OBJECT-ORIENTED APPROACH TO DESIGN PROCESS MODELING

PhD thesis

Neven Pavkovic
Faculty of mechanical engineering
& naval architecture, Zagreb, December 2000.

SUMMARY

The subject of this thesis is the development of the object-oriented design process model framework. The proposed framework should be used as an open toolbox, independently of the design task class and the design process phase. In such an approach, every occurrence, relation and other real-world 'things' from the domain of the design process are attempted to be modeled as objects. It is assumed that the object-oriented methodologies can provide more appropriate and flexible ways of software system modeling than the other techniques. The design process is here viewed as a sequence of transitions from an initial state of data, constraints and goals to a final state: a complete description of the mechanical artefact being designed. These transitions are allocated to individual participants or teams, and individual or integrated computational tools or models. The main results of this research are identification and description of the basic structural, relational and behavioral entities of the design process model. The proposed model is built upon following basic structural entities: design parameter, product documentation object (any type of document), physical component object (part), functional component (organ), interface to external software tool, action, designer and the design task. These entities are modeled as the basic classes of the object-oriented design process model. The complex network of relations between design process model entities is analyzed as a central issue of the presented research. It turned out that a very large and complex object network could be efficiently modeled and managed in an object database environment. Thus, a design process model is built using a "bottom up" approach in which the basic structural entities and the network of relations are used as building blocks for more complex entities which model and control the design process flow. A design process is represented with design plan - a collection of nodes and their connections in a directed graph. The connections between nodes represent the information flow and/or the preplanned execution paths. Possible design plan topologies, the plan generation and exploitation issues are discussed, as well as the possibilities of implementing dynamic changes while the plan is being executed. A design plan node models one step of the design process, including: checking of preconditions, list of actions, checking the postconditions and deciding about the next step. Preconditions and postconditions include sets of constraints and rules. The design plan role could be viewed as "design information manager and process organizer". Through several levels of referencing, the procedures for executing the plan may access attributes or operations of any object that constitute the proposed representation. The structure of the proposed model is documented in the UML language, using "Rational Rose 2000" software tool. An example of "real world" implementation is realized in POET object database.

Keywords :
design theory, design process, computer-based design support, design process modelling, design information management, planning and workflow methodology, object-oriented technology, complexity management, UML, object databases

Table of contents

1. Introduction

1.1 Motivation

The existing CAD systems support the design process mainly only in geometric operations. They have no (or very little) support for managing and planning the design process. Considering all existing particular software tools that are in use in the design process, it is necessary to explore the principles and methods of their integration in complete (and more efficient) system for computer supported product development. Such systems are often named "design environments". The main goal of design environment should be to raise the level of design process organization and to improve the design information management. The approach presented in this work aims to introduce the object-oriented design process model as the kernel of "design environment" system. It is expected that an object-oriented approach to design process modeling could contribute to the flexibility of the system, making possible to implement and to combine several design process models and methods within the same framework.

1.2 Research goal and hypothesis

Engineering data structures are much more complex than in common business database applications. The associations between structures are numerous, and the same data structure can have many different roles. Hence, a design process topology is here considered as two networks of relations:

- Complex multi-level relationships between engineering data structures
- A network of sequence relations between design process steps, including the models of iterative processes

The thesis of the presented research is that a design process topology (as above defined) could be efficiently modeled with object-oriented methodology. This thesis is examined and validated on an example of the prototype model implementation, realized in object database. The structure of the proposed object model is documented in the UML language.
The goals of this research could be summarized as follows:

- to compare the potentials of an object-oriented approach to design process modeling with traditional relational database technologies used in management of engineering data in design office.
- to consider criteria for classifying the relations in engineering data structures
- to develop a design process model framework which could serve as the "integration platform" for various software tools being used in the design process
- the object-oriented design process model framework should be flexible as much as possible, enabling adaptation to specific application environments
- the global framework structure should enable permanent improving (upgrading) of model entities and their relationships

1.3 Research methodology

The research guidelines could be summarized as the following considerations:

- Treating (observing) the object-oriented design process model as an integral part of the entire computer support for product development process. In such approach the design should be treated primarily as the process of generating and processing information.
- Theoretical extraction of all things, events, phenomenon, (occurrences) and information as entities of the design process model.
- Mapping form entities to classes, defining the attributes, operations (methods), and relationships between objects (classes).

In the proposed approach, every occurrence, kind of action, set of information, relation and other real-world "things" from the domain of the design process, are attempted to be modeled as objects. In the presented research phase, the considerations are limited to recognition of basic entities, their attributes and relationships, while the necessary operations are only denoted. Proposed design process model is conceived as an "open toolbox" with packages of classes and objects. In such an environment the designer should mainly use the existing classes as templates and building blocks. The applied research methodology could be best described with scheme according to Duffy & Andreasen [3],[27] (figure 1).

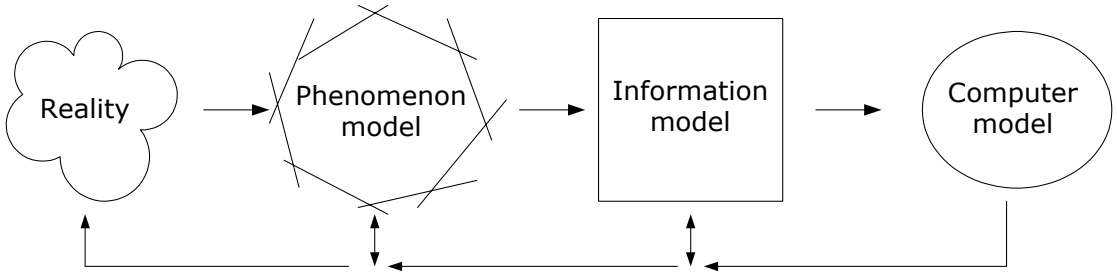

Figure 1: The research methodology

The path from left to right explains research carried out where phenomenon models are gradually formalized by means of information and computer models [27]. The path from right to left is verification, thus models are confronted with reality, i.e. empirical observations.

Object-oriented analysis begins here with an examination of the real-world 'things' that are part of the problem to be solved. Those things (which we will call objects or entities) are characterized individually in terms of their attributes (transient state information) and behavior (functional process information). In object-oriented terms, we discover and describe the classes involved in the problem domain, i.e. the reality and the phenomenon model. In parallel to these individual characterizations, we also model the links or collaborations between the problem domain's objects (and therefore the solution's classes). Object-oriented design then turns from modeling the problem domain towards modeling the implementation domain, i.e. the information model and the computer model.

1.4 General modeling issues

In the context of presented research issues, it is important to consider some questions about modeling:

- Which principles should be the guidelines for outlining (making the basic concept of) the model?

- How do we create the models, and which are the techniques for representing the models?

Very interesting discussion about these issues (and maybe some answers to above questions) can be found in work of Schregenberger [38]. Schregenberger cites Stachowiak which have introduced a convincing conception of models in [40]. According to Stachowiak, statements about *something* (about "originals") always refer to a *certain purpose*, present the original under *distinct aspects*, stem *from someone*, are directed to *somebody* and are embedded in a *specific context*. This general conception allows us to systematically characterize and discuss any kind of human expression. We use models to store, process and communicate knowledge. According to [38], the optimal handling of models is decisive for the success of any engineering brainwork.
Interesting observations about modeling of information systems in the domain of civil engineering can be found in [11] and [12]. In [11] Froese gives a survey and comparative analysis of several "core models" for design information processing.

