Preface

The growing amount of knowledge creates new opportunities as well as challenges. Unfortunately, the challenges often start to be the problems. In chemical and process engineering, the most common problems related to the huge amount of available data, information and knowledge are: difficulties with estimation of their quality, lack of efficient methods enabling the fast access to the relevant information or knowledge and "use once" model of knowledge application. The above-mentioned problems are common for all activities in chemical and process engineering: modelling, simulation, design and control. However, design phase is critical from the point of view of the satisfactory functioning of the process unit or the whole system. The wrong assumptions or errors made at this stage could be corrected only with the great amount of time and money but often it is too late for any essential change. The design is difficult as usually there is a lot of uncertainty involved. The good designers used to deal with the problem using their intuition supported by the past experience. The trouble is that industry and society are more and more innovation hungry. There is a growing demand for designs which are less and less similar to their predecessors.

There are two major approaches to deal with this situation, either to make new experiments, develop new models and on this basis build new designs or to use the existing information and knowledge. The second option is much more economically viable and less time demanding than the first one.

The use of the existing information and knowledge is performed in two ways. First method is aimed at getting new information by searching the exiting knowledge repositories. It is so-called knowledge discovery from literature. This approach usually leads to radical innovations. The second method is based on the assumption that the similar problems have the similar solutions. It is a basis of case-based reasoning. It usually leads to incremental innovations.

The objective of this book is to bridge a gap between the huge amount of available knowledge and its very small subset which is not only generated and stored but also actively used. The book is a sort of guide in a store where

knowledge is stocked up and we are invited to look for the pieces which could be useful for us in solving new problems. The authors have penetrated only a very small fragment of this huge warehouse – a room in which some elements of knowledge related to chemical and process engineering have been left.

This book is about knowledge re-use by applying of case-based reasoning to the problems typical in chemical product and process design. It is composed of three parts: description of the product and process design and decision support methods related to it, presentation of case-based design principles, issues related to adaptation of the retrieved solutions and case-based reasoning environment and finally examples of application of case-based reasoning to product and process design. The application part covers the broad spectrum of examples dealing with products formulation, synthesis of the system of processing units and mathematical models re-use.

The authors would like to thank many people for the valuable discussions, comments and advice. We are not able to mention all of them but we are particularly grateful to Dr. Tivodar Farkas and Dr. Christan Botar-Jid. We highly appreciate Professor Janusz Kacprzyk for his encouragement and constant support during the preparation of this book.

We hope that this book will contribute to a broader use of case-based design in engineering practice.

Yuri Avramenko
Andrzej Kraslawski
Lappeenranta, May 2007

2

Decision Support in Design

Design is a problem solving activity. Decision making during the design activity deals with highly complex situations. Decision-making methods can be applied as techniques that are able to assist the designer in the design process.

2.1 Decision Making Process

Making a decision implies that there are alternative choices to be considered, and in such a case the goal is not only to identify as many of these alternatives as possible but to choose the one that best fits with specified objectives (Harris, 1998).

For most familiar everyday problems, decisions based on intuition can produce acceptable results because they involve few objectives and only one or two decision-makers. In the engineering environment, problems are more complex. Most decisions involve multiple objectives, several decision-makers, and are subject to external review. The specific methods for decision support are the key aspect in design practice.

A general decision making process can be divided into the following steps:

1. Problem definition
2. Requirements identification
3. Goal establishment
4. Evaluation criteria development

The process may return to a previous step from any point in the process when new information is discovered. Thus, this repeats most of model of design process, which is virtually the decision-making process.

2.1.1 Definition of the Problem

Problem definition is the crucial first step in making a good decision. This process must, as a minimum, identify root causes, limiting assumptions, system and organizational boundaries and interfaces, and any stakeholder issues.

The goal is to express the issue in a clear *problem statement* that describes both the initial conditions and the desired conditions. The problem statement must however be a concise and unambiguous material. It is essential that the decision-maker ensures what problem is going to be solved before proceeding to the next steps.

Result: Problem statement – functions, initial conditions, desired state etc.

2.1.2 Identification of Requirements

Requirements are conditions that any acceptable solution to the problem must meet. Requirements represent what the solution to the problem must do. For example, a requirement might be that a process must produce at least ten units per day. Any alternatives that produced only nine units per day would be discarded. Requirements that do not discriminate between alternatives need not be used at this time.

In mathematical form, these requirements are the constraints describing the set of the feasible (admissible) solutions of the decision problem. It is very important that even if subjective or judgmental evaluations may occur in the following steps, the requirements must be stated in exact quantitative form, i.e. for any possible solution it has to be decided unambiguously whether it meets the requirements or not.

Result: List of absolute requirements.

2.1.3 Establishment of Goals

Goals are broad statements of intent and desirable programmatic values. Examples might be: reduce worker radiological exposure, lower costs, lower public risk, etc. Goals go beyond the minimum essential requirements to wants and desires. Goals should be stated positively (i.e. what something should do, not what it should not do). In mathematical form, the goals are objectives contrary to the requirements that are constraints. Because goals are useful in identifying superior alternatives (i.e. define in more detail the desired state of the problem), they are developed prior to alternative identification.

The goals may be conflicting but this is a natural concomitant of practical decision situations. During goal definition, it is not necessary to eliminate conflict among goals nor to define the relative importance of the goals. The process of establishing goals may suggest new or revised requirements or requirements that should be converted to goals. In any case, understanding the requirements and goals is important to defining alternatives.

Result: List of clearly formulated goals.

2.1.4 Generation of Alternatives

Alternatives offer different approaches for changing the initial condition into the desired condition. Be it an existing one or only constructed in mind,

any alternative must meet the requirements. The decision team evaluates the requirements and goals and suggests alternatives that will meet the requirements and satisfy as many goals as possible. If the number of the possible alternatives is finite then it is possible to check one by one for meeting the requirements. The alternatives vary in their ability to meet the requirements and goals. If an alternative does not meet the requirements, three actions are possible:

(1) The alternative is discarded.
(2) The requirement is changed or eliminated.
(3) The requirement is restated as a goal.

The infeasible alternatives must be deleted from the further consideration, and the explicit list of the alternatives is generated. If the number of the possible alternatives is infinite, the set of alternatives is considered as the set of the solutions fulfilling the constraints in the mathematical form of the requirements.

