Design as a Problem of Requirements Explication

Martin Dzbor[§]

Abstract. Engineering design is a human activity using different knowledge sources. From the point of availability it is possible to distinguish well-structured, explicit knowledge as opposed to tacit, implicit and often experience-based knowledge. Each type plays a particular role in engineering design, and thus in knowledge-based design support systems (KBDSS). This paper addresses several issues with KBDSS. It begins with a discussion of the processes underlying engineering design. Further, it presents a theoretical formalism for engineering design and shows the role of reflection in design. Proposed framework is further illustrated on a simple example, which is followed by some suggestions for further research.

1. INTRODUCTION

We look at the support systems for engineering design. The knowledge-centred view is emphasised in three facets describing the proposed approach: knowledge representation for design, design process control, retrieval and presentation of relevant knowledge.

The *design task* occurs when an agent decides to change the status of an artefact or surrounding world [1]. It is a goal-oriented process leading from initial objectives towards an artefact realising the change. In a real-world practice the task is ill-structured [2], i.e. its solution may not be found until significant effort to understand the problem has been made. Design can be thus considered as an intellectual and knowledge-rich activity of an agent [3].

Engineering design uses scientific principles, technical information, designers' imagination and experience to develop systems that perform desired functions [4]. In practice designers are rarely presented with a detailed specification of design problems [5]. Specifications must be built up from uncertain design situations. Real-world design typically begins with initial requirements that are vague and incomplete. Initial requirements must be transformed to consistent and complete requirements, and a solution must be found satisfying them. Experienced designers usually know how to convert an uncertain situation into a soluble design problem; experience acquired in the past helps them to tackle current problems [6].

Design consists of several activities: description of requirements, finding a solution, its evaluation, simulation, etc. All of them can be supported in one way or another. Traditional CAD focused on the support for the solution phase with the description phase being undervalued [3]. However, finding relevant requirements to a design problem is as important as finding the solution. The discovery of requirements and solutions must be seen as interactive in terms of knowledge provided [7]. This knowledge includes experience from past designs, theoretical knowledge, design guidelines etc.

In the following parts the engineering design is understood as an iterative transformation of initial incomplete requirements to a consistent formulation of design problem and its solution. Proposed (partial) solutions may help to uncover new, relevant requirements or otherwise influence the existing ones. Modified requirements trigger solution refinement, thus revealing a principle of the co-evolution of mutually complementary concepts [8]. In the paper we look at the application of knowledge-level models containing different types of knowledge in the early phases of design with emphasis on the phenomenon of co-evolution. We understand the notion 'model' in Newell's terms [9] as an entity that can explain certain actions not on the computational level but rather on a higher level of abstraction – *knowledge level*.

2. NATURE OF DESIGN

Before starting with knowledge-based support, let us mention some key features of design problems, upon which we build the remainder of this paper. As mentioned above, engineering design is a knowledge-rich activity and designers often draw on their previous experience when they tackle initial design situations. Because of the uncertainty of initial requirements, the design space is not well defined but must be developed 'on-the-fly'. Reflective processes therefore play a significant role in the understanding and consequently supporting design. A more detailed, comparative study of some selected features of design can be found in [8].

Reflection is a term introduced by Schön [5] to explain the nature of non-trivial problem solving. Design is seen as an iterative process, where one may change the current perspective (frame) when this does not suit the design situation satisfactorily. This change will cause new objects to be identified within the situation, which may further lead to another change in perspective. The need for changing the current perspective (frame) is usually caused by an unexpected result in the current perception of a problem. Schön refers to it as 'a surprise' and claims that in theory any result inconsistent with designers' theoretical and/or empirical knowledge may be perceived as 'a surprise'. When designers find something surprising (either positively or negatively) in the design based on their current perspective, they reflect on the actions made so far.

In fact, there are several reflective loops occurring during the design. Some of them are mentioned in [8]. For instance, the simplest form of reflection happens when trying to formulate design requirements so as to conform to some given rules. Another form may occur as '*a reflection on action*' and represents an influence of the proposed solution on the current set of requirements. Both forms are definitely knowledge-intensive. The former type contains a strong element of '*interpretative knowledge*' that helps to understand the 'meaning' of a particular rule. The latter one relies heavily on '*analytical knowledge*' helping to clarify details of a proposed solution. Although analysis of a design solution plays a crucial role in the reflection, it must not be confused with the much more complicated process of reflection itself. Reflection, in addition to the analysis of a solution, contains an important feature of *appreciating* the process that led to a particular solution and knowledge that was used in this process.

From the point argued in the paper, the latter form of reflection is especially interesting because it involves the co-evolution of two interacting 'worlds' – requirements and solutions. In addition to these, we should not forget the 'world of knowledge of and about the design' that also changes as the design problem evolves.

