DESIGN PROCESS: HOLISTIC VIEW

V. Sedenkov

Belarusian State University
SW Engineering Department
e-mail: sedenkov@bsu.by

Keywords: CAD reengineering, design problem, action system, design process designing

Abstract: *Design process (DPR) disintegration, which apparently has entered the phase of self-development, is economically wasteful and technologically unpromising. A possible strategy to oppose this tendency is the revealing and neutralization some trigger factors. DPR complexity, for instance, is frequently referred to those assuming that the complexity reduction would weaken disintegration tendencies as well. But disintegration had arisen concurrently with design computerization. Hence, the properties of key design paradigm signs (paradigmants) should have yet something that does not hamper disintegration progress. Paradigmants modification contributing to DPR integration is referred to as reengineering of the current design paradigm (generally, CAD). The paper presents theoretical background, results and implications of logical reengineering.*

1. INTRODUCTION

The tendency of design process (*DPR*) disintegration into autonomous segments, presenting types, levels, aspects, stages and goals of designing, seemingly has entered the phase of self-development. Disguising itself as inevitability, this tendency entails drastic implications. *DPR* disintegration, conceived at one time with design automation, imparted to the latter a "piece-wise" mode and preserves it to present day as extensive and highly wasteful activity. The promising studies (*PLM*, *CE*, knowledge harnessing, etc.) cannot reach the expected efficiency under conditions of segmented *DPR*, while exploration for the segments often has no prospects. A rather fragmented, if not a chaotic, picture of this research [1] makes, on the one hand, its area boundless, and, on the other hand, causes the feeling of "the end of design methodologies" [2]. Finally, *DPR* disintegration triggers a number of other disintegrations – educational courses, expert corps, design science itself.

Our standpoint in respect of *DPR* disintegration consists in active counteraction. One of the strategies of such opposition may be the identification and subsequent elimination or neutralisation of disintegration factors (among those are, for instance, *DPR* complexity, *DPR* model dependence on an executive processor and a domain, unsolved problems, etc.) But such straightforward strategy does not exceed the limits of restraining facilities, and may turn to be unsuccessful under conditions of the current *CAD* paradigm.

The way to counteract disintegration more effectively should be, in our opinion, *CAD* reengineering. This strategy is based on the fact that disintegration is rooted in the properties and states of a few distinctive features of a design paradigm, which we call *paradigmants*. We associate with those the following fore members: design progress concept, design goal presentation, "the problem of the design problem", and action system. Then the proposed reengineering will consist in purposive modification of paradigmants' properties or states in that direction, which ensure holistic *DPR* presentation and realization. Starting with the elements of theoretical base being used, the paper describes reengineering operations and the results obtained.

2. ELEMENTAL THEORETICAL EXTRACTIONS

2.1. Continuous process theory

The subject matter of continuous process theory (CPT) [3] is the *scheme technique* of processes. The goal of the discipline is to proof the runability of some process through the building up for its scheme a runable continuous structure of processes. CPT serves for the major theoretical support of CAD reengineering. Its technique is characterized by the following:

- Each process (*PR*) can be presented by its *scheme*: *PR=(D, P)*, where *P* stands for a *processor* that performs transformation of energy, raw materials, information or products entering its input (I^P), and D stands for a *procedure* that describes the function of P over its I^P. D and P are referred to as a process *object* and *subject* respectively.

- A set of process schemes is added with a number of binary relations.
- Process schemes linked by distinguished relations make up a *structure*.
- The rules for structure formation and conditions for the structure runability are stated.

There are two relations appropriate for making up the structures: providing relation or **p**-*relation* and relation of determination or **d**-*relation*. PR_1 and PR_2 are linked with **p**-relation ($PR_2 \xrightarrow{p} PR_1$) if the output of PR_2 serves for the input of PR_1. If the output of PR_2 becomes a scheme component of PR_1 (D or P), these two processes are linked by **d**-relation ($PR_2 \xrightarrow{d} PR_1$).

A set of processes (or their schemes) continuously linked by **d**- or **p**-relation forms an *elementary structure* of processes (or processes schemes). This structure is represented by a graph, the nodes of which serves for the processes and each arc is a cross-linking relation. Elementary structures have an *order n*, equal to *1*. Elementary structures generated by one of the relations can form a new structure by the alternative relation; such non-elementary structures have $n=2$. Non-elementary structures may serve for the members in a structure of the next order based on the relation alternative to a previous one. A motive for structure formation may be as follows.

Associate with each process scheme a level of its *uncertainty* (*UL*) as *UL* of the scheme's components.

- A process, which has $UL=0$, is called *physical*: its D and P are real.
- $UL=1$ corresponds to a *logical* process: its D and P have descriptions sufficient for their physical implementation.
- A *virtual* process has $UL=2$: its D and P exist only as mental images.
- $UL=3$ is assigned to a *conditional* process (PR^C): its result has been declared but D and P are presented by their symbols only.

Constructive proof of logical runability for PR^C consists in stepwise reduction of its *UL*. A step of reduction is referred to as *determination* of conditional, virtual or logical process. While two-stroke determination of PR^C, the objective of the *virtual* (downward) determination is the reduction $UL=3 \rightarrow UL=2$; during the second or the stroke of *logical* (upward) determination, the reduction $UL=2 \rightarrow UL=1$ will take place. The outcome of this two-stroke determination cycle of PR^C is, so called, *S*-tree (*super-tree*) – an arc-bichromatic tree, each *S*-node of which is an ordinary tree (Fig. 2).