The description of each alternative must clearly show how it solves the defined problem and how it differs from the other alternatives. A description and a diagram of the specific functions performed to solve the problem will prove useful.

Result: list of potential alternative solutions.

2.1.5 Determination of Criteria

Decision criteria, which will discriminate among alternatives, must be based on the goals. It is necessary to define discriminating criteria to measure how well each alternative achieves the goals. Since the goals will be represented in the form of criteria, every goal must generate at least one criterion but complex goals may be represented only by several criteria. If a goal does not suggest a criterion, it should be abandoned. Each criterion should measure something important, and not depend on another criterion. Criteria must discriminate among alternatives in a meaningful way.

It can be helpful to group together criteria into a series of sets that relate to separate and distinguishable components of the overall objective for the decision. This is particularly helpful if the emerging decision structure contains a relatively large number of criteria. Grouping criteria can help the process of checking whether the set of criteria selected is appropriate to the problem, can ease the process of calculating criteria weights in some methods, and can facilitate the emergence of higher level views of the issues. It is a usual way to arrange the groups of criteria, sub-criteria, and sub-sub-criteria in a tree-structure.

According to Baker et al. (2002), criteria should be

– Able to compare the performance of the alternatives.
– Complete to include all goals.

– Operational and meaningful.
– Non-redundant.
– Few in number.

Usually no one alternative will be the best for all goals, requiring alternatives to be compared with each other. The best alternative will be the one that most nearly achieves the goals.

Result: List of criteria representing the goals; collected criteria data for each alternative.

2.1.6 Evaluation of Alternatives Against Criteria

Alternatives can be evaluated with quantitative methods, qualitative methods, or any combination. Criteria can be weighted and used to rank the alternatives. Both sensitivity and uncertainty analyses can be used to improve the quality of the selection process. Experienced analysts can provide the necessary thorough understanding of the mechanics of the chosen decision-making method.

Every correct method for decision support needs, as input data, the evaluation of the alternatives against the criteria. Depending on the criterion, the assessment may be objective, with respect to some commonly shared and understood scale of measurement (e.g. money) or can be subjective (judgmental), reflecting the subjective assessment of the evaluator. After the evaluations the selected decision making tool can be applied to rank the alternatives or to choose a subset of the most promising alternatives.

Result: list of alternatives with defined measures of effectiveness.

2.1.7 Validation of Solution

After the evaluation process has selected a preferred alternative, the solution should be validated to ensure that it is able to solve the problem identified. It may happen that the decision making tool was misapplied. The comparison of the original problem statement to the goals and requirements is performed. A final solution should fulfill the desired state, meet requirements, and best achieve the goals. In complex problems the selected alternatives may also require for further goals or requirements modification and addition them to the decision model.

Once the preferred alternative has been validated, it can be presented as the final decision. A final result could report the decision process, assumptions, methods, and conclusions recommending the final solution.

2.2 Decision Support Methods

Decision support techniques are rational processes/systematic procedures for applying critical thinking to information, data, and experience in order to

make a balanced decision when the choice between alternatives is unclear. They provide organized ways of applying critical thinking skills developed around accumulating answers to questions about the problem. Steps include clarifying purpose, evaluating alternatives, assessing risks and benefits, and making a decision. These steps usually involve *scoring* criteria and alternatives. This scoring (a systematic method for handling and communicating information) provides a common language and approach that removes decision making from the realm of personal preference or idiosyncratic behavior.

Depending on type of information used and way of achieving result (decision-making) the design supporting methods can be distinguished on three major approaches: Algorithmic, Knowledge-based inductive reasoning, and Case-based reasoning. First approach relies on specific procedure (algorithm, model) that transforms input to certain output; second method deals with generalised domain knowledge to make a decision; third one considers exemplary knowledge of designs.

2.2.1 Algorithmic Approach

The algorithmic design approach views the design process as the execution of an effective domain-specific procedure that yields a satisfying design solution in a finite number of steps. The main premise of this approach is that the initial requirements are well-defined and there are precisely defined criteria for determining whether or not an algorithm meets the requirements.

There exist a number of techniques which serve to optimize complex systems: exhaustive search, rapid search, mathematical programming. The search techniques involve many search strategies, such as breath-first, greedy methods, branch and bounds, dynamic programming and so on (Siddal, 1982; Dasgupta, 1989; Chandrasekaran, 1990). An exhaustive search generates an enormous number of alternatives to be considered, therefore the application of such techniques is limited. Search algorithms are judged on the basic of completeness, optimality, time complexity and space complexity. Complexity depends on the branching factor in the state space, and the depth of the shallowest solution. The alternative to an exhaustive search is rapid search, where a set of simple but arbitrary guidelines are adopted to limit the search space. The greatest disadvantage of any rapid search method is that the best solution might be out of the search space.

Mathematical programming techniques can be used to identify the potential design configuration based on the functional requirements. In general, in these methods the solution to the problem is developed by solving a mathematical model consisting of an objective function that is to be optimized and a set of constraints representing the limitation of the resources (Siddall, 1982; Braha and Maimon, 1998; Gani, 2004).

In chemical engineering design mathematical programming techniques are widely used. One of the targets in any industrial process design is to maximize the process-to-process heat recovery and to minimize the utility (energy)

requirements. This goal can be achieved by utilizing Pinch Technology. This technique presents a simple methodology for systematically analysing chemical processes and the surrounding utility systems with the help of the First and Second Laws of Thermodynamics. Pinch Analysis is used to identify energy cost and heat exchanger network (HEN) capital cost targets for a process and recognizing the pinch point (Townsend and Linnhoff, 1983).

Another method utilised for process synthesis is the superstructure generation with following optimization (Grossmann, 1985). The advantage of the approach is the rigorous analysis of features such as structure interactions and capital costs. The disadvantage of the method is the need for a big computational efforts and the fact that the optimality of the solution can only be guaranteed among alternatives considered a priori.

An incomplete, ill-structured design problem may be decomposed into one or more well-structured components, and then the algorithmic methods may be successfully utilized to solve each of these well-structured sub-problems.

2.2.2 Knowledge-Based Inductive Reasoning Approach

This approach to decision support is based on capturing knowledge of a certain domain and using it to solve problems. The design is considered as a problem-solving process of searching through a state-space, from initial problem state to the goal state. Transition from one step to another is affected by applying one of a finite set of operators, based on functional requirements and design constraints (domain specific knowledge) and meta-rules (domain independent knowledge).