3. SUPPORT FOR REFLECTION IN DESIGN

Some definitions of design may imply that a successful innovative design is often a matter of designers' artistry when it comes to converting the initial requirements into final requirements and solutions. However, as shown e.g. in [6, 10], this 'artistry' highly depends on knowledge that is available during the design, and is affected by knowledge structure and designers' ability to draw analogies within and across different domains. Knowledge for and about design may take various forms. Starting with rather tacit knowledge [11] that is 'stored' inside designers' heads and needs to be uncovered before continuing the design process; and ending with

[§] Knowledge Media Institute, The Open University, Milton Keynes, MK7 6AA, UK; Email: M.Dzbor@open.ac.uk clearly articulated knowledge of scientific principles, structural elements, design rules, norms and so on.

Knowledge-based design support system (KBDSS) may be seen as a decision support system enabling designers to explore the structure of design problems and their solutions by *combining* human design expertise with domain and design knowledge stored in a computational system [4]. This paper uses this definition as a base, because it involves both participants in complex design tasks – a designer and a computer-based tool, where the role of a computer is to support designers so that they can spend their efforts on innovative rather then knowledge-maintaining tasks.

We suggest below that similarly as knowledge for design, knowledge bases (KB) used in design may also take two principal forms – explicit and hidden. Unlike other related research that prefer one of these forms to another [4, 12-15], we see both types of KB as equally important especially in the early phases of design when the design task itself must be formulated before a solution to it is sought. We distinguish these two types in the following way:

- Explicit knowledge in design is knowledge of *elements* that can be used to formulate both design requirements and solutions. It is also knowledge of *relations* among the elements. Elements related in a particular way may be seen as *knowledge models* describing artefacts that can be designed in a particular domain.
- The second type of knowledge in design is covered mainly by designers' *experience* from solving more or less similar design tasks in the past. Experiential knowledge may be stored in the form of *design cases* that contain (in addition to elements and relations) various relevant assumptions, their justifications, and various reasoning chains showing how the previous solution was discovered.

3.1 Assumptions and basic definitions

We assume the following points in order to develop and ground our theoretical framework; they will be discussed in-depth further:

(i) A problem can be specified in terms of a set of (possibly incomplete) *requirements* that refer to functional and other elements known in a particular domain [16].

(ii) A *solution* to a problem is developed in terms of a set of structural elements and relations among them assuring the proposed structure meets the desired functions and/or principles.

(iii) There exists '*a domain theory*' by which knowledge about elements and relations can be derived [17].

(iv) During the design process *assumptions* are made that enable the exploration of a space of possible design solutions [1].

(v) Assumptions may or may not be a part of current domain theory; however, when proved useful together with their implications they may *extend* current domain theory [6].

As the assumptions above suggest, our proposal is mainly based on the work of researchers studying different aspects of knowledge suitable for design [12, 16-19]. All cited papers contain mutually interacting concepts of *functions* and *structures* that may be used to understand a problem and develop a solution. I would like to modify their understanding of these concepts in the following way.

Let us define $\mathcal{E} = \mathcal{E}_F \cup \mathcal{E}_P \cup \mathcal{E}_S$ as a set of elements that can be used for designing artefacts in a particular domain, where the subsets stand for sets of *function*-oriented, *principle*-oriented and *structure*-oriented elements, respectively. Examples of individual types of elements include, e.g. '*electricity supply*' as a functional element, '*transformation of solar energy using photoelectric effect*' as a (physical) principle, and finally '*solar panel cell*' or '*battery*' as structural elements. Generally, elements may consist of simpler elements; thus forming complex hierarchy. Whether we use higher- or lower-level elements depends on how deeply we want to immerse in the design of an artefact. In other words, the level of specificity is influenced by designer's aims and problem requirements.

Further, let us introduce $\mathcal{L} = \{l_i\}$ as a set of possible relations (connections) among the elements, where l_i may be a constructional relation ($l_i \subset \mathcal{E}_S \times \mathcal{E}_S\,[\times\ldots]$), an elementary function directly associated with a particular structure ($l_i \subset \mathcal{E}_S \times \mathcal{E}_F$), a domain principle ($l_i \subset \mathcal{E}_P \times \mathcal{E}_F$), or most usually a combination of all three.

$\mathcal{R} = \{r_i\} = \mathcal{R}_X \cup \mathcal{R}_H$ will be a set of requirements that are set on the artefact to be designed. As mentioned at the beginning, requirements may be possibly incomplete, therefore we will talk formally about explicitly given requirements ($\mathcal{R}_X$) and hidden requirements that must be first uncovered to assess their satisfaction ($\mathcal{R}_H$). How the hidden requirements may be uncovered and made explicit is a topic of a separate section.

