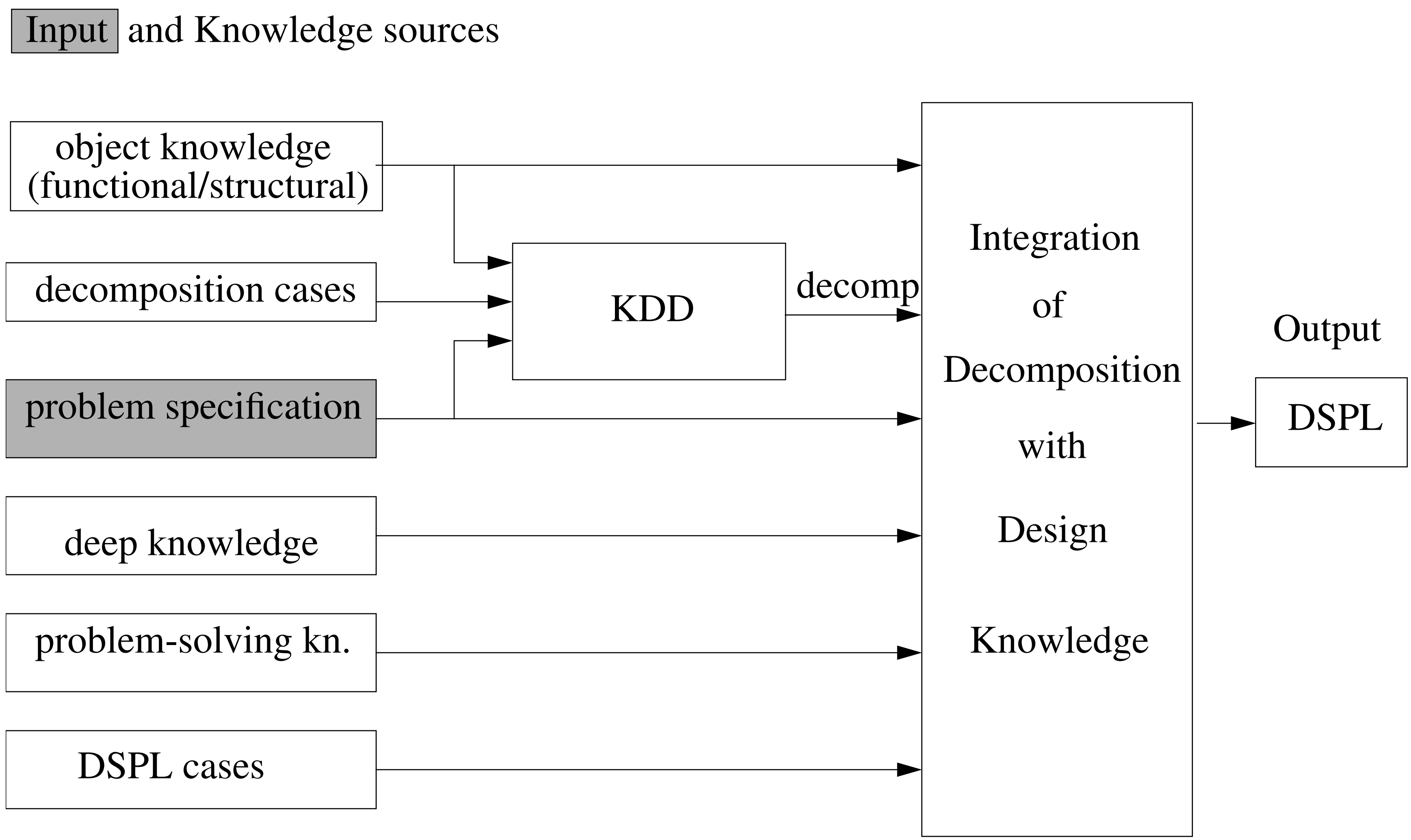
- **Task:** Synthesis of Parametric Design Systems
- **Inputs:** Design Object, Design Specification
- **Output:** **A** Design Expert System for the given object
- **Constraints:** System is in DSPL; it should design efficiently
- **Characteristics:** The design is decomposable, so both cases and subcases are used; It is the design plan with associated problem-solving knowledge, rather than the design result, that is reused.
- **System Name:** **Integration of Decomposition with Design Knowledge**
- **Status:** Ongoing