Due to emphasis of knowledge, such computer systems are known as knowledge-based or expert systems. The term 'expert system' is often used as the input knowledge is usually acquired from human experts. When knowledge is generally acquired through non-human intervention (computer methods), the term 'knowledge-based system (KBS) is more appropriate. The united term 'knowledge-based expert system' (KBES) is further used to represent both or combined methods of knowledge acquisition.

KBES is able to use previously defined rules to solve a new problem. Inductive reasoning, implemented in KBES, means reaching conclusions about a whole class of facts based on evidence on part of that class. KBESs are examples of automatic problem-solvers that rely on domain-specific heuristics.

Such reasoning differs from algorithmic approach with following issues:

– Simulation of human reasoning about a problem domain, rather than modelling the domain itself;
– Reasoning over representation of human knowledge, in additional to doing numerical calculation or data retrieval;
– Suggesting a solution to a problem using heuristic or approximate methods which, unlike to algorithmic solution, are not guaranteed to succeed;
– Capability to explain and justify solutions or recommendations to convince that the reasoning result is correct.

Algorithmic approach is the reasoning strategy which is guaranteed to find the solution to whatever the problem is, if there is such a solution. For the large, difficult problems with which expert systems are concerned, it may be more useful to employ heuristics: strategies that often lead to the correct solution, but which also sometimes fail. Humans use heuristics in their problem solving. If the heuristic does fail, it is necessary for the problem solver to either pick another heuristic, or know that it is appropriate to give up. In design problems, there may be many millions of possible solutions to the problem as presented. It is not possible to consider each one in turn, to find the right (or best) solution; heuristically-guided search is required.

Some rules used for inductive reasoning in KBES may only express a probability that a conclusion follows from certain premises, rather than a certainty. The items in the knowledge base must reflect this uncertainty, and the inference engine must process the uncertainties to give conclusions that are accompanied by likelihood that they are true. Assumptions – for instance, about the reliability of a piece of evidence – may have to be abandoned part way through the reasoning process.

Expert systems usually contain inference engine, knowledge base and two interfaces to communicate with user and experts (Fig. 2.1). Knowledge based system instead of expert interface includes knowledge generation part.

The inference engine is responsible for extracting appropriate rules from knowledge base and generating new information. There are two main ways for inference: forward chaining and backward chaining. The forward chaining is used for problem-solving when data obtained from communication with the user are the starting point. The system attempts to achieve conclusions. A problem with forward chaining is that many goals are possible to achieve whether useful or not. In contrast, backward chaining, often described as
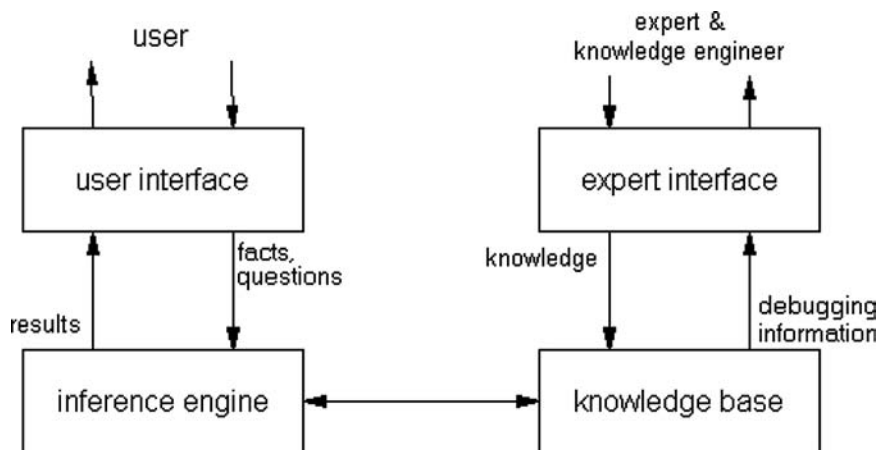

Fig. 2.1. Expert system layout

goal-directed reasoning, starts with a hypothesis or specific goal and then attempts to find data from interaction with the use to prove or disprove the conclusion. Whereas the forward chaining is often used in KBES developed for design problems, backward chaining is specifically applicable to troubleshooting and control problems. These methods of inference can often be combined in KBES.

An inference engine may also have the capability to reason in the presence of uncertainty both in the input data and also in the knowledge base. The major methods are Bayesian probabilities and fuzzy logic.

KBES approach is the base for many of the computer-aided design systems developed in recent years (Tong and Sriram, 1992; Wilke et al., 1998; Nakayama and Tanaka, 1999). A review of knowledge-based methods for design tasks in chemical engineering has recently been presented by Li and Kraslawski et al. (2004).

The knowledge-based inductive reasoning approach is very useful for solving tightly coupled, highly integrated design problems. However, when faced with an original design problem with no previous rules to help it, expert systems are incapable of original creativity.

Knowledge-based expert systems are based upon an explicit model of the knowledge required to solve a problem – so called second generation systems (Clancey, 1985) using a deep causal model that enables a system to reason using first principles. But whether the knowledge is shallow or deep an explicit model of the domain must still be elicited and implemented often in the form of rules or perhaps more recently as object models. The tight problem of KBES in many sectors is knowledge acquisition, often being referred to as the knowledge elicitation bottleneck. To overcome this difficulty special information techniques can be applied. The knowledge can be collected with decision tree generated by various algorithms. Despite obvious advantages automotive generation of knowledge base (decision tree) has several difficulties:

– Only classification problems can be addressed.
– Human interventions are still required to define attributes and original knowledge matrix.
– When new examples become available it is necessary to rebuild the existing tree.

An expert system is purposed to perform at a human expert level in a narrow, specialised domain. Thus, the most important characteristic of KBES is its high-quality performance. A unique feature of an expert system is its explanation capability. This enables the KBES to review its own reasoning and explain its decisions. An explanation in expert system in effect traces the rules fired during a problem-solving session.

KBES employs symbolic reasoning when solving a problem. Symbols are used to represent different types of knowledge. Algorithmic approach always performs the same operations in the same order, and it always provide an exact solution (if it is principally possible). Unlike algorithmic approach, KBES do

not follow a prescribed sequence of steps. It permits inexact reasoning and can deal with incomplete, uncertain and fuzzy data.