2.2. Problems schematics

When some conditional *PR* has been declared, it may need determination with respect to *D* and *P*. In that case, the processes *SD* (search for *D*) and *SP* (search for *P*) should be executed for *PR* (Fig. 1).

$$\mathrm{SD} \xrightarrow{d} \mathrm{PR} \xleftarrow{d} \mathrm{SP}$$

Fig. 1. *The structure of processes on d-relation*

The triple of processes as a whole, which represents the elementary structure based on **d**-relation, is identified as a *problem scheme* (*PRB*):

$$PRB = \langle\langle SD, SP\rangle\langle PR\rangle\rangle \qquad (1)$$

Here $\langle PR\rangle$ is the core of the scheme. The result of execution $\langle SD, SP\rangle$ is said to be a *solution* to the problem, while the outcome of $\langle PR\rangle$ is an *answer* to this problem. (It was G. Polya [4] who had segregated the problem realization into obtaining a solution and computing an answer.) The problems, which have an answer but have not a solution, are referred to as unsolvable of the *second kind*. The problems that have neither solution nor answer (due to the nature laws, for instance) have the unsolvability of the *first kind*.

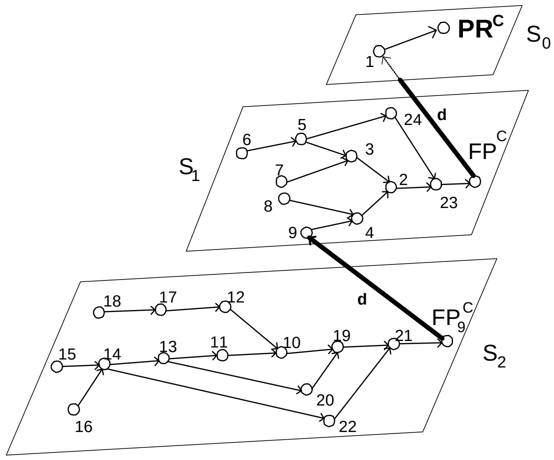

Fig. 2. *S-tree fragment*

Relations identified for a set of processes schemes stay valid for a set of problems schemes as well. This makes possible the problems cooperation with obtaining the problem structures equal to non-elementary structures made up out of processes schemes. In that case, relations between problems are equivalent to relations between the cores of their schemes.

In addition, introduce for a pair of problems one more relation – the relation of *substitution*. The problem to be substituted is named *original* (*OP*), and the substituting one is termed *conjugate* (*CP*). *OP* and *CP* are coupled in the following way: (a) their input data are different, (b) an actual answer to *CP* is identical to the virtual (required) answer to

OP, (c) the input data for *OP* play part of the control data (I^D) for *CP*. If *OP* has unsolvability of the second kind, the search and realization of *CP* is the unique way to get the answer to *OP*. The immediate example is presented by knowledge engineering: it deals with unsolvable problems of the second kind (for instance, diagnosis problem, or *OP*), and the part of *CP* is performed there by the problem of knowledge-based inference.

3. DESIGN PROGRESS CONCEPT

3.1. Design progress concept: AS IS

The product development process is a regular technical evolution [5]. However, the concept of evolution, accepted by a design paradigm, can have different forms: evolution of populations [6], evolution of individual entity [7], and pseudo-evolution when design evolving has a mediate form, reflecting some evolution of design description notation. In the last case, design description manipulation (stepwise refinement, for instance) results in design structure modifications.

Design progress concept (DPC) accepted in CAD is taken from the manual designing: this is the evolution of a description made for component architecture and treated as a reduction of an abstract level of the complete product structure presentation.

3.2. Design progress concept: TO BE

In the first place, DPC reengineering presuppose the disavowal of the *H*-centered DPC as stepwise concretization of a design complete description. So, there are two remainder candidates for a free DPC vacancy: autogenetic evolution and the evolution of individual. The first one is too laborious to be used while designing of complex products. So, the only choice for the part of DPC will be the "evolution of individual" treated as the adaption of the current design state (design maturity level, *ML*) to a new state of adaption environment (*AE*). The notion of *AE* is a derivative one. Its primary image is the notion of a product operation environment (*OE*). We specify the latter as follows.

OE is a set of ambient ongoing processes relevant to the product under design (the latter will enter *OE* after its physical implementation): $\{PR_q\}$. With the relevance to these processes, the product will play only three parts: it can be a process *subject*, a process *input* or a *disturbance* for a process (Fig. 3).

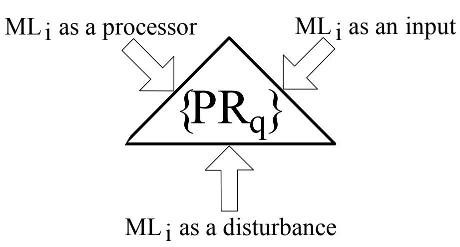

Fig.3. *Product relations to operation environment processes*

Hence, the family $\{\mathbf{PR_q}\}$ may be divided into three sets:

1. $\{\mathbf{PR}^{C}_{q_1}\}$ – the set of processes whose members accept the product as theirs subject (processor) and specify for the latter the *operating conditions* (*Cn*).
2. $\{\mathbf{PR}^{R}_{q_2}\}$: each member of the set takes the product for its input and places on this input a number of *requirements* (*Rq*).
3. $\{\mathbf{PR}^{L}_{q_3}\}$: members of this set take the product for a disturbance – a potential modifier of their *D* or *P*. These processes impose *restrictions* (*Rs*) on the product.

Thus, the hierarchy of sets in the family of *OE* processes assumes the form shown in Fig. 4.

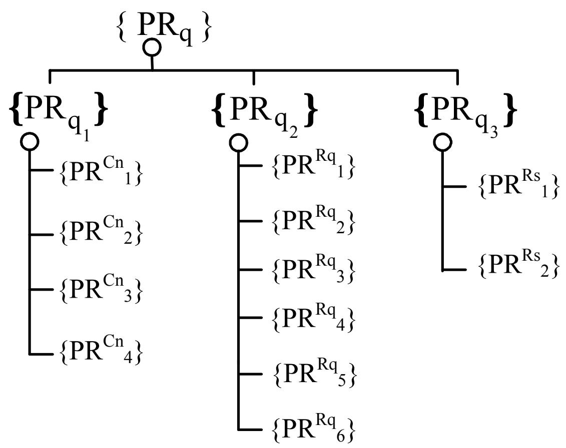

Fig. 4. *Operation environment hierarchical structure*

So, the processes of *OE*, which takes the product for their input, impose upon this input the *requirements* (*Rq*); this family of processes is shared by the life cycle stages. The processes that take a product for their subject (*P*) are the aim-achieving processes with different extents of completeness; these processes declare for their subject the *operation conditions* (*Cn*) and specify one or other scope of functionality. The processes, which take a future product for a *disturbance*, stipulate the necessary *restrictions* for the product (*Rs*), minimizing *non-intrinsic* product *resource* consumption. If the product operation itself is considered as a new process in *OE* being disturbed by the inner product processes, then the latter should be also imposed with restrictions minimizing *intrinsic* product *resource* consumption. Therefore, $AE = \{\{Rq\} \cup \cup \{Cn\} \cup \{Rs\}\}$.