Reasoning framework

- **Reasoning framework:** CBR with subcase assembly + non-CBR patching
- **Reuse motivations:** To obtain design knowledge from the cases that is directly reusable, rather than from deep knowledge.
- **Benefits:** More tractable; Better efficiency/solution quality
- **Limitations:** Relies on the case library; Needs assistance from other methods

An Application of CBR in Design System Generation

The System: Integration of Decomposition with Design Knowledge

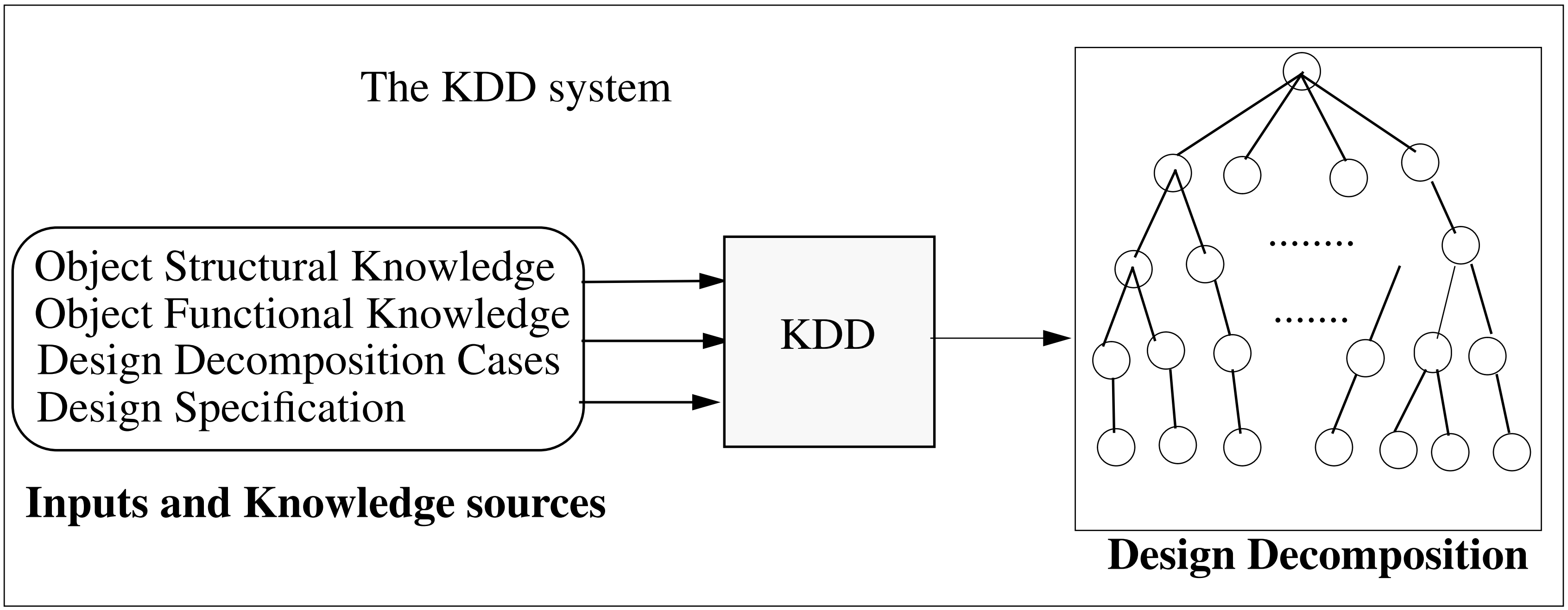


An Application of CBR in Design System Generation

Components of the System (1)

KDD (Knowledge-based Design Decomposition System) [J. Liu 1993]

A front-end system that generates decomposition framework for the target system



Note about the output:

Each node is a group of attributes.

Root node contains all attributes; leaf nodes single attributes.