2.2.3 Case-Based Reasoning Approach

Case-based problem solving is based on the premise that a design problem solver makes use of experiences (cases) in solving new problems instead of solving every new problem from scratch (Kolonder, 1993). Lansdown (1987) argues that "innovation arises from incremental modification of existing ideas rather than entirely new approaches". Coyne et al. (1990) classify the case-based approach into three activities: creation, modification, and adaptation. Creation is concerned with incorporating requirements to create a new prototype. Modification is concerned with developing a working design from a particular category of cases. Adaptation is concerned with extending the boundaries of the class of the cases.

Case-based reasoning (CBR) solves new problems by adapting previously successful solutions to similar problems. It has several features, which make this approach different from KBES, namely:

– CBR does not require an explicit domain model and elicitation becomes a task of gathering case histories.
– Implementation is reduced to identifying significant features that describe a case, an easier task than creating an explicit model.
– Largely volumes of information can be managed.
– CBR systems can learn by acquiring new knowledge as cases thus making maintenance easier.

A case-based reasoning approach can handle incomplete data: it is robust with respect to unknown values because it does not generalize the data. Instead, the approach supports decision making relying on particular experiences.

2.3 Knowledge Engineering

The described above approaches to decision support in design deal with knowledge of certain organization. Different approaches have different knowledge organizations. However, the process of acquisition, structuring and representation of knowledge precedes any reasoning activity and it can be regarded as common for all approaches. This process is known as knowledge engineering.

There are two main views to knowledge engineering. The traditional view is known as "Transfer View". In this view, the assumption is to apply conventional knowledge engineering techniques to transfer human knowledge into artificial intelligent systems. The alternative view is known as the "Modeling View". In this view, the knowledge engineer attempts to model the knowledge

and problem solving techniques of the domain expert into the artificial intelligent system.

Knowledge engineering relates to the building, maintaining and development of knowledge-based systems. It has a great deal in common with software engineering, and is related to many computer science domains such as artificial intelligence, databases, data mining, expert systems, and decision support systems.

Various activities of KE specific for the development of a knowledge-based system:

(1) Assessment of the problem
(2) Development of a knowledge structure
(3) Implementation of the structured knowledge into knowledge-bases
(4) Acquisition and structuring of the related information, knowledge and specific preferences
(5) Testing and validation of the inserted knowledge
(6) Integration and maintenance of the system
(7) Revision and evaluation of the system.

KE deals with the knowledge, and mainly with the structure (organization) of knowledge. Therefore, the organisation of knowledge is a key element of KE.

2.3.1 Classification of Knowledge

Initial source of knowledge base is a set of data. Data refers to facts, codes, marks and signals. Data is transformed by processing to information which is organized to be meaningful to the object receiving it. Knowledge can therefore be regarded as information which is understood and can be applied to get new information (Fig. 2.2).

Knowledge can be derived from other knowledge. Priori perceived knowledge can be transcribed to five primary types of content: facts, concepts, processes, procedures, and principles (Clark and Chopeta, 2004).

Facts are specific and unique data or instance. Concept is a class of items, words, or ideas. There are two types of concepts: concrete and abstract. Process is represented by a flow of events or activities that describe how things work rather than how to do things. There are normally two types: business processes that describe work flows and technical processes that describe how things work in equipment or nature. Procedures are series of step-by-step actions and decisions that result in the achievement of a task. There are two types of actions: linear and branched. Guidelines and rules form principles. It includes not only what should be done, but also what should not be done. Principles allow one to make predictions and draw implications. Given an effect, one can infer the cause of phenomena. Principles are the basic building blocks of causal models or theoretical models (theories).
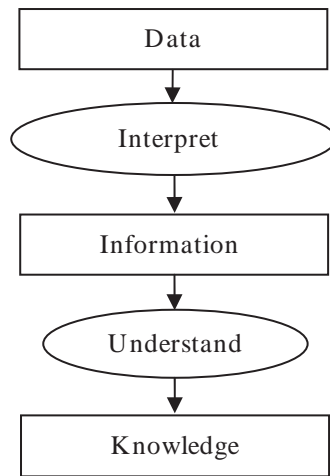
Fig. 2.2. Transformation data to knowledge

These contents can be used to create two categories of knowledge: declarative and procedural, where the first comprises concepts and the second are actions.

Declarative Knowledge

Declarative knowledge refers to representations of objects and events and how these knowledge and events are related to other objects and events. They focus on the why rather than the how. Declarative models include propositions and schemata. Proposition consists of a predicate or relationship and at least one argument. Schemata are higher-level cognitive units that use propositional networks as their building blocks. These are often abstract or general in nature that allows to classify objects or events as belonging to a particular class and to reason about them. Schemata can be conceptional knowledge, plan-like knowledge, and causal knowledge.

Concepts are simple schemata that represent a class of objects, events, or other entities by their characteristic features. Concepts enable a person to identify or classify particular instances (concrete object or event) as belonging to a particular class. In a language, most words identify concepts and at least to a certain degree, they are arbitrary in that they can be categorized in many alternative ways.

Experts possess more powerful concepts in their domain than novices that help them to solve problems. These concepts give them patterns for labeling various memory states, which allow them to classify problems according to their solution mode or deep structure. Where as novices typically classify problems according to their surface structure or superficial feature.

Plan-Like Knowledge is simple schemata that describe how goals are related in time or space. They allow us to understand events and organize functions and actions. Plans are often referred to as scripts (or simple procedures) because they represent routine sequences of events.

Causal Knowledge is complex schemata that link principles and concepts with each other to form cause–effect relationships. They are able to interpret events, give explanations, and make predictions.

Procedural Knowledge

Procedural models focus on tasks that must be performed to reach a particular objective or goal. It is characterized as knowing how. Procedural knowledge is often difficult to verbalize and articulate (tacit knowledge) than declarative knowledge.

Procedural knowledge emphasizes hierarchical or information processing approaches based upon productions. A combination of productions creates production systems. Productions are the building blocks of procedural knowledge and are composed of a condition and an action or IF and THEN statement. A production system is a set of productions for cognitive processing. It is characterized by the recognize-act cycle in which one production leads to another production. There are two types of productions: rules and heuristics. The difference between rules and heuristics is based on the validity and rigour of the arguments used to justify them – rules are always true, valid and can be justified by arguments; heuristics are the expert's best judgments, may not be valid in all cases and can only be justified by examples.