4. DESIGN GOAL

4.1. Design goal presentation: AS IS

The current design paradigm either equates design presentation with product presentation or does not draw an essential distinction between those. Most often it is said of a product but not a design presentation. But for all that, the model of available product perception, borrowed from cognitive practice (structuring levels – subsystems, assemblies, parts), is used as the model for creation of unavailable design (system, organs, parts [8]).

The structure of *DPR* segments, which return product design descriptions, practically reflects the structure of a product under development presented at the level of subsystems, assemblies or parts.

We shall call the product presentation fixing its componentization at the moment Δt_i as *synchronous* one (*sh*). This presentation is based on cognitive motives, it is natural only for a real product and is usually realized by an hierarchy of abstract leyers for the lists of components. Devide the creation of *sh*-presentation on three stages, the results of which servers for building blocks while the product structure description formation:

1.1 *sh-structure* scheme – an hierarchy of names assigned to product decomposition layers (subsystems, assemblies, parts);

1.2 *sh-decomposition scheme* – sets of product constituents; each set refines an element from the previous layer of hierarchy;

1.3 *sh-structure* – product componentization obtained by application of 1.2 to 1.1.

4.2. Design goal presentation: TO BE

The goal of designing is a *design* – a self-dependent and irrelative to any object (specific product or process) entity, which should have its own unique presentation. The non-existent design may have only *diachronic* (*dh*) structure – a sequence of names of yet unknown synchronous states ordered at continuous time base and interpreted as the design *MLs*. Then designing in evolutionary DPC appears as a sequential assignment of synchronous images (semantics) to the members of diachronic design structure. The language for *sh*-states remains unchangeable from the start to the final version of a design.

It is pertinent to note that the *DPR* structure is not considered here as the mapping of a product structure. Quite the contrary, a unified design *dh*-structure is borrowed for a product from the design process design (cf. section 7).

The tuple of abstract design states or *MLs* (the "boxes" assigned with *sh*-presentations) converging to a required *ML* is called an *approximate model* of a design (*AM*). By analogy with synchronous product structure from AS IS, we single out different in power *MLs* assosoations and call those as *premodels* or levels of *AM* completeness. List these premodels for the case of diachronic structure and juxtapose them with the *sh*-premodels (Fig. 5):

2.1 *dh-structure scheme* – *q*-hierarchy (hierarchy of recurrent tuples) of higher *MLs*;

2.2 *dh-decomposition scheme* – a set of lower *MLs* ordered into some structure, the members of which are called *intervals*;

2.3. *dh-structure* – a full range of intervals (for instance, *ML* intervals) obtained by substitution 2.1 into 2.2.

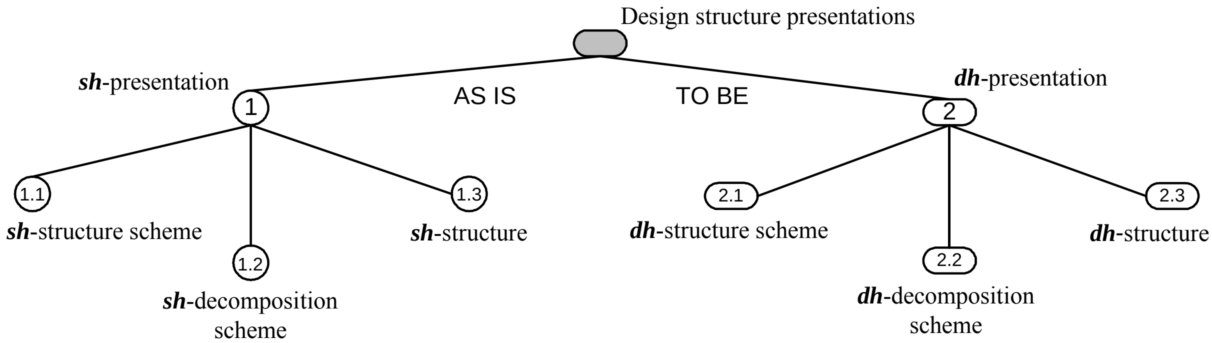

Fig. 5. *Basic premodels obtained while development design structure presentations in AS IS and TO BE*

The structures of diachronic *AM* are specific for different objects. The choosing of those, as well as the use of one product *AM* for the premodel role of another product, clears the way to generate for some collection of objects a unified *AM* structure. In the case of evolutionary DPC, such a collection consists of the needed product, its operation environment and a design process. Since the design goals in our DPC are defined as the reqired product design and its *AE*, let us construct for those individual *AMs*.

4.3. Product design approximate model

Following evolutionary DPC, we distinguish for a design four sequentially attainable states and name them *design goals*:

- *Prototype* (*PRT*) – design state that implements basic features of the required product, declared by conceptual description of the demand;

- *Market version* (*ITM*) – further functional evolution of *PRT* to the level of secured market callability;
- *Manufacturing version* (*COM*) – *ITM* evolution that stresses especially a range of issues from the product process planning to after-sales service;
- *Artefact* (*ART*) – aesthetic, sustainable and usable *COM*.