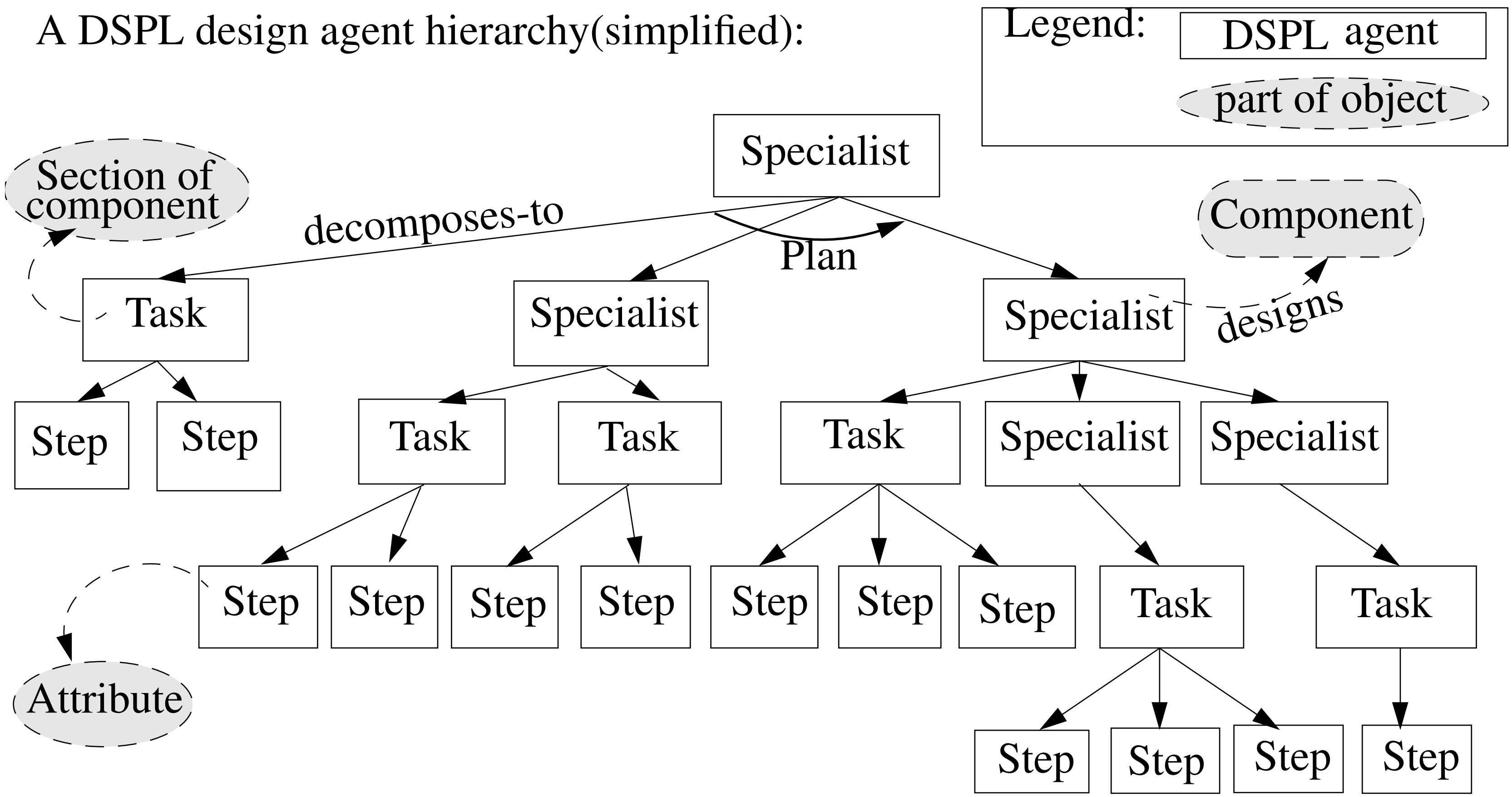
Ordering of attributes exists within each node.

Components of the System (2)

DSPL (Design Specialists and Plans Language) [Brown & Chandrasekaran 1989]