There also can be distinguished the specific class of knowledge, which stays above of previous declared categories of knowledge, called meta-knowledge.

Meta-Knowledge

Meta-knowledge is knowledge about knowledge. More precisely speaking, meta-knowledge is systemic problem and domain-independent knowledge which performs or enables operations on another more or less specific domain-dependent knowledge in different domains/areas of human activities. Meta-knowledge is a fundamental conceptual instrument in such research and scientific domains as, knowledge engineering, knowledge management, and others dealing with study and operations on knowledge, seen as an unified object/entities, abstracted from local conceptualizations and terminologies.

Examples of the first-level individual meta-knowledge are methods of planning, modeling, learning and every modification of a domain knowledge. The procedures, methodologies and strategies of teaching, coordination of e-learning courses are individual meta-meta-knowledge of an intelligent entity (a person, organization or society). The universal meta-knowledge frameworks have to be valid for the organization of meta-levels of individual meta-knowledge.

Knowledge can be classified according to the origin of the knowledge. The source of *empirical* knowledge is practical experience. Observations are made when running the process. *Theoretical* knowledge is based on natural laws and scientific theories. The third form of knowledge is subjective, *experience-based* knowledge.

When describing certain domain, general and problem independent knowledge is called *background* knowledge. If the background knowledge describes a specific part of the domain it is called *contextual* knowledge. *Episodic* knowledge is of narrative character. It records the story of something happened in the past.

There are two levels of knowledge: shallow or deep knowledge. *Shallow* knowledge can deal with very specific situations, whereas the *deep* knowledge is a representation of all information of a domain.

2.3.2 Knowledge Acquisition

The objective of knowledge acquisition is to collect or elicit knowledge from the experts and other sources and structure it in a certain way.

The first step of knowledge acquisition is to collect all the potential sources of knowledge. They are text book written specifically in the domain, research and technical reports, journal articles, reference manuals, case studies, operational procedures and organizational policy statements. Availability of documents may vary; in some domain there may be many available, and in others none at all. The reports and books contain factual knowledge; they are often detailed, precise and well structured but are not always relevant to knowledge acquisition task. Often, the analysis of significant amount of documents is highly time-consuming. The range of problems which textbooks examine and solve is always smaller than the range of problems that a human expert is master of.

Knowledge can also be obtained from discussion with organization personnel like projects leader and consultants. The most important branch of knowledge acquisition is knowledge elicitation – obtaining knowledge from domain experts.

Expert knowledge includes:

 – Domain-related facts and principles
 – Modes of reasoning
 – Reasoning strategies
 – Explanations

Two kinds of knowledge can be elicited from experts:

 – Explicit knowledge is the knowledge which the domain expert is able to articulate.
 – Tacit knowledge is the knowledge which the domain expert is not conscious of having but does exists as proved by expert's known capability of solving problems in the domain.

Explicit knowledge is easy to elicit from experts since it is mainly factual in nature. Tacit knowledge is difficult to identify and elicit but it is essential for successful development of knowledge-based systems. Knowledge obtained from experts have following features: incomplete – experts may forgot, superficial – exerts often cannot go to details, imprecise – experts may not know exact detail, inconsistent – when expert fall into contradictions, incorrect – when experts may be wrong.

Such features could rise a lot of problems in creation of knowledge base. Needs in communication with experts as well as in retrieval data from various documents exists more or less in all approached for decision making support. But the acquired knowledge have to be interpreted and translated into the rules and heuristics in the KBES approach, which is also time-demanding task. In contrast to, CBR approach relies only on set of acquired information (even not knowledge in many cases).

In addition to manual methods of knowledge acquisition there are automated methods whereby the computers are used. Using a computer for a knowledge acquisition overlaps with software engineering problems.

2.3.3 Software Engineering versus Knowledge Engineering

Software engineering provides the mechanisms for validating the implementation of well specified algorithms. Human–computer interaction provides analysis and design techniques based on prototyping of the user interface to address aspects of systems where the risks are associated with the users' needs, or the system usability. Data engineering addresses the permanent storage of large amounts of data and the efficient retrieval of the relatively small portion required for any process. In contrast, knowledge engineering addresses the structure of complex but ill-defined processes where the solution to defining the process is to define the knowledge involved in the process explicitly in a knowledge-based system (KBS).

Conventional software development follows the waterfall life cycle model. This requires complete system requirements at the start of development. Errors later in development can be fixed at little cost; errors at the start of development incur large costs. If the risks of failure of the project are associated with the efficiency of the implementation of a system this is appropriate. If the risk of a project failing is due to the uncertainty of the algorithms to perform the functions required, user requirements or enterprise objectives then an approach which is flexible at the start of the process is appropriate.

Conventional software engineering approaches produce efficiently implemented code to execute algorithms to perform required functions which will always produce the correct outcome for correct input. The knowledge engineering approach allows users and experts to describe requirements and methods to perform the required functions at a high level close to the one in which they think about the task: the Knowledge Level. These can then be presented back to them for validation of the content, and modification.

If algorithms to perform the required functions cannot be determined then heuristics which produce correct outcomes sufficiently often for some task requirements can be used – there may not be sufficiently detailed domain theory to supply algorithms so human expertise in the domain can be used.

If heuristic knowledge cannot be acquired which produces correct outcomes sufficiently frequently then the project should be terminated – there may not be domain expertise to acquire. Since this possibility continues after initial problem definition (including feasibility studies) into the acquisition of knowledge, then staged contracting should be used to protect the client, and the commitments made by the developer.

Knowledge engineering differs from conventional software engineering mainly at the early stages of the life cycle when user requirements and functional methods (or knowledge) are being acquired. The tools for implementation, user interface design, testing, maintenance and updating systems may differ, but the principles which govern all software systems are the same. Therefore, although the early stages of knowledge acquisition will involve a knowledge engineer and a (or more) domain experts, later stages will involve software engineers for implementation/integration.

2.3.4 Knowledge Representation

Knowledge representation (KR) is the study of how knowledge about the world can be represented and what kinds of reasoning can be done with that knowledge. Important questions include the tradeoffs between representational adequacy, fidelity, and computational cost, how to make plans and construct explanations in dynamic environments, and how best to represent default and probabilistic information

A variety of ways of representing knowledge in a knowledge base have been developed over the years.