Let us also define $\mathcal{T} = \mathcal{T}_M \cup \mathcal{T}_C$ as a domain theory comprising of two subsets – a set of 'theoretically achievable' models and a set of previously solved cases, respectively. The difference between these two subsets will be highlighted in the following sections.

3.2 Model-based design knowledge

In this section I discuss the roles of knowledge models <u>for</u> design, which must be distinguished from a model <u>of</u> design process as shown in Conclusion. Many researchers see model for design as a kind of 'prototypic models of design solutions' [14, 20] that need to be 'filled in' in order to derive solution to a specific problem. I would like to present the term 'model' purely as an abstract representation of various elements and relations among them on multiple knowledge levels that together form rather complex 'meshes' connecting functions, principles and structures. Unlike other approaches, this understanding does not demand existence of 'rules' prescribing when a particular model can be applied. The proposed approach is opportunistic rather than prescriptive.

To understand the difference of such an interpretation from other similar works, I will refer back to the set of explicitly articulated requirements ($\mathcal{R}_X$) that are typically expressed in terms of desired functions the artefact should deliver [16]. Since there are no rules for the activation of a particular model, the relevance of certain part of domain theory depends on the current *context* in design. 'Context' $\mathcal{C} = \mathcal{R}_X \cup \mathcal{A} = \{r_i^X, a\}$ is determined by all explicitly set requirements ($\mathcal{R}_X$) and current assumptions ($\mathcal{A}$) that needs to be confirmed or rejected in course of design (see also section 4.2).

In other words, a set of relevant models and potential solutions ($\mathcal{S}_G \subset \mathcal{T}_M$) is retrieved from the domain theory $\mathcal{T}_M = \{t_i^M \subset \mathcal{E}_S \times \times \mathcal{E}_P \times \mathcal{E}_F\}$ so that available functional elements from $\mathcal{T}_M$ are compared against and associated with those that are desired for a final artefact ($\mathcal{C}$). If retrieval yields a non-empty set $\mathcal{S}_G$, we say that $\mathcal{S}_G$ is an interpretation of context $\mathcal{C}$ in domain theory $\mathcal{T}_M$.

3.3 Case-based design knowledge

Knowledge of theoretical models as defined in the previous section is unfortunately, not sufficient if our aim is to support design in a way that would be similar to what experienced designers do. As several studies of design [5, 6, 10] point out, designers in practice often re-use their previous experience when they attempt to see a new problem in a familiar framework. Previous experience may take form of 'design cases' that can be indexed according to multiple criteria [12, 20], retrieved and adapted to the current 'case'. A set of known cases is defined as $\mathcal{T}_C = \{t_i^C \subset \mathcal{E}_i^S \times \mathcal{E}_i^P \times \mathcal{E}_i^F \times \mathcal{L}_i \mid \mathcal{E}_i^F \Leftrightarrow \mathcal{R}_i\}$, where $\mathcal{R}_i$ is a refined set of requirements on the artefact designed in scope of case t_i^C.

A *case* differs from a *model* in several conceptual facets. First, a single case t_i^C may contain one or more models t_i^M that were already interpreted in the context of requirements associated with case t_i^C. In other words, 'models' are interpretation-free and elements from a domain theory can be combined in any way that is allowed by domain theory through the introduction of new assumptions. On the contrary, elements and relations making up a 'case' are bound by a set of requirements that specifies the design task for that particular case, and no further assumptions are needed to be uncovered in that task. However, the task and its requirements may be re-formulated to suit the needs of the current problem and new assumptions may be added to extend the context of the current problem.

Adaptation of a retrieved case for the current problem is a critical operation in all case-based approaches [15]. We believe that a properly structured and indexed domain theory may be helpful in this effort. As mentioned above, domain theory is hierarchically structured; therefore it seems to be beneficial to use it (in addition to the description of different elements) also for the discovery and appreciation of similarities between conceptually different elements. Eventually we should be able to map elements and relations from a previous case onto the current (only partially described) problem.

3.4 Representation of design knowledge

Support for design process in its early phases needs a mechanism for knowledge clarification. A mechanism that is flexible, describes design elements on conceptually different levels, suggests possible relations and associations between elements, and finally, suits both parties involved in the design (humans and computers). One such mechanism has the form of knowledge models [9] (KM) that are built using entities known as *common ontologies* [21, 22]. Ontology is an explicit specification of conceptualisation; also it is a representation vocabulary specialised to a domain or subject matter. Ontologies help to clarify structure of domain knowledge and provide means for knowledge communication, sharing, re-use, and transfer among agents using different internal representations and methods. Ontology is inherently hierarchical entity, which makes it a very useful means for the representation of complex knowledge; such as characterises e.g. design.