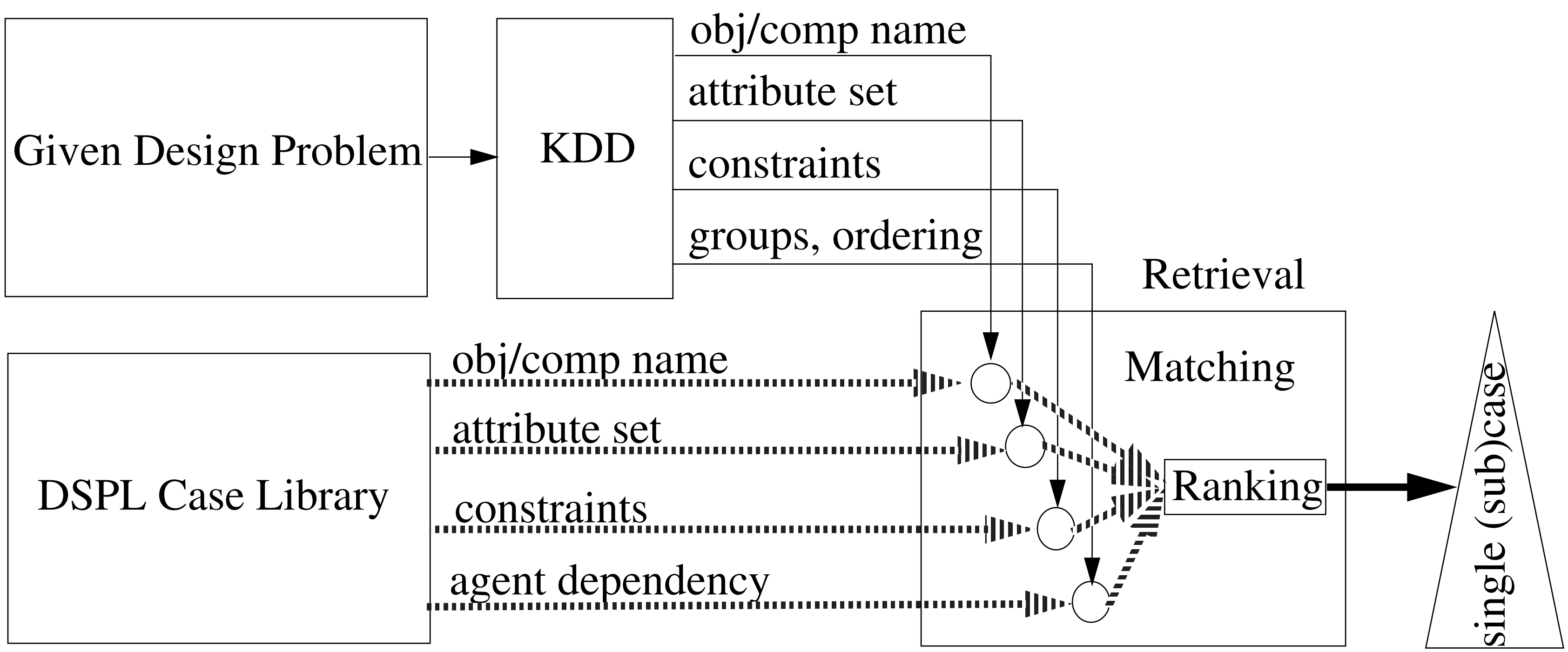
A language that describes how a hierarchy of design agents do design on different decomposition levels, as shown below.

A DSPL design agent hierarchy(simplified):



Design Knowledge Reused

- **Contents:** Control/Plans/Decomposition/Problem-solving/Constraint testing.
- **Representation:** DSPL: cases and subcases.
- **Used:** Via case retrieval/adaptation/assembly.
- **Acquisition:** Hand-coded by/from experts.
- **Gained:** Reduced complexity in synthesizing the design system;
  Efficient and good quality of the design;
  Use of expert knowledge/experience.
- **Case indexing and retrieving:**



Reuse/Adaptation Issues

- The design knowledge needed for target system is hard to obtain via compilation from deep knowledge, so reuse is preferred to synthesis.
- Adaptation is hard, so minimize adaptation. It's hard because:
  1. Applicability assessment and static evaluation of DSPL cases is hard.
  2. DSPL subcases are context sensitive and use in different contexts requires more understanding of the cases than from just surface features.
  3. The complexity of fixing a problem in a retrieved case is comparable to synthesizing a same sized piece of system.
- Well-structured DSPL cases and subcases form a decomposition hierarchy, facilitating case assembly.
- DSPL hierarchy is searched top-down with larger cases preferred to avoid adaptation/assembly.
- When no good case is found for an object/component, its next level subobjects/subcomponents in KDD decomposition are built from matched subcases, and assembled.
- Non-CBR methods such as constraint manipulation and symbolic evaluation are used to patch the small portion that can't be built from adapting/assembling cases.

Adaptation methods

Without more understanding of the case than the surface indices, case adaptation can be driven by mismatch of those indices at retrieval stage as follows:

Mismatch in the DSPL case	**Adaptation to the case**
Extra attributes	Ignore or discard
Different attribute ordering	Adjust the order if it does not contradict the dependencies
Missing attributes	Synthesis
Constraint mismatches: extra missing differring	Discard Decomp constr & merge it in the case Parameterize constraint