The commonly used methods for knowledge representation are production rules, frames, semantic networks, ontology and objects.

Production Rules

They express the relationship between several pieces of information. The rules are conditional statements that specify actions to be taken or advice to be followed under certain sets of conditions.

Each production rule implements an autonomous piece of knowledge and can be developed and modified independently of other rules. However, when combined, a set of rules may yield better results that the sum of results of the individual rules and independency is lost. It must be taken into account when adding new rules to a current knowledge base to avoid conflicts.

Frames

They are templates for holding clusters of related knowledge about a particular object. They are able to represent the attribute of an object in a more descriptive way that is possible using production rules. The frame typically consists of a number of slots which, like attributes, may or not contain a value.

Semantic Network

Because any knowledge incorporates concepts and will be expressed using terms, the interdependencies between knowledge and language are essential for the definition itself.

A semantic network is a directed graph consisting of vertices, which represent concepts, and edges, which represent semantic relations between the concepts. Such networks involve fairly loose semantic associations that are nonetheless useful for human browsing. It is possible to represent logical descriptions using semantic networks such as the existential graphs or the related conceptual graphs. These have expressive power equal to or exceeding standard first-order predicate logic. The semantic networks can be used for reliable automated logical deduction. Some automated reasoners exploit the graph-theoretic features of the networks during processing.

One can consider a mind map to be a very free form variant of a semantic network. By using colors and pictures the emphasis is on generating a semantic net which evokes human creativity. However, a fairly major difference between mind maps and semantic networks is that the structure of a mind map, with nodes propagating from a centre and sub-nodes propagating from nodes, is hierarchical, whereas semantic networks, where any node can be connected to any node, have a more heterarchical structure.

Ontology

An ontology is a knowledge model that represents a set of concepts within a domain and the relationships between those concepts. The word ontology means "the study of the state of being". An ontology describes the states of being of a particular set of things. This description is usually made up of axioms that define each thing. It is used to reason about the objects within that domain.

Ontologies generally describe:

– *Individuals*: the basic objects
– *Classes*: sets, collections, or types of objects
– *Attributes*: properties, features, characteristics, or parameters that objects can have and share
– *Relations*: ways that objects can be related to one another
– *Events*: the changing of attributes or relations

The individuals in an ontology may include concrete objects such as tables, automobiles, molecules, and reactor, as well as abstract individuals such as numbers and words. Actually, an ontology need not include any individuals, but one of the general purposes of an ontology is to provide a means of classifying individuals, even if those individuals are not explicitly part of the ontology.

Classes may contain individuals, other classes, or a combination of both. Ontologies vary on whether classes can contain other classes, whether a class can belong to itself, whether there is a universal class (that is, a class containing everything), etc. The classes of an ontology may be extensional or intensional in nature. A class is extensional if and only if it is characterized solely by its membership. If a class does not satisfy this condition, then it is intensional. While extensional classes are more well-behaved and well-understood mathematically, they do not permit the fine grained distinctions that ontologies often need to make.

A partition is a set of related classes and associated rules that allow objects to be placed into the appropriate class. If the partition rules guarantee that an object cannot be in both classes, then the partition is called a *disjoint* partition. If the partition rules ensure that every concrete object in the superclass is an instance of at least one of the partition classes, then the partition is called an *exhaustive* partition.

Objects in the ontology can be described by assigning attributes to them. Each attribute has at least a name and a value, and is used to store information that is specific to the object it is attached to. The value of an attribute can be a complex data type.

An important use of attributes is to describe the relationships between objects in the ontology. Typically a relation is an attribute whose value is another object in the ontology. The most important type of relation is the subsumption relation (knows as *is-a*). This defines which objects are members of classes of objects.

The addition of the *is-a* relationships has created a hierarchical taxonomy; a tree-like structure that clearly depicts how objects relate to one another. Another common type of relations is the Meronymy relation (written as *part-of*) that represents how objects combine together to form composite objects. The examples of described relation types are represented in Fig. 2.3.

As well as the standard is-a and part-of relations, ontologies often include additional types of relation that further refine the semantics they model. These relations are often domain-specific and are used to answer particular types of question.

Knowledge Representation Languages and Ontology Analysis

One of the developments in the application of KR has been the proposal (Minsky, 1981) and development (Brachman and Schmolze, 1985) of frame-based KR languages. While frame-based KR languages differ in varying
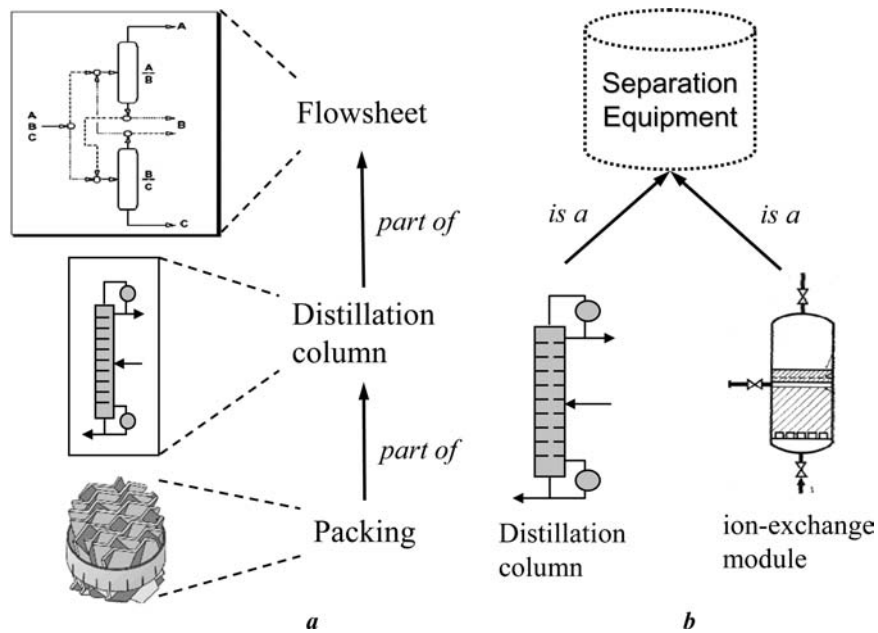
Fig. 2.3. Meronymy (**a**) and subsumption (**b**) relations examples

degrees from each other, the central tenet of these systems is a notation based on the specification of objects (concepts) and their relationships to each other. The main features of such a language are:

- Object-orientedness. All the information about a specific concept is stored with that concept, as opposed, for example, to rule-based systems where information about one concept may be scattered throughout the rule base.
- Generalization/Specialization. Long recognized as a key aspect of human cognition (Minsky, 1981), KR languages provide a natural way to group concepts in hierarchies in which higher level concepts represent more general, shared attributes of the concepts below.
- Reasoning. The ability to state in a formal way that the existence of some piece of knowledge implies the existence of some other, previously unknown piece of knowledge, is important to KR. Each KR language provides a different approach to reasoning.
- Classification. Given an abstract description of a concept, most KR languages provide the ability to determine if a concept fits that description, this is actually a common special form of reasoning.