To sum up, ontologies can be used to represent different types of design knowledge. These may be understood as multiple dimensions according to which knowledge may be classified so as to assist in the exploration of complex design-related KB:

(1) Ontology for *domain knowledge* consists of common and basic terms used for the representation of requirements and solutions in a particular domain; it includes theoretical foundations of a domain and relations among elements that can be used in that domain. It is more static compared to other dimensions, i.e. it changes less frequently than design knowledge.
(2) Ontology for *knowledge indexing* maintains an unambiguous structure in KB and relates domain elements through generic reference ontologies. It is crucial for the efficient retrieval of cases related to current design problem, as well as for the reasoning by analogy with retrieved cases. Our approach assumes that initial requirements on a designed artefact are functional and uses knowledge about functional elements in a particular domain as index to explore the principles and structures through which the desired functions may be attained.
(3) Ontology describing *design tasks and cases* reflects good practices and experience gained when solving problems in the past. It describes ways how similar problems were approached in the past and includes design guidelines, successful designs from the past, various justifications and explanations. This knowledge is used to manipulate domain knowledge and is subject to frequent changes and extensions, as new methods are proved useful.

4. CO-EVOLUTION IN DESIGN

In this section I will present some implications of the theoretical proposal made in the previous sections with an emphasis on uncovering or explication of hidden requirements. This hidden subset of all requirements set on the artefact plays an important role in the exploratory design. To some extent it is even possible to say that hidden requirements discovered during design is what makes all the difference between routine and innovative designs. Further sections attempt to formulate an operational framework that supporting the co-evolution and discovery of hidden requirements and solutions.

4.1 Discovery and development of a solution

As mentioned in section 3.1, a solution to a design problem is a combination of structural elements, which is able to deliver desired functions through identified domain principles. We may formally define solution as a relation between two entities – a combination of elements and current formulation of a problem. However, before doing this, we have to formally introduce *problem formulation* $\mathcal{P}_i$ as a union of the set of all requirements and the set of all assumptions; i.e. $\mathcal{P}_i = \mathcal{R}_i \cup \mathcal{A}_i$. Combination $s_i \in \mathcal{S}$ satisfying problem formulation $\mathcal{P}_i$ is a solution when together with knowledge from domain theory $\mathcal{T}$ satisfies all current requirements and assumptions:

$$\text{solution}(s_i, \mathcal{P}_i) \leftrightarrow (s_i \cup \mathcal{T} \vdash \mathcal{R}_i) \wedge (s_i \cup \mathcal{T} \vdash \mathcal{A}_i) \qquad \textbf{Eq. 1}$$

In other words, certain combination of domain elements may be considered as a solution in a particular iterative step only if it can be meaningfully interpreted in the current context of a given problem. Such a definition of the solution may seem straightforward at first. However when we take into account that $\mathcal{R} = \mathcal{R}_X \cup \mathcal{R}_H$, we see that to be able to decide whether a combination s is an actual solution we must uncover yet hidden requirements $\mathcal{R}_H$.

Next, the process of solution discovery using domain theory will be formally introduced. Let us begin with the identification of '*absolutely relevant*' functional elements, i.e. those that are directly mentioned as explicit requirements and denote this subset as $\mathcal{F}_0$.

$$\mathcal{F}_0 = \mathcal{R}_X \cap \mathcal{E}_F = \{e^F \mid e^F \in \mathcal{E}_F \wedge \text{used\_in}(e^F, \mathcal{R}_X)\} \qquad \textbf{Eq. 2}$$

Set $\mathcal{F}_0$ can be extended so that it would contain also other functional elements that may be somehow related to those explicitly mentioned. Such a set $\mathcal{F}$ may be named a set of '*not absolutely irrelevant*' elements. It is this particular extension that plays significant role in design in a sense that it makes designers aware of other 'similar' elements that may have been forgotten but have some relationship with the current problem. Formally:

$$\begin{aligned}\mathcal{F} = \{e^F \mid e^F \in \mathcal{E}_F \wedge e^F \in \mathcal{F} \vee \text{is\_assumption}(e^F) \vee \\ (\exists e_p^F \in \mathcal{F} \mid \text{used\_in}(e_p^F, \mathcal{R}_X) \wedge \text{similar}(e^F, e_p^F))\}\end{aligned} \qquad \textbf{Eq. 3}$$

$$\begin{aligned}\text{similar}(e^F, e_p^F) \leftarrow \exists \langle e^F, e_p^F \rangle \subset \mathcal{E}_P \times \mathcal{E}_F \mid e^F \neq e_p^F \\ \wedge (\text{used\_in}(e^F, \mathcal{X}) \wedge \text{used\_in}(e_p^F, \mathcal{X}) \\ \vee \text{subclass}(e^F, e_p^F) \vee \text{superclass}(e^F, e_p^F))\end{aligned} \qquad \textbf{Eq. 4}$$