Next, let each design goal involves four successively attainable *design subgoals*. List them as follows:

- *Quasisystem* (*qSYS*) – a minimal set of product units capable to realize within the scope of some *ML* the basic functions specified by a developer;
- *System* (*SYS*) – the extension of *qSYS* with the components that ensure interaction of their units and introduce control functions;
- *Quasidesign* (*qDES*) – space layout of the system constituents;
- *Design* (*DES*) – it is *qDES*, every component of which is assigned with a shape, materials and all necessary joints.

Transform the received hierarchy of elements into *quasi-hierarchy* (*q-hierarchy*) by closing the nesting hierarchy, i. e. making the latter actual across horizontal as well. In this case, the terms of a tuple concretizing some parent design state are coupled in the way when the previous term is nested into the next one (→). Besides, the state corresponding to the end term of the tuple (for instance, *DES*) is equivalent to the parent design state for this tuple (for instance, *PRT*). The result of *dh*-structure construction for the product design *AM* is shown in Fig.6

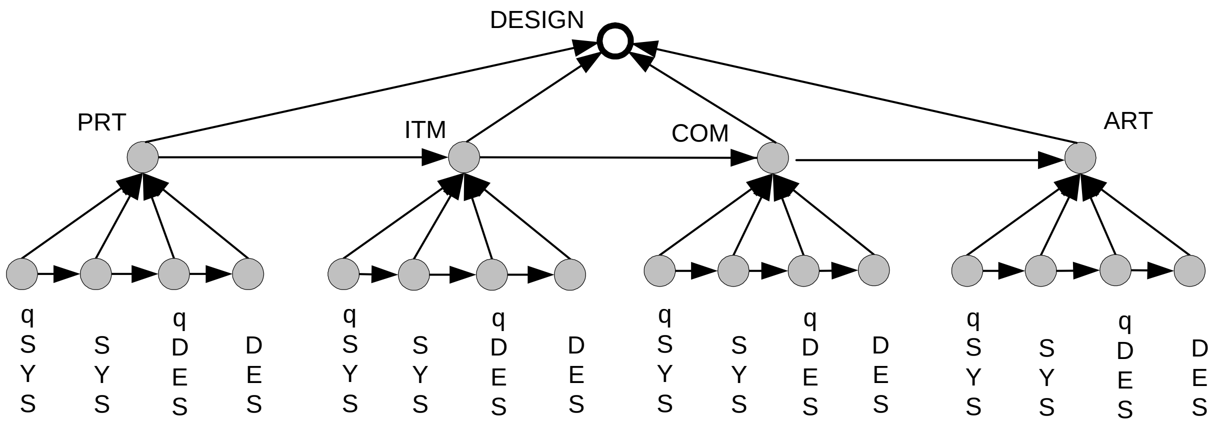

Fig. 6. *dh-structure for a virtual product design*

4.4. Operation environment *dh*-structure

Here we employ a complete cycle of compiling *AM* structure out of premodels, since, in contrast to inexistent product, *OE* is available and needs only to be refined. To get a scheme of *OE dh*-structure, we use its *sh*-structure (Fig. 4): take for the desired scheme a vector space, the rank of which is equated to the number of members in the second layer of *sh*-hierarchy processes (Fig. 7*a*).

The scheme of *OE dh*-decomposition will make up the member tuples out of the third hierarchy layer from Fig. 4. Then the substitution of *dh*-decomposition scheme into the scheme of *dh*-structure (Fig. 7*b*) will set the length of vectors, which constitute *OE dh*-structure interval space.

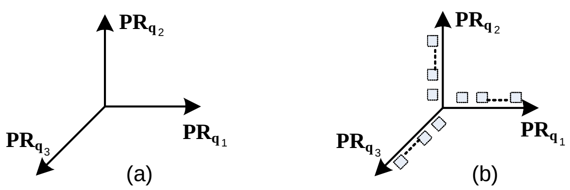

Fig. 7. *Adaption environment construction*

Having restored the three-dimensional space of intervals (Fig. 8) and assigned a track of their scanning, we shoud get *dh*-structure of *OE*, named **&**-*cube*.

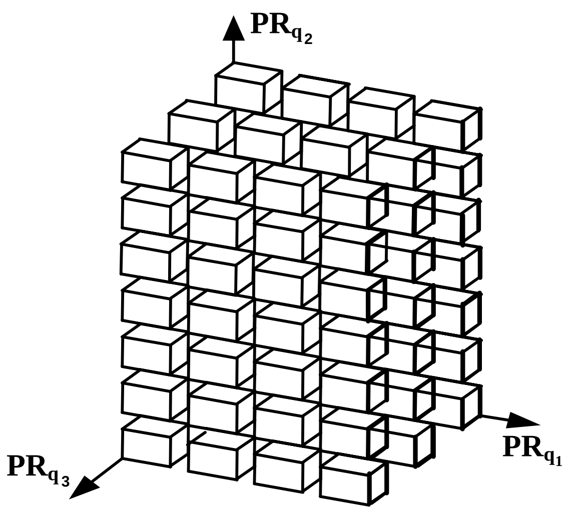

Fig. 8. *Operation environment dh-structure*

There are two possible types of the tracks: the first one looks like $<PR_{q_1}<PR_{q_2}<PR_{q_3}>>>$ while the second one is <PR_{q_2} <PR_{q_1} <PR_{q_3}>>>. Incident intervals along the track in *OE dh*-structure are coupled with the nesting relation: the contents of a previous interval becomes a part of the next interval.

5. ACTION SYSTEM

This paradigmant is presented by: (a) two types of processors (informal processor *H*, i. e. human being, and formal processor *C*, or computer), (b) a list of type members, (c) a sort of relation between the types and type members (for instance, "agent-server") during realization of separate procedures or *DPR* model in large.

5.1. Action system: AS IS

Semi-intuitive design terminology remains to be *H*-oriented. Common semantic base for *H* and *C* is missing: *C* has been plunged into alien environment of notions, models and methods that imposes considerable restrictions on its abilities.