Object orientation and generalization help to make the represented knowledge more understandable to humans, reasoning and classification help make a system behave as if it knows what is represented.

It is important to realize both the capabilities and limitations of frame-based representations, especially as compared to other formalisms. To begin with, all symbolic KR techniques are derived in one way or another from First Order Logic, and as a result are suited for representing knowledge that does not change. Different KR systems may be able to deal with non-monotonic changes in the knowledge being represented, but the basic assumption has been that change, if present, is the exception rather than the rule.

Two other major declarative KR formalisms are production systems and database systems. Production systems allow for the simple and natural expression of IF-THEN rules. However, these systems have been shown to be quite restrictive when applied to large problems, as there is no ordering of the rules, and inferences cannot be constrained away from those dealing only with the objects of interest. Production systems are subsumed by frame-based systems, which additionally provide natural inference capabilities like classification and inheritance, as well as knowledge-structuring techniques such as generalization and object orientation.

Database systems provide only for the representation of simple assertions, without inference. Rules of inference are important pieces of knowledge about a domain.

What makes up a specific domain ontology is restricted by the representational capabilities of the meta-model – the language used to construct the model. Each knowledge representation language differs in its manner and range of expression. In general, an ontology consists of three parts: concept definitions, role definitions, and further inference definitions.

The concept definitions set up all the types of objects in the domain. In object oriented terms this is called the class definitions, and in database terms these are the entities. There can be three parts to the concept definitions: concept taxonomy, role defaults and role restrictions. The taxonomy is common to most knowledge representation languages, and through it is specified the nature of the categories in terms of generalization and specialization. Role defaults specify for each concept what the default values are for any attributes. Role restrictions determine for a concept any constraints on the values in a role, such as what types the values must be, how many values there can be, etc.

A role is an attribute of an object. In object-oriented terms it is a slot, in database terms (and even some KR languages) it is a relation. Roles which represent relationships are unidirectional. A role definition may have up to three parts as well: the role taxonomy which specifies the generalization/specialization relationship between roles; the role inverses which provide a form of inference that allows the addition of a role in the opposite direction when the forward link is made; and the role restrictions where the role itself may be defined such that it can only appear between objects of certain types (domain/range restrictions), or can only appear a specified number of times (cardinality restriction).

The final part of an ontology is the specification of additional inference that the language provides. Examples of this are forward and/or backward chaining rules, path grammars, subsumption and/or classification, demons, etc.

Knowledge Engineering must address the issue of reliable methodology to meet the practical engineering objectives it now has. Secondly, the systems produced through knowledge engineering methods must be able to re-use not only abstract ideas, but also implementation level knowledge. To do these issues of portability and interoperability must be addressed. A consequence of addressing these two issues could be to lose the apparent freedom provided by expert systems and to become bound by the formalities of software engineering. To avoid this, knowledge engineering must maintain its influence on user interfaces and the ability of KBS to explain their reasoning.

2.4 Decision Supporting Systems

Decision making in design often requires access to and the processing of a large amount of data and logical relations which (due to the nature of the problem) cannot or should not be replaced by the intuition of decision maker. In many design situations it is not a small task to examine even the possible range of feasible alternatives. In the context of decision support, the problem is a situation description in which information is missing. The goal is to complete the situation description until the demand for information is satisfied. The use of computers for processing situations leads to implementing a Decision Supporting System.

A Decision Supporting System (DSS) is a supportive tool for the management and the processing of large amounts of information and logical relations that helps a decision maker (design engineer) to extend his vision and thus help to reach a better decision. In other words, a DSS can be considered as a tool that performs the task of data processing and provides relevant information that enables a design engineer to concentrate on the part of the decision making process that cannot be formalized.

Because there are many approaches to decision-making and because of the wide range of domains in which decisions are made, the concept of DSS is very broad. A DSS can take many different forms. In general, a DSS is an information system that provides the ability to analyze information and predict the impact of decisions before they are made. A decision is a choice between alternatives based on estimates of the values of those alternatives. Supporting a decision means helping people working alone or in a group gather intelligence, generate alternatives and make choices. Supporting the choice making process involves supporting the estimation, the evaluation and/or the comparison of alternatives. In practice, references to DSS are usually references to computer applications that perform such a supporting role.

The goal of a DSS is to supplement the decision powers of the human with the data manipulating capabilities of the computer (Emery, 1987). It is

not intended to solve a decision problem. Therefore it should not support reaching a single or unique decision nor should it restrict a possible range of decisions.

Furthermore, it is usually not possible to decide whether a solution found by a DSS is correct or not. Rather, this information may be more or less useful; it may be better or worse than other information (Lenz et al., 1998).

Richter (1992) identified four characteristic properties of DSS:

(1) The amount of information that has to be coped with is too large to be handled by humans without the support of an appropriate technical system.
(2) The decision has to be made quickly.
(3) Data has to be prepared for decision making.
(4) The process of decision making is highly complex and requires specific algorithms.

Turban et al. (2005) composed more longer list of ideal characteristics and capabilities of DSS:

1. Support for decision makers in semistructured and unstructured problems.
2. Support managers at all levels.
3. Support individuals and groups.
4. Support for interdependent or sequential decisions.
5. Support intelligence, design, choice, and implementation.
6. Support variety of decision processes and styles.
7. DSS should be adaptable and flexible.
8. DSS should be interactive and provide ease of use.
9. Effectiveness balanced with efficiency (benefit must exceed cost).
10. Complete control by decision-makers.
11. Ease of development (modification to suit needs and changing environment).
12. Support modeling and analysis.
13. Data access.
14. Standalone, integration and Web-based.