1.5 Process representation methods

Process representation methods (for general or specific purposes) are the subject of many research projects. American National Institute of Standards and Technology makes efforts to develop a general purpose process specification language that should become a standard [20]. The goal of the NIST Process Specification Language (PSL) project is to investigate and arrive at a common, unifying model of process which will ultimately be suitable for multiple process-related applications, yet powerful and robust enough to meet each set of requirements. According to [20], twenty-six representations were analyzed to determine their applicability for representing a pre-determined set of process requirements. Five representations were found to play a supporting role in the other twenty-six representations that were analyzed. In many cases, these twenty-six representations integrated the concepts and constructs of these supporting representations to represent information requirements that the supporting representations captured especially well. These five supporting representations are:

- AND/OR Graphs
- Data Flow Diagrams (DFD)
- Directed Graphs (Digraphs)
- State Transition Diagrams (STD)
- Tree Structures

A first version of the Process Specification Language (PSL) is announced in [36].

The approach proposed in this paper uses directed graph for the representation of process flow. The other aspects of process representation are covered with objects connected (assigned) to graph nodes in various ways. These issues are discussed in chapter 6.2.

1.6 Related research projects

Similar treatment of OO formalism in representation of the design process can be found in the work of Gorti et al [14]. Their model is implemented as a layered scheme that incorporates both an evolving artefact and its associated design process. They define the design process with five primitive objects: goal, plan, specification, decision and context. Liang and O'Grady [22] have developed a concept of a design object that represents physical entities such as parts or components as well as non-physical entities, such as design history or vendor information. They build the design process model formalism on a definition of a design model, design objects and design methods.

2. An outline of the object-oriented design process model

This chapter discuss some basic issues about designing an outline of object oriented models and software systems. The research projects in this area are mainly focused on business systems. The available methodologies (although built for general purposes) are mainly assigned to business systems. Therefore, here are emphasized the difficulties arising in modeling the design process which is in some aspects fairly more complex than common business processes.

2.1 Mapping from real world to conceptual and object model

Meyer [26] defines Object-Oriented design formally as "the construction of software systems as structured collections of abstract data type implementations. According to [19], more informally, Meyer defines it as "the method which leads to software architectures based on the objects every system or subsystem manipulates (rather than "the" function it is meant to ensure)". An object can be viewed as the realization that certain knowledge and certain operations are conceptually related to each other, so that it makes sense to bundle them together.

To build a system with object-oriented approach, means to analyze the problem and to find the objects that should be included. Therefore, the first task is to extract and define the notions and events (phenomenon) that determine the design process.
The process of outlining an object design process model can thus be considered as mapping of phenomenon from the domain of the real world to the entities of conceptual and logical domain (Figure 2). This mapping is done by means of theoretical generalization and extraction. Entities from conceptual and logical domain are then being mapped to classes of object model. According to [8], the goal of object modeling is to have a "one to one" correspondence between entities and objects. The schema of this mapping process is shown on figure 2.

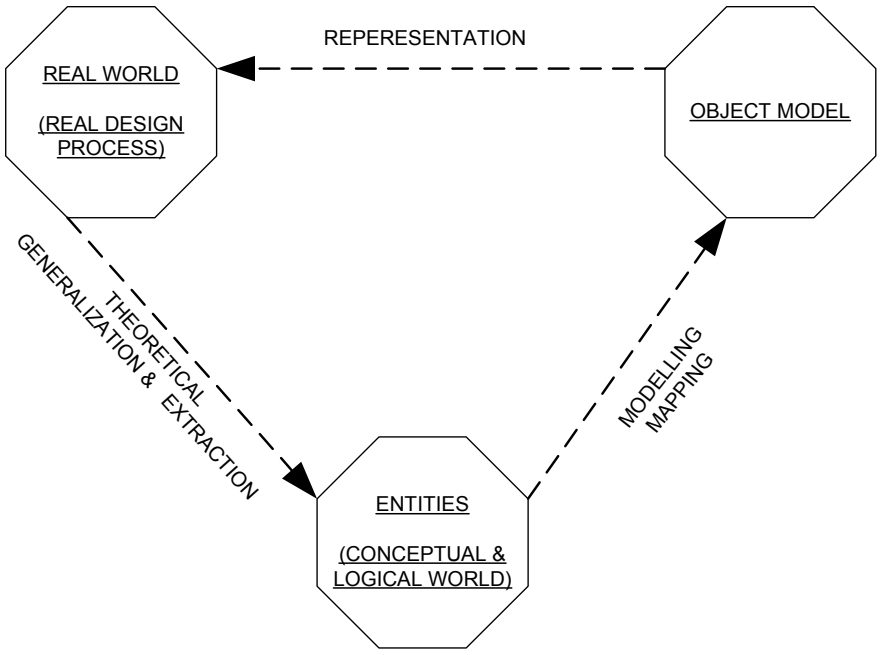

Figure 2: Mapping from real world to object model, through conceptual and logical world

2.2 The problems of conceiving the design process model in the conceptual domain

Could the existing general design theories or models serve as starting points for conceiving the concept of object design process model? Four of them have been considered – General design theory (Yoshikawa, Tomiyama [44]), axiomatic design theory (Suh [42]), General design process model of Hubka and Eder [18], and VDI 2221. Neither one of these theories and models doesn't consider all aspects of design process, [46] but the bigger problem is that there are mainly not oriented (focused) to information modeling. General design theory and axiomatic design theory are mostly highly abstract, not covering the organizational and teamwork aspects and issues at all. Hubka and Eder gave a model that is broader, but it is based mostly only on phenomenological description. Such an approaches aims primarily to give a deeper understanding of design, but (by author's opinion) couldn't give a significant contribution to modeling computer based organization and management of the design process in current collaborative environments. Of course, these important theoretical backgrounds should not be neglected - their ideas and cognitions must become the main principles and guidelines for extracting and defining the model entities as well as for establishing the model structure. Keeping this in mind, the presented research aims to develop a framework that could be able to include and integrate the different models or their parts (substructures).

Once outlined and established, the computer based design process model will be the subject of permanent improving and maintenance processes caused by new design science cognitions or changes in the environment where the design process proceeds. In the process of modeling, the design process should no longer be viewed as a static institutionalized structure, but rather as a dynamic network that is constructed in real-time as design goals, needs, priorities, and resources evolve [47].

Therefore, it is proposed to outline the global design process model structure as an "open toolbox". Such an approach should enable the designer to create his own classes and partial models of the design process according to his current needs. The goal is to develop a framework that enables us to model the design process (and to integrate used software tools) independently of design phase and class of design task.

It can't be expected that it is possible to build a model general enough to enclose all phenomenon variations in different real environments. Keeping this in mind, the presented research aims to develop a framework that could at least enable the use of different partial models (from design process domain) in an integrated manner.

A real system (real world domain) whose modeling is being considered here is a teamwork, computer supported design process which uses some kind of computer network (intranet) for team member collaboration and data share. Is it supposed that such environment intensively uses CAD software, databases, PDM and other kinds of software tools.

In outlining the design process model the focus will be in efforts to adopt the model as much as possible to purposes of computer application. In such approach the authors would not try to build neither primarily prescriptive or descriptive model, but some kind of a hybrid. The design process will be treated as information generation and transformation, estimating that the majority of the information is computer stored. In other words, the designing is treated as a process for which a computer support should be modeled by mapping a common working process (from the real world domain) to the domain of object-oriented software system.

Based on the presented approach, the entities of conceptual domain will be proposed. In the process of outlining the model structure, the efforts will be focused to make the model general enough for wide spectrum of application. In the same time the model should contain a rich set of

specialization possibilities, to be able to efficiently adapt to specific characteristics of particular application environment.
Let us compare the conceptual modeling (outlining) of common business processes and design process. The majority of business processes are relatively simpler in structure than the design process - therefore it is easier to extract their conceptual model from the real world domain. Most of the transactions and operations in business processes are well defined and determined. In the contrary, the big part of transactions in the design process are very difficult to predict, and impossible to prescribe. The main difficulty of this research, particularly in developing the entities of the proposed model, was (and still is) the lack of the general consensus in design terminology, taxonomy, and typology, as emphasized in [4] and [5]. Design science misses the "CAD theory" as pointed out by Akman, Hagen and Tomiyama in [1].