5.2. Action system: TO BE

While $P \in PR=(D,P)$ is a physical processor in *PR*, $D \in PR$ can be considered as a logical processor, which also has own input – the processed control data (I^D). Then any $D \in PR$ is executed in the general case by a pair of processors – *working* one (P_W), processing an input of $P \in PR$ (I^P), and *information* processor (P_I), supplying I^D for $D \in PR$.

This observation brings up the situation: on the one hand, P_I has not been presented in *PR* scheme; on the other hand, we are dealing with *PR* providing (I^P) related only to its subject (*P*) whereas the latter should be provided with I^D as well.

The only way to avoid inconsistency and harmonize the situation is to include P_I into the scheme $PR=(D, P)$ and consider P_W and P_I as a unified $P \in PR$ named *diprocessor*, di$P=P^{P_I}_{W}$, and concretized as H^H, H^C, C^H or C^C. A process, the subject of which is one of the listed diprocessors, is referred to as *manual*, *computerized*, *automated* or *automatic* respectively.

Thus, *AS* reengineering assumes the exchange of hierarchical relation between *H* and *C* for the relation of cooperation when P_W and P_I have equal awareness for some function realization and ability to change the status. Henceforth we are dealing with three types of processors – *H*, *C* and di*P*. Hence it follows that the subject (*P*) of automated *DPR* will be di$P=C^H$.

6. DESIGN PROBLEM

6.1. Design problem resolution: AS IS

Designing is referred to as a unique type of problem solving, which requires devising future states of the world (goals), recognizing current ones (initial states) and finding path to bridge both (transformation function, *TF*) [9]. But design problem (*DP*) has the reputation of ill-structured and even wicked problem [10]. By the ill-structuredness is meant the deficit of information in each of three *DP* components: there is very little information about initial problem state, even less information about the goal and no information about *TF*. Nevertheless, *DP* is somehow solved and this presuppose the explicit distinction for it (perhaps ersatz) a goal, an initial state and *TF*. How could it occur?

While reducing design problem underdetermination, the virtual goal (design) is splitted into *k* (*k=1,2,...*) abstract and ordered images. This action entails the splitting of original *DP* into a series $\{DP_i\}$, $i=\overline{1,k-1}$. Then the pairs of elements with numbers (*i*, *i+1*) from the set of goals would constitute the initial state and goal of each DP_i respectively, while the load of TF_i is the transformation of the *i*th design state into (*i*+1)th one. To illustrate the outlined skeleton and sharpen the way of *DP* solution in CAD paradigm, turn now to the problems schematics (cf. section 2.2.).

In response to the description of needs, intentions and requirements, generate a scheme of conditional DPR_k, which has to return a realizable design within the scope of "refining" DPC. Restore for the DPR_k scheme the design problem scheme – DP_k = <<*SD*, *SP*><DPR_k>> – and start its solution as two-stroke determination of DPR_k (cf. section 2.1.).

Virtual determination of DPR_k with respect to its object gives ↓*D*="Deriving the description of a realizable design from its previous more abstract description". The value of virtual $\downarrow D \in DPR_k$ indicates the necessity to generate for DPR_k a process DPR_{k-1} that supplies the needed design description. After DPR_{k-1} extanding to the problem scheme DP_{k-1} and DP_{k-1} virtual solution with respect to *D*, we come to generation of DPR_{k-2}, and so forth (Fig. 9) until the next generated *DPR* turns out to be virtually and logically provided (DPR_1). This stands for the end of DPR_k virtual determination.

DP_k = <*SD*, *SP*><DPR_k>
↑
DP_{k-1} = <*SD*, *SP*><DPR_{k-1}>
↑
DP_{k-2} = <*SD*, *SP*><DPR_{k-2}>
↑
....
↑
DP_2 =<*SD*, *SP*><DPR_2>
↑
DP_1 =<*SD*, *SP*><DPR_1>

Fig.9. *Problems structure reflecting DPR_k determination*

It should worth to notice that DPR_1 is provided not at all by the needs, intentions and requirements (or, henceforth, by $\{Rq\}\cup\{Cn\}\cup\{Rs\}$) but by available analogue or prototype for the desired product.

If there is no analogues solution, designer has to synthesize its initial approximation ("to peep at solution") – this agrees with the way of *DP* realization named by Restrepo [9] "solution-led", while the presence of analogue gives its "problem-led" version.

Then the stroke of upward (or logical) determination of processes DPR_i, $i=\overline{1,k}$, begins. Determination of $\uparrow D_1 \in DPR_1$ gives the following value of *TF*: $\uparrow D_i$ = "Transformation of the *i*th design state into the (*i*+1)th state". Regular and complete *TF* is unavailable here since it involves unsolvable structure synthesis problem (*SSP*). Therefore, the state changing is concretized as translation the design description of the previous abstract level into design description with abstract level reduced for one.

Thus, design problem, that was valid until upward DPR_k determination, has been substituted by the conjugate one – the problem of translation, which is regularly solvable only in the particular case (automated synthesis of products with highly regular structure, mostly in electronics).

Outside this case, the solution of translation problem consists of implicit structure synthesis, worked out by *H*, and the ascending treatment of design subproblems distinguished and ordered into hierarchy within the scope of DP_i. Due to the ascending treatment of subproblems inside DP_i and ascending solution of DP_k in the structure in Fig. 8, we identify the realization of original *DP* in CAD as "*upward*" one (DP⇑). List now the major implications of this realization analysis.