2.4.1 Classification of DSS

There is no universally accepted classification of DSS. Different authors propose different classifications. Using the relationship with the user as the criterion, Häettenschwiler (1999) differentiates *passive*, *active*, and *cooperative* DSS. A passive DSS is a system that aids the process of decision making, but that cannot bring out explicit decision suggestions or solutions. An active DSS can bring out such decision suggestions or solutions. A cooperative DSS allows the decision maker (or its advisor) to modify, complete, or refine the decision suggestions provided by the system, before sending them back to the system for validation. The system again improves, completes, and refines

the suggestions of the decision maker and performs the validation. The whole process then starts again, until a consolidated solution is generated.

Using the mode of assistance as the criterion, Power (2002) differentiates *communication-driven* DSS, *data-driven* DSS, *document-driven* DSS, *knowledge-driven* DSS, and *model-driven* DSS.

A model-driven DSS emphasizes access to and manipulation of a statistical, financial, optimization, or simulation model. Model-driven DSS use data and parameters provided by users to assist decision makers in analyzing a situation; they are not necessarily data intensive. Dicodess is an example of an open source model-driven DSS generator (Gachet, 2004).

A communication-driven DSS supports more than one person working on a shared task; examples include integrated tools like Microsoft's NetMeeting or Groove (Stanhope, 2002).

A data-driven DSS or data-oriented DSS emphasizes access to and manipulation of a time series of internal company data and, sometimes, external data.

A document-driven DSS manages, retrieves and manipulates unstructured information in a variety of electronic formats.

A knowledge-driven DSS provides specialized problem solving expertise stored as facts, rules, procedures, or in similar structures.

Using scope as the criterion, Power (1997) differentiates *enterprise-wide* DSS and *desktop* DSS. An enterprise-wide DSS is linked to large data warehouses and serves many managers in the company. A desktop, single-user DSS is a small system that runs on an individual personal computer.

2.4.2 Architectures of DSS

Different authors identify different components in a DSS. Sprague and Carlson (1982) identify three fundamental components of DSS:

– The database management system (DBMS)
– The model-base management system (MBMS)
– The dialog generation and management system (DGMS)

Haag et al. (2006) describe these three components in more detail: the DBMS stores data, which can be further divided into that derived from the local data repositories, from external sources such as the Internet, or from the personal insights and experiences of individual users; the MBMS handles representations of events, facts, or situations using various kinds of models; and the DGMS is the component that allows a user to interact with the system.

According to Power (2002), academics and practitioners have discussed building DSS in terms of four major components: the user interface, the database, the model and analytical tools, and the DSS network.

Häettenschwiler (1999) identifies five components of DSS:

– The users with different roles or functions in the decision making process (decision maker, advisors, domain experts, system experts, data collectors)
– The specific and definable decision context

- The target system describing the majority of the preferences
- The knowledge base made of external data sources, knowledge databases, working databases, data warehouses and meta-databases, mathematical models and methods, procedures, inference and search engines, administrative programs, and reporting systems, and
- The working environment for the preparation, analysis, and documentation of decision alternatives

Marakas (1999) proposes a generalized architecture made of five distinct parts:

- The data management system
- The model management system
- The knowledge engine
- The user interface, and
- The user(s)

Holsapple and Whinston (1996) classify DSS into the following six frameworks: Text-oriented DSS, Database-oriented DSS, Spreadsheet-oriented DSS, Solver-oriented DSS, Rule-oriented DSS, and Compound DSS.

The support given by DSS can be separated into three distinct. interrelated categories (Hackathorn and Keen, 1981): Personal Support, Group Support and Organizational Support.

DSSs which perform selected cognitive decision-making functions and are based on artificial intelligence or intelligent agents technologies are called Intelligent Decision Support Systems (IDSS).

A DSS is a problem dedicated system usually designed for a specific decision making process and its environment. Using DSS is useful in complex design situations for which specification of attainable goals and rational decisions is quite complicated. The DSS finds a solution closest to the specified goals. This ability to provide answers for decision support in a changing environment is the main advantage of decision supporting systems.

There are two alternative approaches for the design of DSSs: normative and descriptive (Lenz et al., 1998). The normative approach attempts to establish general rules for rational behaviour. It is realized by utilizing a knowledge-based reasoning technique. On other hand, the descriptive approach does not rely much on general principles but on examples of successful problem solving episodes. Such episodes are investigated to obtain knowledge about how the solution was derived. This can clearly be implemented by utilizing a case-based reasoning approach.

2.5 Conclusions

In the chemical process design there is a growing demand for an improvement to the design process in order to generate better flowsheets within a shorter development time. Existing design supporting tools have been developed for

specific purposes and related to separate parts of process design. Therefore a tool or methodology that is able to support overall design activity (from A to F levels) would be very valuable.

Due to uncertain and incomplete input data and the lack of formal methods, approaches to innovative design and redesign support are proposed to assist the design engineer rather than to automate the process. Engineer intervention is required to generate or evaluate a proper solution. The problem solving process then is to provide the user of a design supporting system with documents to satisfy his demands.

Knowledge-based systems using rule-based reasoning and various algorithmic techniques have been applied to build design decision support system. Although such systems have been met with some success, difficulties have been encountered in terms of formalizing generalized design experiences as rules, logic and domain models. In order to support innovative design tasks, conventional problem solving methods are not applicable, in general. The use of experience is of particular importance. Recently, researchers have been exploring the idea of using case-based reasoning to complement or replace other approaches to design support. In order to support creative design tasks, the application of analogical problem solving is advantageous.

The idea of supporting the designer by means of case-based knowledge to help navigate through a dynamic design process seems to be promising. Moreover, a general approach which can support various stages of design activity is only possible with case-based reasoning: it relies on particular experience of design and there is no need for derivation of specific heuristics of the design process for each design stage.

Case-based reasoning (CBR) can support innovative design and redesign activity by reminding designers of previous experiences that could match with the new design situation, not necessary totally but only partially. This approach is able to support almost all steps of chemical process design, except perhaps the first and last ones (i.e. from B to E). But even for steps A and F, the sort of supporting activity can be realised. The next part describes a case-based design supporting paradigm.