2.3 The outline of proposed system architecture

Based on all previous considerations and research work, the fundamental structure of the object-oriented design process model is proposed as on the Figure 3. According to [19], in a properly designed model, the classes will by grouped neatly into sub-systems that are loosely coupled and the linkages between classes in different sub-systems will be minimized. The system structure should contain stable abstractions (as much as possible) which should be isolated from those abstractions which will be more likely the subject of change. Guided by aforementioned principles, the fundamental structure is divided in four loosely coupled sets of entities (classes), as shown in figure 3:

- elementary classes – they model basic entities from design process domain – e.g. design parameters, designers, design tasks, design requirements, etc.
- classes that model the network of relationships between objects or their attributes
- more complex classes that represent partial design process models and/or design process execution flow
- the components of system architecture – "service classes" that don't model directly the design process, but serve to provide the functionality of the complete system

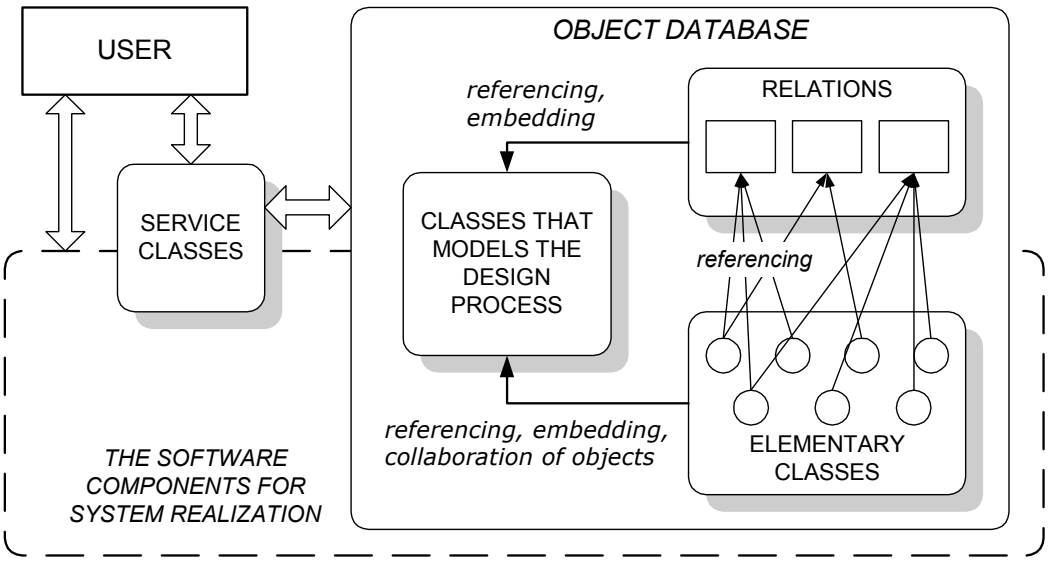

Figure 3: Fundamental structure of proposed object-oriented design process model

Such a structure promotes a bottom-up approach – the elementary entities and their relationships are combined and used in building the more complex entities to describe the design process.

Elementary classes model the basic notions (structural elements) from the domain of the "real world" design process. Those elements can be divided to "operators" and "subjects":
"Operators" are the objects which take an action(s) on "subjects". Examples of operators are designers and software tools.
The "subject" is here considered as any kind of information set which is being generated or transformed in the design process.
The relationships between entities in the design process model are far more complex than those in common business applications. Therefore it is assumed that it will be more suitable to model some of those relationships as classes.
Classes that models the design process will be the composite classes whose definitions will arise (be extracted) from decomposition of the design process viewed as sequence of actions in the process of information generation and transformation. Those composite classes will include the elementary classes and "relation" classes by referencing, embedding and by modeling the collaboration processes between objects.
Classes that model the design process should be defined in a way to enable the further adaptation to specific situations in particular application environment. Those classes should constitute an "open toolbox" for building the specific partial models, enabling the inclusion and flexible use of various design methods.
The three above discussed sets of classes are focused on static, mainly data, structures of the proposed model. To ease and improve their usage and manipulation we propose storing their instances (objects) in object database.
"Service classes" model the issues of the proposed system exploitation. This group of classes includes sets of operations that realize the functionality of the software system - e.g. interfaces and procedures for generation, manipulation and maintenance of model (system) elements. Service classes should have no instances - therefore they are not stored in object database. Particular entities in each of four main subsystems are specified in Table 1.

3. A proposal of entities in object oriented design process model

The following table contains a list of proposed entities, divided in four loosely coupled groups:

ELEMENTARY ENTITIES	RELATIONS BETWEEN OBJECTS OR THEIR ATTRIBUTES	PROCESS REPRESENTATION AND EXECUTION	THE COMPONENTS OF SYSTEM ARCHITECTURE
• design parameter • parameter's database • product documentation • product physical component • product functional component • design process action • software tool interface • design task • designer	• dependency relations between parameters • design structure matrix (design tasks' dependencies) • relations between sets of objects of the same elementary class • relations between sets of objects of different classes • expressions – relations between object attributes • design constraints • design decision rules – relations between conditions and actions	• design plan • design plan node • matrix representation of the design plan network • design process execution flow • design process capturing • design plan generation & execution • design plan sketch • knowledge bases about particular design processes	• "service" classes – the system functionality • object database dictionary • object database with design plan skeleton (framework) • created plans' archive • executed plans' archive • interfaces to knowledge bases and EDM/PDM systems • knowledge base about design support software tools

Table 1: Entities of the proposed object-oriented design process model

The following chapters contain the definitions and descriptions of each of proposed groups of elementary entities mapped to classes of object – oriented software system. Each entity is mapped to one class.

4. Elementary classes

4.1 Design parameter

Considering a design process as a process of information generation and transformation, it is assumed that the basic (simplest) entity of the design process model is a variable, often named a design parameter or design attribute.
Considering features of design process, especially in collaborative teamwork, one could notice the need for additional elements (besides value) that the design parameter class should encapsulate – e.g. "value status", references (pointers) to relevant knowledge, physical unit, etc.

The notion of "value status" can be particularly useful in iterative processes, where it enables the development of improved algorithms for solving such problems [10]. The "value status" should

also be one of the crucial elements in the development of software tools for teamwork support, to improve the communication on shared parameters. Such a system is currently under development as a part of presented research project. Besides "value status" it includes the management of "propositions and arguments". Each member of the design team (in the framework of his design task) has his own requirements and propositions for the values of shared design parameters. Such propositions are supported with arguments. All propositions and arguments, together with the decisions and the decision process flow are recorded in the database that represents the design history. The development of the described tool partially uses the algorithms described in [28]. The software tool will be realized in the web environment.

Design parameter, modeled as an object should encapsulate:

- value and SI unit;
- value status;
- value proposals and relevant arguments;
- references to relevant knowledge;
- the procedures for capturing and recording the proposals and arguments in collaborative processes (similar as in [28]).

The parameter *value status* is proposed as one of:

- determined, but could be changed;
- determined and fixed;
- assumed as: single value;
  value in discrete or continuous interval.