Now, let us call the elements contained in set $\mathcal{F}$ as '*pattern*' to appreciate the fact that they may be used for the retrieval of relevant models from $\mathcal{T}_M$ or relevant cases from $\mathcal{T}_C$. Case solutions or models that are applicable to the current problem can be retrieved from the available knowledge repository using a technique, in which a pattern will be sought among the requirements and functional elements of a particular case (or model). Once the pattern is found, it is possible to retrieve also principles and structures realising that particular functionality – either from theory $\mathcal{T}_M$ or case repository $\mathcal{T}_C$.

When a retrieved case is to be used as a 'mould' for the current problem, it must re-formulated using 'the vocabulary' of the current problem. Case adaptation is often a complex task and a lot of work is devoted to this single operation. It is not my intention to discuss this more in-depth here; only briefly – to find possible mappings between elements already known in the current problem and elements known in a retrieved case can be found using hierarchical (ontological) knowledge of domain elements and relations.

Anyway, the result of such retrieval is a set of relevant solutions, or rather relevant combinations of elements that satisfy explicit requirements of the current problem. Whether these combinations will keep their feature of a problem solution in a long run, depends on what new requirements and assumptions will be added to the current problem formulation. A new assumption may render currently sound combinations of elements invalid and trigger the process of perspective shift and consequently refine both, the formulation of and solution to the problem.

4.2 Discovery of hidden requirements

It seems appropriate to assume that there exist several conceptually different types of hidden requirements, such as for instance:

(i) '*deducible domain requirements*'… these can be derived using knowledge of known elements within a particular domain, especially knowledge of domain principles (e.g. if law of conservation of energy holds and electricity or sunlight are different forms of energies, it follows that energy of sunlight must be preserved and possibly transformed into different kind of energy – electricity);

(ii) '*common truths*'… these cover commonly accepted facts that are usually not proved or derivable from domain knowledge (e.g. unless specified otherwise we design artefacts for conditions valid on the Earth);

(iii) '*statistical requirements*'… these can be derived through inductive reasoning and generalisation from previous cases, when a requirement is accepted and justified by reference to an existing case without further proofs (e.g. a typical requirement on a satellite is to make its weight as low as possible);

(iv) '*reflective requirements*'… initially expressed as assumptions that must be 'proved' through development of a solution before it can be 'promoted' among uncovered requirements relevant to the current problem (e.g. suppose, we want to use solar panel as a power supply for a satellite, then according to domain knowledge we must first ensure there is enough sunlight available; we are also aware that we arrived at this additional requirement through backward causal reasoning)

The following formulae describe the mentioned conceptual differences formally using the same notation as in previous sections:

(i) $$\mathscr{R}_X \cup \mathscr{T}_M \cup \mathscr{L} \models \mathscr{R}_H \qquad \textbf{Eq. 5}$$

(ii) $$(\mathscr{R}_X \cup)\ \mathscr{T}_M \cup \mathscr{T}_C \models \mathscr{R}_H \qquad \textbf{Eq. 6}$$

(iii) $$\mathscr{R}_X \cup \mathscr{T}_C \models \mathscr{R}_H \qquad \textbf{Eq. 7}$$

(iv) $$\begin{aligned} \mathscr{R}_X \cup \mathscr{T}_M \cup \mathscr{A} \cup \mathscr{S} &\models \mathscr{R}_H \\ \mathscr{S} \cup \mathscr{T} &\models \mathscr{A} \end{aligned} \qquad \textbf{Eq. 8}$$

As already mentioned above, the last type describes formally the interplay between and co-evolution of explicit and hidden requirements on one side and solutions satisfying these requirements. As a new requirement is uncovered or an assumption is proved, they can be included in the set $\mathscr{R}_X$, which may have some impact on the retrieval of related cases or models from domain knowledge. A new case, model or mapping may have impact on the identification of some new assumptions, which in their turn may uncover another hidden requirement. Thus we are provided with an abstract recursive description of processes that may be observed in design and that are often referred to as '*solution talkback*' [5, 11].

5. ILLUSTRATIVE EXAMPLE

Let us assume that the 'supported stage' of design begins with a non-empty set of initial design requirements (e.g. find a device supplying electricity to a satellite). How the theoretical framework may be applied is shown below using an OCML-like notation.