1. The mode of *DP* realization is *H*-centered (borrowed from manual design): the abstract goal (design) is splitted into three abstract subgoals – conceptual, embodiment and detailed design. This entailes the splitting of initial *DP* into three successively solved design problems – $\{DP_i\}$, $i=\overline{1,3}$. Each of three design stages employs a distinct language for the design description obtained.
2. For the initial state of DP_1 it is assumed some approximated solution, which has the form of available analogue. Thus, the problem under solution in CAD corresponds implicitly not to design problem but to conjugate one. The latter is the translation problem for descriptions of the same design accomplished at different abstract layers[1].
3. $\{Rq\}\cup\{Cn\}\cup\{Rs\}$ as the formal image of needs, intentions and requirements cannot serve for the DPR_i input. (Mejers [11] also remarks that "needs, requirements and intentions" and "structure" belong to different conceptual worlds.) $\{Rq\}\cup\{Cn\}\cup\{Rs\}$ plays the role of control information for incomplete, irregular and domain-specific TF_i, equivalent to $\uparrow D_i \in DP_i$.
4. *TFs* coupled with DP_2 and DP_3 (i. e. $\uparrow D \in DPR\text{-}DP_2$ and $\uparrow D \in DPR\text{-}DP_3$) are different functions of translation (for conceptual design and embodiment design). As for $TF \in DP_1$, resulted in conceptual design, it can not be a synthesis procedure for the latter.

   Conceptual design describes the level of subsystems, and when there is no an analogues for the product under design the separation of subsystems is the final but not initial phase of design development. Hence, $TF_1 \in DP_1$ is the same sort of translation as *TF* from DP_1 and DP_2; the only difference is that TF_1 implicitly "translates" into conceptual design an available prototype of the required product.
5. The mode of *DP*⇑ realization imparts quite a naive nature to design automation in CAD: *H*-technology of *DP* treating with attached computer (computerized solution of subproblems and their coalitions). Naive design automation is extensive (task-by-task, product-by-product, aspect-by-aspect, etc.), expensive, and unbounded in time and space (of problems). The search for continuous problem areas of computerization, their compilation and recompilation is one of the permanent factors of *DPR* disintegration.

6.2. Design problem resolution: TO BE

Within the scope of *DP* structure analysis (Fig. 8), some directions for the ongoing reengineering are quite obvious.

1. The solution of *DP*=<*SD, SP*><*DPR*> with respect to subject (*P*) is de facto presented today by di*P*. Hence, the *DP* solution with respect to object (*D*) cannot stay a semi-intuitive and *H*-centered procedure, obscured for di*P*'s *C*-component. Despite the fact that many authors prefer to continue research on "how the designer works on *DP*" [12, 13], we think that *DP* solution with respect to object should dictate "how the di*P* has to work ".
2. Picking the evolution *DPC* (adaption the current design state to a new state of *AE*) opens up possibilities to split the goal (final design) not into *k* levels of its abstract presentation but into *n* (*n=1, 2,…*) specific levels of design maturity (*ML*) where $n >> k$. It should allow the replacement of uneven transitions of design states peculiar to CAD with the quasi-continuous increase of their *MLs*, employing for description of those a single language.

[1] Thus, *DP* used to be referred to as ill-defined or wicked is actually a phantom problem.

3. The way of DP_1 realization defines the way of resolution for the rest problems in the structure in Fig. 8 with the only difference that DP_1 turns, on the phase of logical DPR_1 determination, into the structure synthesis problem, while DP_2 and DP_3 move to incremental synthesis. This opens the way to reduce $\{DP_i\}$ to iterations of $DP_1=SSP$ with getting a unified and complete *TF*.

However, design creation cannot be reduced to merely *SSP* realization. Besides, the demands of completeness and domain-independence we make on *TF* are unattainable on application of $DP\Uparrow$. So, on account of above stated steps 1-3, we lay down the fourth generalized step of *DP* solution.

4. The completeness and domain-independence of *TF* are attainable only under downward *DP* realization or "realization as a whole" ($DP\Downarrow$). Therefore, we begin the revision of analyzable paradigmant not with the splitting of design goal and *DP* but with an attempt to realize the unsolvable *DP* "as a whole", i. e. with the search of a conjugate problem for *DP*, which should have *SSP* as a component part. Describe this attempt in a formal way as determination effort for $DPR\in DP$ (cf. section 2.1.).

7. DOWNWARD DESIGN PROBLEM RESOLUTION

Holistic *DPR* generation is incompatible with *DP* decomposition, so we try to cope with it in "as a whole" manner. When the desired product has no prototype, $DPR\in DP$ is provided only by initial $\{Rq\}\cup\{Cn\}\cup\{Rs\}$, which describes immediate *AE*. Hence, *DP* is unsolvable without its initial state. *DP*'s unsolvability has the second kind (cf. section 2.2.): the existing answer (a design) can be obtained through realization the conjugate problem concerning *DP*.

Begin the search for adequate conjugate problem with the search of a process conjugate (supplying the same answer) to $DPR\in DP$. Such process we have called the *process of DPR design implementation* and designated as DPR^*. Since the latter has *UL=3*, proceed to its two-stroke determination. On the stroke of virtual (or downward, ↓) determination, the tree of processes with virtually defined nodes is generated. On the second (or upward, ↑) stroke the processes in the nodes will be determined logically. The course of such determination is reflected in Fig. 10. In addition, this diagram is accompanied with protocol comments where the logical determination of each tree branch begins right after the end of its generation.