Any complex design project contains a huge set of data. It is supposed that it is not necessary to model all of design parameters as separate objects instead modeling them as attributes of objects. The examples of parameters that should be modeled as objects are:

- parameters that are “shared” among several other composite objects
- parameters that participate in any kind of relationships

4.2 Design parameter container (DPC)

Design parameter container (DPC) is an organized warehouse (repository) for the design parameters. DPC contains all design parameters modeled as objects, as also the operations for maintenance of its structure and for access control.
Engineering data structures are often more complex than those in business problems. When modeling the design world with objects, it is assumed that there will be many situations in which the same parameter will be the attribute of objects that belongs to the several different (independent or loosely dependent) classes. Even if those attributes could be set as “public” for the whole model, they still can have different values in particular objects. The value of every such “shared” parameter must be unique in time and design space, so it must be written outside the scope of the objects that must share its value. Following this requirement, the DPC can be seen as a pool of data shared among several objects that don’t belong to the same class hierarchy. In such approach objects of different classes that share the same parameter should have the reference (pointer) to the appropriate parameter modeled as object in DPC (Figure 4).

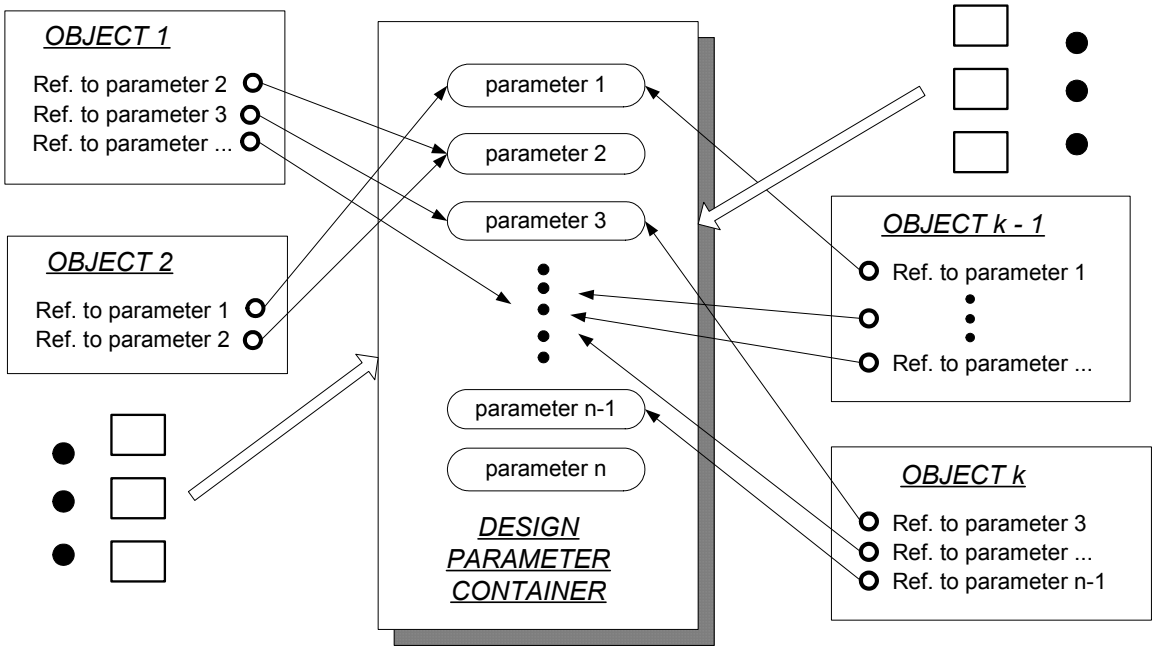

Figure 4: DPC as repository for parameters "shared" among objects of different classes

Data stored in DPC are design parameters modeled as objects. This data collection should be hierarchically structured and categorized, to ease the operations of searching, retrieving and other manipulations in the case of the large number of parameters stored. Every parameter has the unique name and storage address. Assuming that every parameter belongs to only one group or category, the structure of DPC can be realized and managed in a relational database, but the better solution would be an object database. In the process of the development of a new product, the DPC can be filled with parameters as the design evolves. For variant and adaptive design tasks, the DPC of previous designs can be used as a template, which can be modified and upgraded.

The proposed data structure is not intended to be a replacement for a complete product data model. The author is only emphasizing the necessity of managing the values of parameters shared among distinct classes and objects at one common place, (in one pool) in a universal manner. When the design is completed, the DPC contains only the subset of the information set about the product being designed.

4.3 Product documentation class

Entity "product documentation" models particular kind of set of written information about product being designed. This entity models the "existence" of document, i.e. encapsulates all notions and events from the life cycle of a particular document that contains a set of information about product. It also contains the interface operations for the transfer and the generation of the data form the information set of the particular document.

The "product document model" should include:

- document identification and classification
- a set of operations that makes the interface between "real document in electronic form" and object oriented design process model
- a set of references to design parameters whose values are the part of the documents' data set

- events (states) from the documents' life cycle: defining (opening); active in design process; in the process of control; in the approval process; filed away; etc.

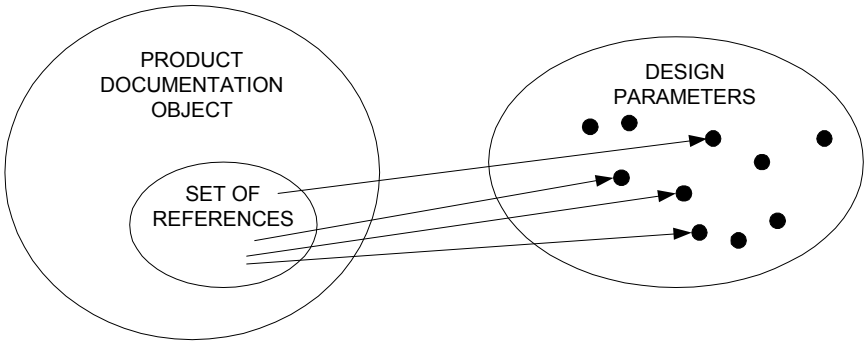

Figure 5: A set of references to design parameters as a subset of product documentation object

The primary role of the "product document" object is to encapsulate all above mentioned information and behavior aspects.

4.3.1 Attributes of "product documentation" class

PDM systems cover the majority of aspects and events from the life cycle of the product documentation. However, they don't include the design process model and mostly they don't support all the aspects of design parameter management in an appropriate way. In the design environment that already uses the PDM system, the role of "product documentation" object should be as an interface between PDM system and the design process model. In that context, some of the attributes of the "product documentation" class should be the same as the data in the PDM system.

In any case, the basic attributes of "product documentation" class should be:

- document identification code (identification & classification code)
- document classification code
- reference to the person (“designer object”) who is the owner and has all the responsibilities for the document in the life cycle of the document
- life cycle state of the document
- "path" to the electronic form of the document
- access privileges for particular design team members
- a set of references to design parameters whose values are the part of documents' data set
- the description (definition) of the document

To illustrate former discussions, let us consider one simple example of modeling the real world things and notions by mapping to logical and object domain. The Figure 6 emphasizes the semantic difference between product physical components and their design documentation. “CAD model of shaft assembly” is being modeled as (mapped to) one product documentation object. This object contains interface operations for transferring the values from CAD model to parameters modeled as objects and stored in design parameter container (DPC). The product documentation object contains references (pointers) to all relevant parameters in DPC. Other objects of design process model could access those parameters (and share their current values) by referencing them. The components of the “shaft assembly” are each modeled as separate

"product physical component object", which are stored in other database. Product documentation objects and product physical objects contains a set of references to each other. One product documentation object can contain information about several physical components as well as one physical component can be described (defined) within several documents.