5.1 Not-irrelevant functional elements

The input is a set of initial requirements; particularly important are those about functionality of the artefact. Their formulation may consist of the function identifier, substance identifiers and concepts delivering that functionality. Assume that domain knowledge base contains information about (not irrelevant) functions and other elements as shown below:

```
(Substance-Concept Energy ())

(Substance-Concept Electrical-energy (Energy)
(applicable-to-parameters
   :has-domain '(electrical electromagnetic)
   :equals-to '(electricity)))
 …

(Function-Concept Energy-supply ()
(applicable-to-parameters
   :has-function '(supply-energy)
   :has-substances '(energy)
   :relates-to-concepts '(generator load))))
 …
```

Computer-based KBDSS may take the initial requirement formulated by the designer and apply it to the available indexing KB containing known substances and functions. Further, let us assume that in our example there is no direct association in KB containing the facts mentioned in our requirement. However, a simple reasoning in the ontology reveals that terms not irrelevant to '`electricity`' include '`electrical-energy`' and its parent concept '`energy`'. Consequently, the KBDSS can associate concepts '`energy`' and '`electrical-energy`', and suggest two functions as possibly relevant: '`supply-energy`' and '`supply-electrical-energy`'. Once the designer accepts this suggestion, KBDSS may retrieves other related elements (e.g. structural element '`generator`' or domain principle '`supply-electrical-energy`').

5.2 Development of potential 'models' of a solution

Obviously, the list of retrieved elements is not exhaustive because it contains only those elements that are known as theoretical domain knowledge or were used in the previous design cases. Anyway, the list of relevant simpler elements may later trigger the introduction of new elements as design progresses. The next step is to investigate how the desired functions may be realised. Assume, there are following facts in the domain KB:

```
(Model-Concept Electrical-model ()
(applicable-to-parameters
   :has-structure
      (:connections '()
       :container '(electrical))
   :has-substance '(electricity)
   :described-by-quantities
      '(power voltage current)))

(Model-Concept Battery (Electrical-model)
(applicable-to-parameters
   :has-structure (:connections '(term-A term-B))
   :has-primary-functions
      '(supply-electrical-energy)
   :has-secondary-functions '(charge discharge)
   :described-by-quantities '(capacity)))
```

Similarly as substances, these structural elements are ordered in an ontological hierarchy. For instance, there is a concept of '`battery`' derived from a generic '`electrical-model`'. Simple reasoning through these elements reveals other related elements; such as two connections called '`term-A`' and '`term-B`', an '`electrical`' container containing the substance '`electricity`', and possibly physical quantities typically used in this context. What is their possible role in the current problem? That can be shown when an explanation of desired functions is retrieved from the stored domain knowledge; e.g. in a form of domain principle:

```
(Principle-Concept Supply-electrical-energy
(applicable-to-parameters
   :has-structure
      (:connections ‘(term-A term-B)
       :container ‘(electrical))
   :has-substance ‘(electricity)
   :described-by-quantities
      ‘(power voltage current capacity)
   :defined-as
      ((pump ‘electricity (BETWEEN ‘term-A ‘term-
      B) (THROUGH ‘electrical))
      (allow ‘electricity (BETWEEN ‘term-A ‘term-
      B) (THROUGH ‘electrical) :voltage (propor-
      tional ‘capacity)))))
…
```

5.3 Explication of hidden requirements

Device ‘`battery`’ as identified in section 5.2 satisfies the desired requirement through the application of domain principle ‘`supply-electrical-energy`’. Let us suppose that the designer decides to pursue this alternative further. Other relevant elements, concepts and descriptions may be retrieved from the domain knowledge base that are connected with already identified elements. This corresponds to the extension of set $\mathcal{F}_0$ with other not irrelevant elements to make up set $\mathcal{F}$. For instance, we may learn about physical quantities and functional elements like ‘`capacity`’, their typical features and principles governing their behaviours, such as ‘`electricity-charge-discharge`’ or ‘`time-dependence`’, etc.

The discovery of these elements and their relations may be a trigger that makes the designer aware of possible shortcomings of the current combination of elements. It may give also some hints how to overcome the deficiency, or the fix may be spotted and introduced directly by the designer. Anyway, the last example shows the form how current combination of elements may look like and how it may be used to show possible implications and eventually extend the set of initial design requirements (see section 5.1).

```
(delivers-func ‘Battery ‘Supply-electrical-energy
(applicable-to-parameters
   (:I (has-quantity ‘Battery ‘capacity)
    :J (by-domain-theory ‘reference-1)
   (:I (has-property ‘capacity ‘time-dependent)
    :J (by-domain-theory ‘reference-2)
   (:I (has-behaviour ‘capacity ‘decrease-in-time)
    :J (by-domain-theory ‘reference-3) …))
```

Implications (denoted by `:I`) introduce new facts that are related to known elements; implications are justified (`:J`) by referring to available domain theory ($\mathcal{F}_M$ or $\mathcal{F}_C$). This is one way how implications uncover hidden knowledge, which may eventually lead to the formulation of new requirements or modification of existing ones.