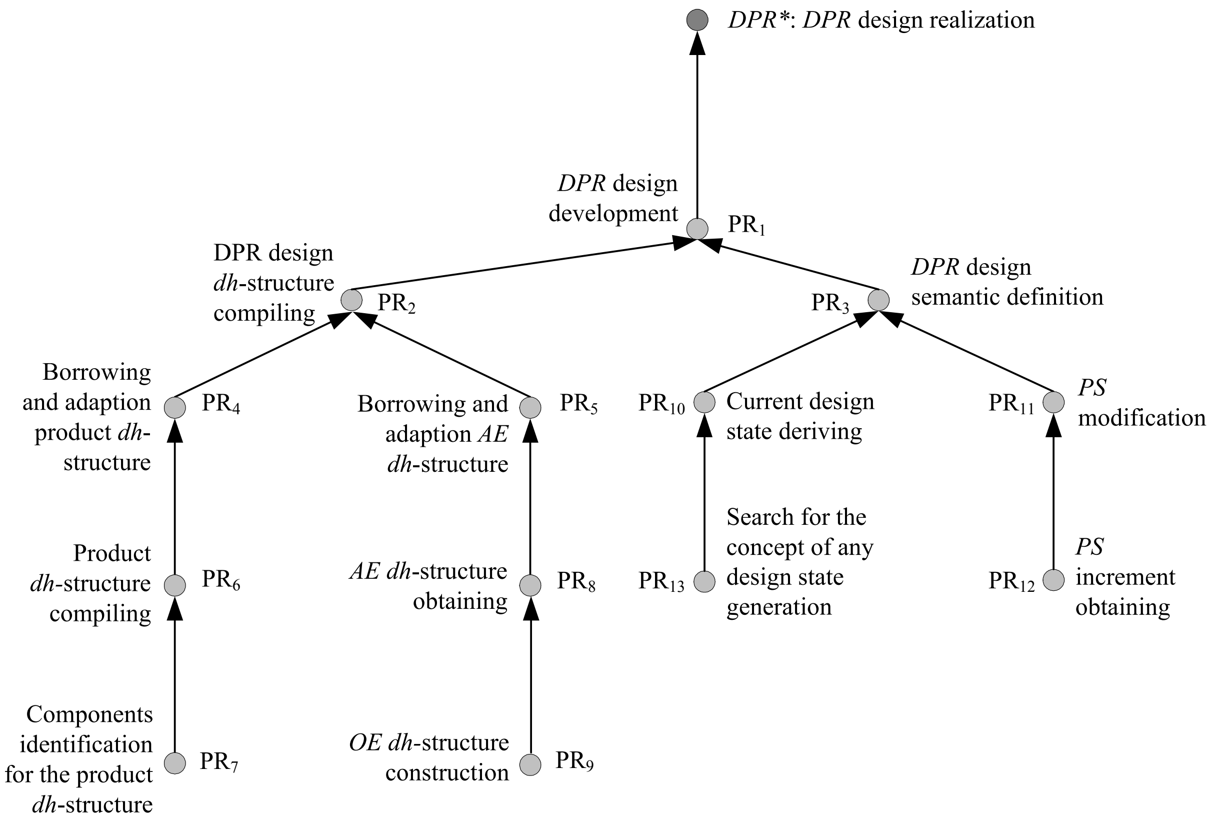

Fig. 10. *The tree of DPR^* determination*

7.1. The resolution protocol

DPR*: *DPR* design implementation.

↓D*=A procedure of *DPR* design implementation. Logical determination of ↓D* may start only after *DPR** has been provided with a design (not a model) of *DPR*. So, the scheme of a process that produces and delivers to *DPR**'s input such a design is generated.

↓PR$_1$: *DPR* design construction.
↓D$_1$=Integration of structural and semantic aspects of the *DPR* design.

Thereby we declare that the structure and semantics of *DPR* design will be derived independently and, with these operations completed, integrated by *PR$_1$*. The structural aspect of nonexistent design may be presented only in diachronic (*dh*) mode. Synchronous content of each diachronic structure element serves for its semantic aspect. The integral of those should deliver the *DPR* design semantics as such.

↓PR$_2$: *dh*-structure construction for *DPR* design.
The search for virtual *D$_2$* will be accomplished for the following reasons. In view of the accepted evolutional design concept, the structure of *DPR* design has to mirror concurrently both the product and *AE* design structures, preserving their isomorphism as well. It would be possible when *dh*-structures for the product and for *AE* designs are considered as building blocks (premodels) of *DPR* design structure – the scheme of *dh*-structure and the scheme of *dh*-decomposition respectively.

↓D$_2$=Integration the scheme of *dh*-structure and the scheme of *dh*-decomposition.

↓PR$_3$: regular semantics development for the *DPR* design.
↓D$_3$=Realization the product design progress concept – adaption the current design state to a new state of *AE*.

↓PR$_4$: deriving *dh*-structure scheme for the *DPR* design.
↓D$_4$=Borrowing and adaption the product *dh*-structure for the part of *DPR dh*-structure scheme.

↓PR$_5$: deriving *dh*-decomposition scheme for *DPR* design.
↓D$_5$=Borrowing and adaption the *AE dh*-structure.

↓PR$_6$: getting *dh*-structure for the required product design.
↓D$_6$= Coupling the scheme elements with relevant relation.
↓↑PR$_7$: identification of *dh*-structure elements for the product design (cf. section 4.3.).
↓↑D$_7$=Employing the product DPC.
↑D$_6$=*q*-hierarchy construction using the elements of the product design *dh*-structure (Fig. 5).
↑PR$_6$=↓PR$_6$
↑D$_4$=Using the product design *dh*-structure for the role of *DPR* design *dh*-structure scheme.
↑PR$_4$=↓PR$_4$

↓PR$_8$: deriving *AE dh*-structure.
↓D$_8$=Borrowing and adaption *OE dh*-structure.
↓↑PR$_9$: *OE dh*-structure derivation.
↓↑D$_9$=Using of *OE sh*-structure (cf. section 4.4.).
↑D$_8$=**&**-cube construction out of the elements presenting *AE dh*-structure (Fig. 7).
↑PR$_8$=↓PR$_8$
↑D$_5$=Taking *AE dh*-structure for the role of *DPR* design *dh*-decomposition scheme. The nesting relation between the incident intervals of **&**-cube is called off.
↑PR$_5$=↓PR$_5$
↑D$_2$=Substitution of terminals in *DPR* design *dh*-structure scheme by the scheme of its *dh*-decomposition.
↑PR$_2$: *DPR* design *dh*-structure obtaining.

The received *DPR* design *dh*-structure is represented by sixteen iteration of **&**-cube (Fig. 7): each of four *hyperperiods* (affords a design goal – PRT, ITM, COM or ART) consists of four periods (each period affords a design subgoal – qSYS, SYS, qDES or DES). Every period is represented by **&**-cube of intervals separated into *stages* (along X-line), *phases* (along Y-line) and *tasks* (along Z-line). *DPR* design structure is borrowed and adapted for the role of *AM* structure built for the product and *AE* designs.