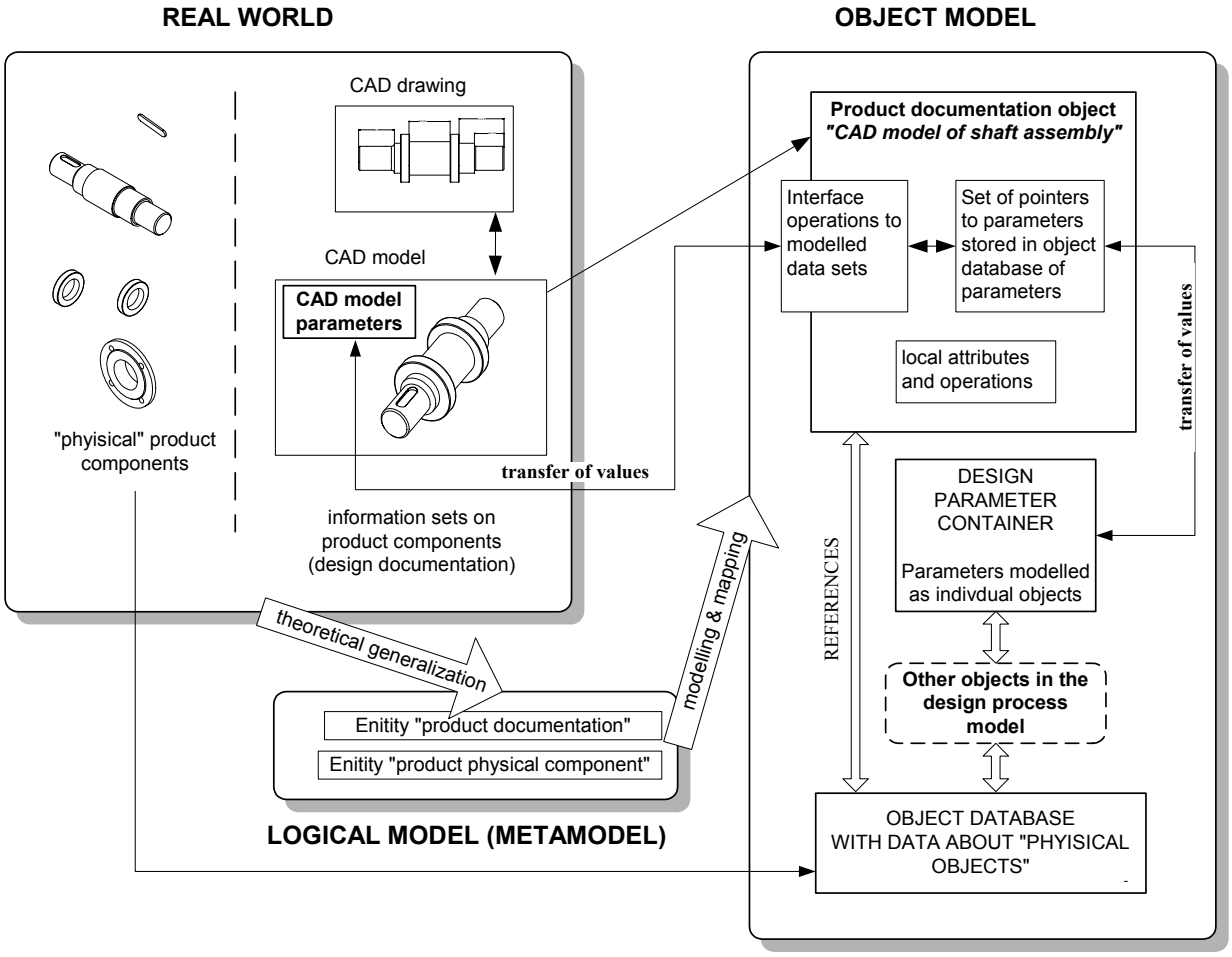

Figure 6: The difference between modeling "physical product components" and their documentation

4.4 Product physical component class

This entity models the existence of product components. It covers all the aspects and the data connected to physical domain. Some object-oriented approaches to design modeling make difference between "physical" and "non-physical" entities and treat them equally [14], [22]. There are also the approaches oriented (focused) only to modeling of "physical" entities (product components) and their interdependencies [50], [21]. The authors' opinion is that it is necessary to cover both aspects, and to provide the simple mechanisms of mapping from physical to information domain and vice versa. According to axiomatic design theory of Suh [42], the product documentation object can be considered as from process domain, and the objects which model the product components can be considered as from physical domain. Suh treats the design process as "zig-zag" mapping between four domains.

Most PDM systems cover the data about physical components as product hierarchical structures - the structures of assemblies, subassemblies and parts. This is only the one aspect (maybe the simplest one) of modeling the interdependencies in the physical components domain. From the design process point of view, more interesting are the functional relations between components - which physical components constitute the organs (functional components) and how the relations between those components fulfills the partial functions and the complete product function. To be able to model these relations it is necessary to have each product physical component modeled as object, or at least those components that are most relevant for product partial functions.
The objects of "product physical component" class should have references to relevant product documentation objects and to relevant design parameters.

4.5 Product functional component class

Product functional component class should encapsulate all the data relevant for building a functional structure of the product being designed. According to [36], many systematic for functional analysis and structuring exist in engineering design methodology and product development, but they do not seem to be very popular, especially in industrial practice. Pulm and Lindemann in [36] point out that functions are often understood as the solution-neutral abstraction of a component part or assembly. Their paper includes a survey and a critical review of functional analysis and structuring methods. They report on computer-based tools which allow modeling the abstract product in semantic networks, giving a standard in representing functions [2], [43]. But the method to build up the functional structure is still missing. According to [17] (cited in [36]) it has been shown that no genuine, abstract function structure exists.
As discussed in previous chapter, physical components and functions (functional components) are related, but this is many-to-many relationship, because a function may comprehend more components, and a component may fulfill several functions. At the beginning of the design task, those relationships as well as objects that participates in them, are mostly unknown for the new designs and relatively well known only in variant and adaptive design tasks.
All above mentioned facts makes defining and modeling the “functional component” class very difficult (and still “abstract”). There is no doubt that such a class should exist in the design process representation. More detailed proposal for the definition of this class is for the moment left for the future research efforts.

4.6 Action class

The entities being considered so far form the base of the "static" structure of the proposed design process model. They can be observed as the basic information building elements. One of the research goals in this thesis is to model the representation of the dynamics of the design process flow. Therefore an entity "action" is defined as the basic (simplest) element of the process flow representation.

Class "action" models the calls of operations of the design process object model. The actions are being created and planned before the design process, being realized in the duration of the design process. Actions mainly operate on product documentation objects. Actions should be modeled as objects in order to be used as elements in building the description of design process flow

dynamics. In such approach an "action" is here considered and defined in the context of generating, transforming and representing of information (only in the context of informatics), but not in the context of methodology of design and design procedures. Therefore, the primary criterion for proposed action classification is the type of "affecting" on design process model objects.

In the first prototype of the model, the following types of affecting are proposed:

- changing of attribute values in one or more objects
- searching and retrieving objects form object database
- the call of particular object operation (method)
- the call of "external" software tools (e.g. CAD modeler)
- creating the new instances of classes, or erasing the non-necessary instances
- viewing the state of one or more objects

A proposal of action class hierarchy is shown on Figure 7. This is also an example of specialization and generalization modeling. On the top of the class hierarchy is the abstract class "action" which should have no instances.