6. FURTHER RESEARCH

The aim of this paper was to present a theoretical framework that can be eventually used to talk about and perhaps better understand processes underlying such complex and typically human activity as design. The framework represents a work in progress that was done mainly by analysing several existing approaches in the research community. Therefore, the next major step in my research is to perform several pilot studies, in which the claims formalised in this paper will be rigorously validated.

The setting of our experiments will involve two designers from practice and one or two ‘advisers’ providing support for the participants. These advisers take the role of a knowledge-based system in the testing phase. However, to restrict the undesired interactions between designers and support articulation of knowledge-intensive processes, all participants will use a tailored user interface that was built to accommodate the proposed framework, except the actual presence of computer-based knowledge base.

Design tasks given to the designers will be from the domain of design of controllers for complex technological processes, and belong to the category of non-routine tasks. However, it is possible that approaches the designers undertake in their work will be eventually perceived by the designers as (personally) innovative. The articipants will be asked to record all their decisions, assumptions and justifications of various decision steps they make during design. Expert advisers are allowed to give some basic hints that must be always justified by a reference to shared domain knowledge; advisers may also require similar justifications from the designers ‘to prove’ that a particular decision was not entirely accidental though it may be difficult to articulate. Features to be observed include among others:

- designers working on two parallel levels (refining problem requirements and developing a solution);
- designers’ attention being shifted between these two conceptual levels of design problem;
- discovery and inclusion of new knowledge thus extending their knowledge about the current problem as well as knowledge about design and domain.

The ultimate question to be answered by the experiment is whether the *reflective framework is a feasible approximation of design* and what are major (dis-)advantages.

7. CONCLUDING REMARKS

This paper presented a theoretical framework for the description of design processes that may be used for the construction of KBDSS. The framework may be perceived as a *formal model* of design with respect to some typical processes underlying otherwise complex act of designing an artefact. Design is seen as a problem of requirements explication through simultaneous development of design solutions and reflection on them and also on processes through which these solutions were attained. The reflection from the space of solutions to the space of requirements may be eventually supported by a computer-based tool, which reasons about the models and cases from the domain knowledge based on incomplete initial requirements. Figure 1 describes the entire framework in a graphical form, showing possible operations how knowledge may be manipulated at various stages of design process.

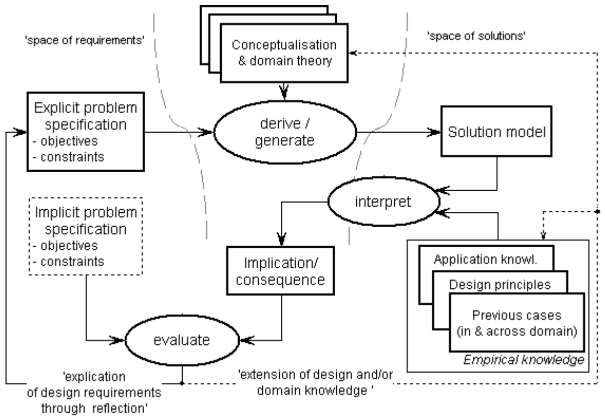
'space of requirements'
Conceptualisation & domain theory
'space of solutions'
Explicit problem specification
- objectives
- constraints
derive / generate
Solution model
interpret
Implicit problem specification
- objectives
- constraints
Implication/ consequence
Application knowl.
Design principles
Previous cases (in & across domain)
Empirical knowledge
evaluate
'explication of design requirements through reflection'
'extension of design and/or domain knowledge '

Figure 1. Formal ‘model’ of reflective design.

The proposed framework is able to address several important features of design process [23]. It is inherently iterative and recursive. Due to recursion it is possible to observe certain degree of reflection directly in the formulae describing design activities. As shown in Figure 1, reflection in design may include both, reflection on the problem formulation and corresponding solution, and reflection on knowledge applied to derive a particular solution. Consequently, both problem formulation and domain knowledge are subject to change as a result of 'reflection'.

Further, the initial ambiguity of design problems is appreciated and addressed in a way, where suitable models and cases are not determined by a strict prescriptive rule. Contrary the approach is rather opportunistic; i.e. that relevance of domain knowledge depends on the current context of design problem formulation. Also, it means that the same element may acquire different interpretation based on the context of requirements in which it is used. Contextual dependence should allow much larger flexibility compared to those approaches where an element is interpreted equally in all different situations. This dependence is also related to the exploratory nature [1] and ill structure [2] of design, because the design space is not completely known before the design process begins. As mentioned at the beginning it is built up 'on-the-fly'.