↓PR$_{10}$: method M$_1$="next design *ML* synthesis" compiling.
↓D$_{10}$=Realization the concept of getting any design state.
↓↑PR$_{13}$: search for the concept of getting any design state.
↓↑D$_{13}$=Structure synthesis problem tackling (resolving for *SSP* its conjugate problem – determination the scheme of operation process associated with the requred product [7]).
↑D$_{10}$=Logical determination of M$_1$: feedback incremental synthesis.
↑PR$_{10}$: M$_1$ retention.

↓PR$_{11}$: obtaining a new state for *AE*={{Rq}∪{Cn}∪{Rs}}
↓D$_{11}$=Modification the *AE* state for an increment.
↓↑PR$_{12}$: increment deriving.
↓↑D$_{12}$= The query to designer.
↑D$_{11}$=Tradeoff the current *AE* state with an increment.
↑PR$_{11}$: deriving M$_2$="*AE* state synthesis".

↑D$_3$=Integration M$_1$ and M$_2$.
↑PR$_3$=↓PR$_3$
↑D$_1$=Assigning the pair (M$_1$, M$_2$) to intervals of *DPR* design *dh*-structure.
↑PR$_1$: Completion of *DPR* design construction.

↑$D \in DPR^*$=Traverse the intervals of *DPR* design structure with implementation in each interval M$_1$ and M$_2$.
↑$P \in DPR^* = C^H$.

In that way, the downward "solution" of design problem comes to the end. During the course of this solution, the design of holistic design process was obtained. Diachronic structure of *DPR* design has been borrowed by approximate models constructed for the product and *AE* designs. So, we came to the regular and isomorphic presentations for three entities: *DPR* design, *AM* for a product design and *AM* for the design of adaptation envirinment.

Physical replay of DPR^*, as *DPR* design implementation, should give the same answer that was required from initial *DP* – the desired product design. Functions and architectural features of $D \in DPR^*$ impart to it the status of special-purpose *OS* intended for design support (OS^D). The facilities for physical replay of $DPR^*=(OS^D, C^H)$ are called *Design Machine* (*DM*). Domain-independent *DM* remains unchangeable for all types, aspects, stages and objects of designing.

8. CONCLUSION

Analysis of paradigmant's states associated with CAD paradigm had indicated that possibility to receive holistic *DPR* is concerned with their directed modification referred to as CAD reengineering. The dominant bulk of this modification is related to the four key paradigmants:

- design progress concept (DPC);
- action system (*AS*);
- the problem of design problem (*DP*);
- design goal presentation.

As a result of modification activity, the above paradigmants have received the following interpretation:

- DPC: evolutionary synthesis;
- *AS*: its basic unit is *di*-processor $P_{\mathrm{w}}^{\mathrm{P_I}}$ where the working and information processors use a unified design language and may trade their roles;
- *DP*: instead phantom *DP*, the conjugate problem is under solution while the product design creation;
- creative design goal presentation is associated with the approximate model (*AM*) for a product design – the series of design states converging to an accepted one.

However, the completion of paradigmants correction with consolidated *DPR* obtaining and unified design machine construction did not imply yet the generation of conjoint design system.

While resolving *DP* in the downward mode, supporting *computer urged design* (CUD) and pretending for the part of invariant domain-independent core of various design systems, *DM*, nevertheless, is oriented to a greater extent to the *system* design problems support and to a lesser extent to deal with a great many of *applied* design problems, the experience and resolving aids for which are gathered in CAD.

So, we can safely assume that synergy of the downward and upward modes of design problem resolvind should be of sound benefit to design automation. Alike, the holistic design system should be compilled through cooperation of CUD and CAD facilities as two shoulders for one yoke of design computerization.

Thus, the attempt to reengineer the CAD paradigm with the aim of receiving holistic design process has revealed the outlines of another design paradigm, absorbing CAD as a constituent. It has got already the name – CUD. The rest is the subject for further research.

References

[1] Tomiyama, T. *Engineering design research in Japan*, Proceedings of the ASME Design Theory and Methodology Conference DTM'90. ASME, New York, pp. 219-224.

[2] Nauman, T., Vajna, S., *Adaptive system management*, Proceedings of the TMCE 2004. Millpress, Rotterdam, 2004, pp. 1183-1184.

[3] Sedenkov, V., *Evolutionary Design of Complex Objects*, Belarus State Polytechnic Academy, Minsk, 1997 (in Russian).

[4] Polya, G. *Mathematical Discovery*, New York – London, J. Wiley&Sons, inc., 1965.

[5] Linde, H., Hill, B. *Erfolgreich Erfinden-Widershruchsorientierte Innovationsstrategie fur Entwickler und Konstrukteure*, Hoppenstedt Technik und Tabellen Verlag Darmstadt, 1993.

[6] Clement, S., Vajna, S., Mack, P. *Autogenetic Design Theory – a Contribution to an Extended Design Theory*, Proceedings of TMCE 2002, pp.373-380.

[7] Sedenkov, V. *Product Structuring and Synthesis in Evolutionary Design*, Proceedings of the TMCE 2000, pp. 183-196.

[8] Andreasen, M. M., *The role of artefact theories in design*, Proceedings of the Work shop Universal Design Theory, Shaker Verlag, Karlsruhe, Germany, 1998, pp. 57-72.

[9] Restrepo, J., Christiaans, H. *Problem Structuring and Information Access in Design*, The Journal of Design Research, Vol. 4, Issue No.2, 2004.

[10] Rittel, H., Webber, M. *Dilemmas in a General Theory of Planning*, Policy Science, 4, 1979, pp. 155-169.

[11] Meijers, A.W. *The relational Ontology of Technical Artifacts*. In P. Kroes & A. Meijers (Eds.), The Empirical Turn in the Philosophy Technology. Amsterdam: Elsevier, 2000.

[12] Dorst, K. *On the Problem of the Design Problems – problem solving and designexpertise*, The Journal of Design Research, Vol. 4, Issue No.2, 2004.

[13] Lawson, B. *How designers think: the design process demystified* (2nd ed), Butterworth, London, 1990.