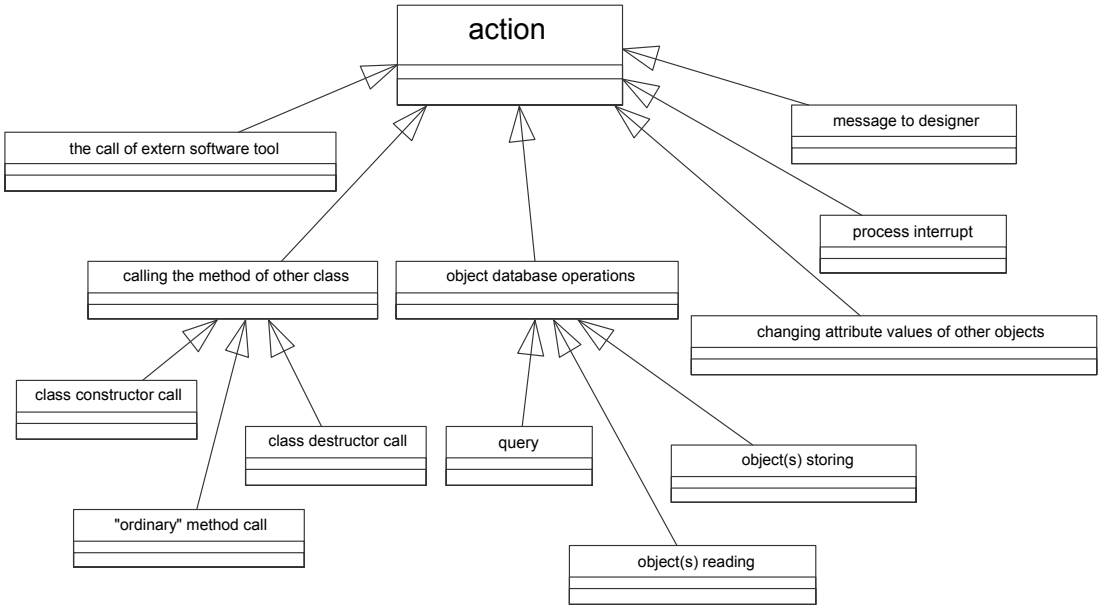

Figure 7: A proposal of action class hierarchy

Further research efforts will attempt to improve the "action" class and its specialization using the process specification language (PSL) specification in [36].

4.7 Software tool interface class

This class encapsulates a set of operations for the data transfer between the objects (classes) of design process model and the software tools that are not the part of the proposed object model, but they are of valuable use in the design process.

Every kind of software tool that is not the part of the object model, but is being used in a particular design process is here considered as an "external" software tool. The most common

examples of such tools are various numerical calculations written in procedural languages or in spreadsheets, expert systems, databases, etc. For the particular design environment such tools are very valuable and contain the knowledge collected through many years of constant development. Therefore, they should be integrated in the object oriented design process model. The author's opinion is that in most cases it will be cheaper and easier to develop an interface than to rewrite the tool as the part of the object model. The main task of such an interface should be to ensure the value transfer between design parameters from the object model to the appropriate variables of the software tool and vice versa, according to Figure 8.

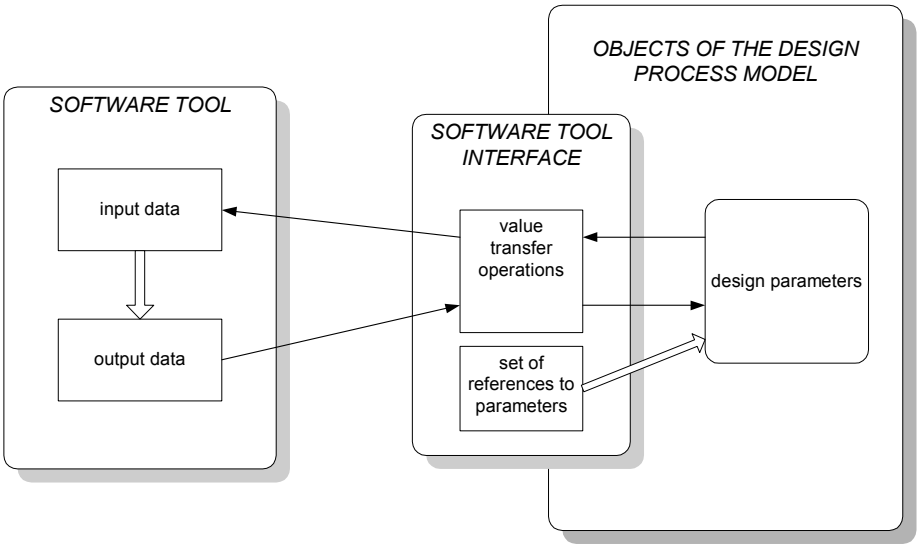

Figure 8: The parameter value transfer through software tool interface

4.8 Design task class

The research efforts in improving the organization of the collaborative design process considers the design task(s) from various viewpoints. Such approaches are focused to model and to manage the information flow, resources, responsibilities and timetables.
"Design task" class should model and encapsulate the information flows and the organizational aspects of the particular design task in the collaborative (team) environment.
The design task can be considered as the element of the design process decomposition in the teamwork environment. The need to define the design task as an object will most likely to appear in complex design processes where the assemblies and subassemblies or particular functional components are so complex that their design process is also being divided (distributed) among small teams (subteams). A survey of publications on workflow management may be found in [48]. The authors also propose the addition of a team concept to today's workflow management systems.

4.8.1 The design task content

The information that a design task should encapsulate may be divided in following:

- Task requirements and goals:
    - required functions and characteristics of the part of the product that is being designed in particular task

For variant and repeated designs:

- references to design parameters whose value or status should be determined (the parameters exist in previous designs)
- a list of parameters that should be created as new instances
- references to existing product document objects that should be worked on
- a list of product document objects that should be created

Organizational data:

- the identification code of design task, timetable, the responsible designer, the design task lifecycle status
- The description of information interdependencies with other design tasks and references to dependant design tasks

4.9 Class designer

This class encapsulates information and operations regarding one particular person from the design team. This class participates in modeling of the workflow management, responsibilities and the hierarchy of access control (to design documentation). A set of instances of "designer" objects models events and notions from domain of interactions between persons in the context of design process execution.

This object should contain references to:

- assigned design tasks
- product documentation objects and product physical and functional objects that are under the responsibility of particular designer

Existing PDM systems cover some of above mentioned aspects, but they don't support or model the communication and the decision process in the collaborative design. This is the area in which further research efforts regarding this entity should be focused, and this should be the primary role of "designer" class.

5. Relations in the object oriented design process model

According to the literature, in the design of object-oriented systems, the most interesting part are not the objects themselves, but the relationships among objects, and these relationships constitute a large proportion of the semantics of most systems [6], [9]. Therefore in the first phase of the presented research, the emphasis has been put on modeling the relations.

5.1 Classification of relations

It should be noticed that criteria for relation classification proposed in this chapter should be viewed just as the starting point and the proposal for further discussion in developing the kernel of the software framework under consideration.

5.1.1 Number of the relata comprehended within relation

In set theory, [23] the relation is defined as a subset of Cartesian product of two sets:

$R \subset AxB = \{ (a,b) \in R \mid a \in A, \; b \in B \}$

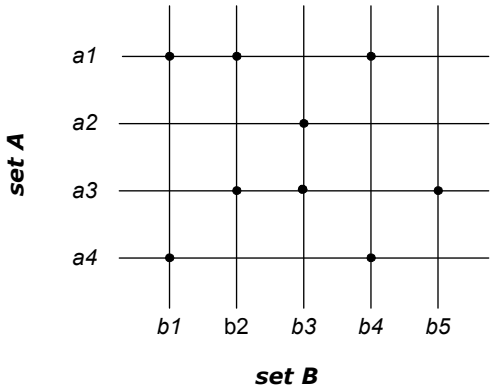

Figure 9: Graphical representation of relation

Relation between two sets is a ***binary*** relation (2 relata). In general, if there are more than 2 relata we have ***n-ary*** relation defined as:

> An ***n***-ary relation on sets $A_1, ..., A_n$ is a set of ordered ***n***-tuples $<a_1, ..., a_n>$ where a_i is an element of A_i for all $i, 1 \leq i \leq n$. Thus an ***n***-ary relation on sets $A_1, ..., A_n$ is a subset of Cartesian product $A_1 \times \ldots \times A_n$.
>
> An **ordered *n*-tuple** is a set of ***n*** objects with an order associated with them. If ***n*** objects are represented by $x_1, x_2, ..., x_n$, then we write the ordered ***n***-tuple as $<x_1, x_2, ..., x_n>$ **.**

Binary relations can be represented with connection matrices:

> Let R be a relation from $A = \{a_1, a_2, \ldots a_m\}$ to $B = \{b_1, b_2, \ldots b_n\}$.
>
> An *m* x *n* connection matrix M for R is defined by $M_{ij} = 1$ if $< a_i, b_j > \in R$
>
> $= 0$ otherwise

While the binary relations are relatively easy for representing and manipulating, a network of n-ary relations that dynamically change during the design process can be very difficult to model and manipulate in computer environment. The ultimate goal of the presented research is to find if the object-oriented environment really offers better modeling capabilities for such problems than traditional database technologies. This issue will be further discussed in following chapters.