ACKNOWLEDGEMENTS

I acknowledge the contribution of Zdenek Zdrahal, John Domingue as well as all reviewers who helped to clarify some interesting points in my PhD. research and/or participated in numerous discussions about possible formalisations of design. The paper was also supported by the Esprit research project *Enrich* (No. 29015).

REFERENCES

[1] T. Smithers, *et al.*, Design as intelligent behaviour: an AI in design research programme. *AI in Engineering*, **5**(2): p.78-109, (1990).

[2] H.A. Simon, The structure of ill-structured problems. *Artificial Intelligence*, **4**: p.181-201, (1973).

[3] K.J. MacCallum, Does Intelligent CAD exist? *Artificial Intelligence in Engineering*, **5**(2): p.55-64, (1990).

[4] M. Tang, A knowledge-based architecture for intelligent design support. *Knowledge Engineering Review*, **12**(4): p.387-406, (1997).

[5] D.A. Schön, *Reflective Practitioner – How professionals think in action*. 1983, USA: Basic Books, Inc.

[6] L. Candy and E. Edmonds, Creative design of the Lotus bicycle: implications for knowledge support systems research. *Design Studies*, **17**: p.71-90, (1996).

[7] S. Nidamarthi, A. Chakrabarti & T.P. Bligh. *The significance of co-evolving requirements and solutions in the design process*. in *11th ICED*. Finland. (1997).

[8] M. Dzbor. *Intelligent Support to Problem Formalisation in Design*. in *3rd IEEE Conf. on Intelligent Engineering Systems (INES'99)*. Slovakia. p.279-284, (1999).

[9] A. Newell, The knowledge level. *Artificial Intelligence*, **18**(1): p.87-127, (1982).

[10] N. Cross, Descriptive models of creative design: application to an example. *Design Studies*, **18**: p.427-440, (1997).

[11] K. Nakakoji, *et al.*, From critiquing to representational talkback: Computer support for revealing features in design. *Knowledge-Based Systems*, **11**: p.457-468, (1998).

[12] S. Bhatta, A. Goel & S. Prabhakar. *Innovation in analogical design: A model-based approach*. in *3rd Intl. Conference on AI in Design (AID'94)*. Switzerland. p.55-74, (1994).

[13] F.M.T. Brazier, *et al.*, Modelling an elevator design task in DESIRE: the VT example. *Int. Journal of Human-Computer Studies*, **44**(3): p.469-520, (1996).

[14] J.S. Gero, Design prototypes: A knowledge representation schema for design. *AI Magazine*, **11**(4): p.26-36, (1990).

[15] I. Watson & S. Perera, Case-based design: A review and analysis of building design applications. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, **11**: p.59-87, (1997).

[16] B. Chandrasekaran, A. Goel & Y. Iwasaki, Functional Representation as Design Rationale. *IEEE Computer*, **26**(1): p.48-56, (1993).

[17] T. Bylander & B. Chandrasekaran. *Understanding Behavior Using Consolidation*. in *9th IJCAI*, California. p.450-454, (1985).

[18] Y. Iwasaki, *et al. How Things Are Intended to Work: Capturing Functional Knowledge in Device Design*. *IJCAI*. (1993).

[19] L. Qian & J.S. Gero, Function-Behaviour-Structure Paths and Their Role in Analogy-Based Design. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, **10**: 289-312, (1996).

[20] A. Gomez de Silva Garza & M.L. Maher, Design by Interactive Exploration Using Memory-Based Techniques. *Knowledge-Based Systems*, **9**(1), (1996).

[21] B. Chandrasekaran, J.R. Josephson & V.R. Benjamins, What Are Ontologies, and Why Do We Need Them. *IEEE Intelligent Systems & their applications*, **14**(1): p.20-26, (1999).

[22] T.R. Gruber, A Translation approach to Portable Ontology Specifications. *Knowledge Acquisition*, **5**(2): p.199-221, (1993).

[23] N. Cross, Natural intelligence in design. *Design Studies*, **20**(1): p.25-39, (1999).

[24] B.J. Wielinga, J.M. Akkermans & A.T. Schreiber, A Formal Analysis of Parametric Design Problem Solving. In *9th Banff Knowledge Acquisition Workshop*, Canada, (1995).

[25] M.E. Balazs, Design Simplification by Analogical Reasoning. *PhD. dissertation, Worcester Polytechnic Institute*, USA. (1999).