5.1.2 Relation semantics in the context of the general object-oriented information modeling

When analyzing different information models one could find some variations in classification of relations. In the proposed model we will rely on classification according to the authors of Unified Modeling Language: *dependency, generalization, association and realization*, [6].
Dependencies represent using relationships among classes that states that a change in specification of one thing may affect another thing that uses it, but not necessarily the reverse.
Generalizations connect generalized classes to more specialized ones in what is known as subclass/superclass or child/parent relationships.
Associations are structural relationships among instances. Association specifies that objects of one class are connected to objects of another. Associations can be binary or n-ary.
Realization is a semantic relationship in which one classifier specifies a contract that another classifier guarantees to carry out. Realization is used in the context of interfaces and in the context of collaborations.
Further discussion in this paper will be focused mainly on modeling binary and n-ary associations.

5.1.3 Relation semantics in the context of the product and design process model

The presented research was up to now focused on generalization and association relations. A prototype of class hierarchy is established, but a lot of work has to be done to refine it, and to develop each subclass. More attention was paid to modeling associations. So far, in the context of design process model, the following semantics of associations have been explored:

- Dependency (algebraic (between design parameters) or information (between design tasks))
- "Appertain to" (affiliation, belonging) - (product documentation - product components, design parameters - design tasks, etc.)
- Sequence (precedence) (of execution - between design tasks, design process steps)
- Responsibility (designers, design tasks, design documentation)
- Hierarchy (components - subassemblies - assemblies)
- Constraints (parameters - requirements)

5.2 Modeling the network of relations between sets of instances of elementary classes

As previously described, the proposed model is founded on four loosely coupled sets of entities (classes): elementary classes, classes that models relations, classes that models the design process and the components of system architecture. In this chapter we will further discuss elementary classes and relations between them.
In the presented approach, the relations between objects of elementary classes are also viewed as objects. An instance of such "relation" class represents binary or n-ary relation between sets of instances of elementary classes. The relations modeled as classes in the proposed model are:

- *dependency relations* between parameters, and between design tasks,

- *binary association relations* between sets of objects of different classes, with various semantics
- *expressions* – relations between object attributes,
- *design constraints* – relations between design parameters and requirements, *design decision rules* – relations between conditions and design process actions

The idea further developed in the proposed model is to use the matrix form for representing sets of relations between objects of same classes, as also between objects of different classes.
In general, a relation can be established between every ordered pair of sets of instances of elementary classes. Upper table in Figure 10 shows such set of binary relations.
The main question here is, which of all those possible relations could be useful to model and to manipulate in the proposed system?
Some of these combinations were already the targets of thorough research efforts, like e.g. "Design structure matrix" - a relation of information dependency between design tasks [41], [10].
The marks have been put in the cells of table in Figure 10 for relations whose exploration and modeling could be useful by author's opinion. Here it must be emphasized that the mark in the cell tells nothing about the semantics of the relation, which should be further explored if the relation is found to be useful.
The table in Figure 10 gives us a compact survey of binary relations between large sets of objects, and can be considered as a good starting point for further discussions.

SETS OF ELEMENTARY CLASSES:		**DP**	**DT**	**PD**	**D**	**PPC**	**PFC**	**A**
design parameters	**DP**	x						
design tasks	**DT**	x	x					
product documentation	**PD**	x	x	x				
designers	**D**	x	x	x	x			
Product physical components	**PPC**	x	x	x		x		
product functional components	**PFC**	x	x			x		
actions in design process	**A**	x	x	x	x			x

Design parameter matrix:

		p1	p2	p3	p4	...	pn
parameter 1	p1		x		x		
parameter 2	p2	x					
parameter 3	p3						
parameter 4	p4		x				
. . . . .	. . .						
parameter n	pn			x			

Design structure matrix

Product hierarchical structure

Figure 10: A survey of relations between elementary classes

The semantics of "Design parameter matrix" can be twofold - algebraic dependencies between design parameters, and similarly as in Design structure matrix - the information dependencies.

5.3 Relations between sets of objects of the same elementary class

In the current phase of research, some of the relations shown in Figure 10 are modeled in the environment of object database. They are implemented as classes whose instances represent the rows of the relation connection matrix. Let us consider a connection matrix for relation from set A to set B (Figure 9). Both sets are considered as instances of classes in the proposed model. Such a matrix can be reduced to three columns (Table 2). The second column in such view contains a subset of elements of set B that are related to particular element of set A. The third column records the relation semantics, if there is a common one for that particular matrix row. Now we can model the matrix as a class (set of instances) in which every instance represents one particular row. Every instance of matrix class has one pointer to the instance of class "A", and a set of pointers to related instances of class "B". The proposed model gives a more compact representation of relation, instead an approach where each instance of class "A" has a set of pointers to related instances of class "B". In the second mentioned approach, every class should have several sets of pointers (the one for each relation in which that class participate). Moreover, having a class that represents the relation, the procedures of searching, retrieving and updating are easier to implement and maintain.

	b_1	b_2	b_3	b_4	b_5	b_6	....	b_n
a_1				x				
a_2			x			x		
a_3		x			x			x
....								
a_n					x	x		

→

	Pointers to related instances of class B	Relation semantics
a_1	b_4	$a_1 = (b_4)^2$
a_2	$b_3\ b_6$	$a_2 = b_3 + b_6$
a_3	$b_2\ b_5\ b_n$	…..
....		
a_n	$b_5\ b_6$	

Table 2: Reducing a connection matrix to three columns

The left part of Figure 11 is an UML notation for the example of the class that models a set of algebraic dependencies between design parameters (a relation between instances of the same class). The "parameter dependency matrix" class has two associations with "design parameter" class. The "left" line is "one to one" association modeled with one pointer, while the "right" line is "one to many" association modeled with set of pointers.

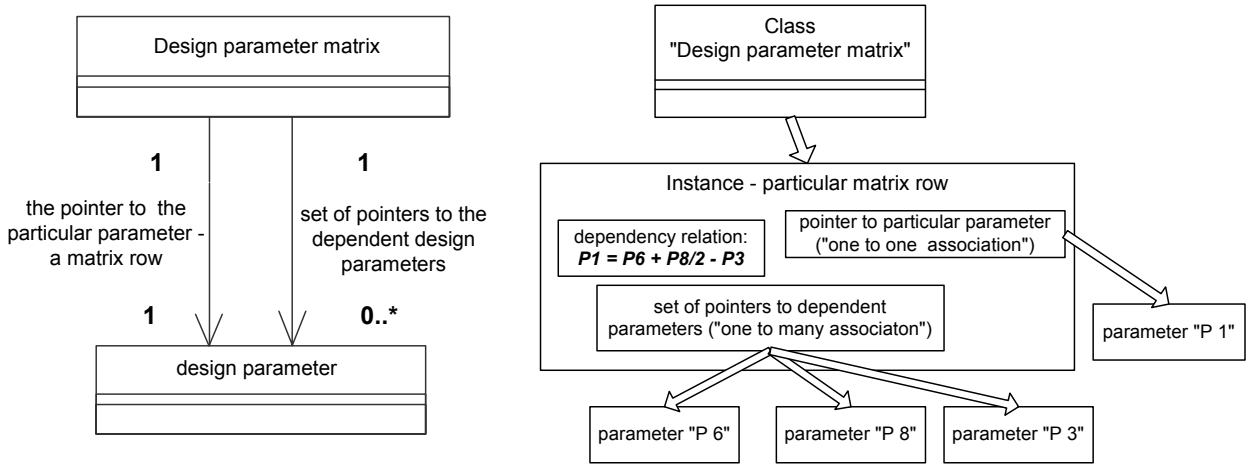

Figure 11: "Design parameter matrix" class

The right part of Figure 11 is an example of one instance of "design parameter matrix class. The scheme shows pointers to parameter objects that are related. The very similar model is developed for "Design structure matrix" class in which every instance has a pointer to one particular design task and a set of pointers to other design tasks that must contribute information to that task for proper completion of the design.

5.4 Binary association relations between objects of different classes

Let us consider modeling of simple binary associations between sets of objects of different classes. Semantics of those associations may be various: “appertain to”, belonging, membership, responsible for, etc. For instance, often there is a need to mark that one object “belongs” to other one, or that one object is a "member" of other object(s). For modeling one-to-many relation (association) between objects of different classes, it is sufficient that every object of one class has a set of references (pointers) to associated objects of other class.
According to Table 3, each object of class "A" contains a set of pointers to all associated (relevant) objects of class "B". Modeling of many-to-many relation (association) between objects of different classes was not considered in prototype model implementation.

Objects of class "A"	Objects of class "B" that are associated with objects of class "A"
a_1	b_2, b_3, b_{12}
a_2	b_3, b_4
a_3	b_2, b_{10}
…….	…….
a_i	$b_i, i \subset \{1, ...k\}$
…….	…….
a_n	……

Table 3: Modeling the binary association between objects of two different classes

An example of various binary associations is represented in UML notation on Figure 12. One designer has several assigned design tasks. In each of these tasks, a set of various product documents are being processed (created or modified). The designer is directly responsible for some of these documents. Each product documentation object has a set of references to parameters which "belongs" to that document, in other words whose values make the part of the document data set. Each design task has references to significant parameters that make the input or output data set for design task. This I/O set is also relevant for dependencies between design tasks in collaborative teamwork. A design task may also have a set of references to product physical and functional components that are being created or processed while solving the task.

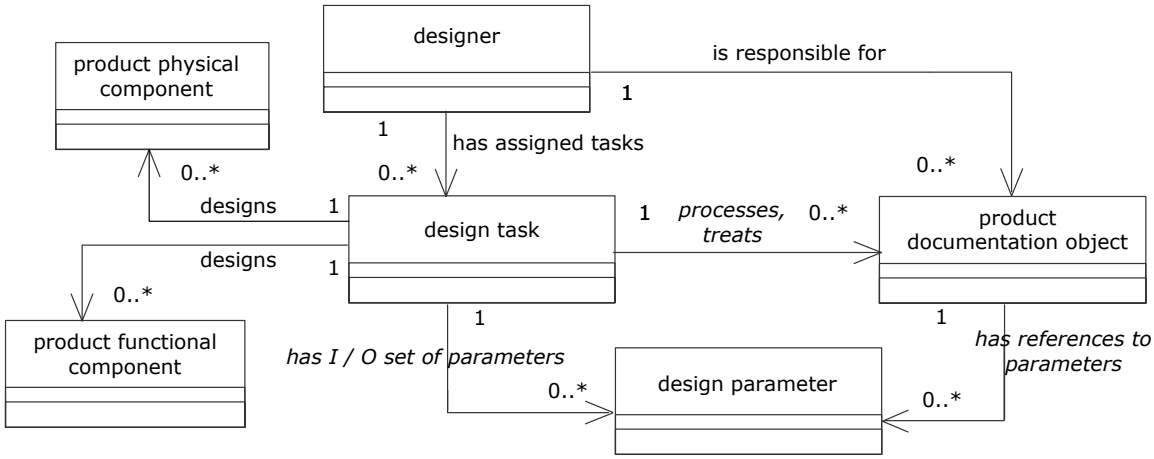

Figure 12: Association relations of "design task" and "designer" classes

5.5 Expressions as the elements of constraints and decision rules

The design process propagates through constraints' checking and decision making [45]. Entities which model the design process stage(s) should include and be able to control the processes of adding or checking the constraints as well as making decisions based on sets of rules. The proposed model introduces the "expression" as the basic element of the formal notation for design constraints and decision rules. The notion of "expression" is in the considered context analogous to expressions in most programming languages – a lexical element made up of tokens and operators.

If an expression is being interpreted as constraint or rule, then its value can be either true or false. Expression contains tokens and operators, where tokens are:

- the attributes of design process model objects
- numeric or character constants

The regular expression must contain at least two tokens and one operator.

The following is the BNF notation of the proposed “expression” syntax:

```
expression ::= <token> | <operator> | <token> { <operator> | <token> }
token ::= <object attribute> | <constant>
operator ::= <arithmetic operator> | <relational operator> | <logic operator>
               | <parenthesis, bracket>
object attribute ::=  #object_name.attribute_name
constant ::= <numeric constant> | <character constant>
logic operator ::= <&&> | <||> | <and> | <or>
arithmetic operator ::=   <+> | <-> | <*> | </> |
relational operator ::= <=> | <≠> |<<=> | <>=> | <<> | <>>
parenthesis, bracket ::=  <(> | <)>
```

Some examples of expressions:

```
#p1.v = #p3.v - #p5.v / 2.81
#p20.v = 'C4732' or  #p20.v = 'C1531'
#p4.s = 'proposal' or #p4.s = 'undetermined'
#p1.v > 10 and #p2.s = 'determined'
#p5.v > #p3.v  or  (#p2.v +10) < #p4.v
```

5.6 Design constraint class

The product being designed must satisfy a set of functional requirements. Functional requirements are satisfied indirectly by changing one or more design parameters, the physical form and operating conditions [49]. Functional requirements can be expressed as the functions of the design parameters, other functional requirements and various intermediate parameters [49]. Intermediate parameters and design parameters make up the vector of unknown parameters of design $\mathbf{P} = [p_1, p_2, p_3, ..., p_n]$. Functional requirements make up the vector $\mathbf{F} = [F_1, F_2, F_3, ..., F_n]$.
According to [49], the relationships between functional requirements and the unknowns can be expressed as:

$$f_i (F, P) = 0 \quad i = 1, k$$
$$g_j (F, P) \le G_j \quad j = 1, s$$

In the Design science, those equations are usually named "design constraints". Many of the equality constraints do not involve any functional requirements [49]. They impose a constraint relationship between the intermediate parameters and the design parameters. The inequality constraints include the limits of the values of functional requirements and design parameters. It must be emphasized here that constraint relationships are not the "design parameter dependency" relationships (considered in chapter 5.3), although in some cases they may be similar or may be expressed with the same formal equation. Each constraint modeled as object should have the pointer to "expression object" which contains the notation of constraint equation.
Constraints modeled as objects should contain the following attributes:

- description and references to relevant knowledge
- status - satisfied or not satisfied
- the status of checking – checking may be "active" or "passive"
- a record of every "checking" action performed in the design process – these can be very valuable for design intent and rationale capturing

5.7 Design decision rule class

Classes that are presented so far, model the basic data structures and their relationships – the "static" aspects of design process domain. From this point forward the dynamic aspects of design process progression will be discussed. Design decision rules are proposed as the basic elements for directing the flow of design process execution.
Decision rules are considered as relations between design process actions and values and states of attributes and objects relevant for making the decision. The design decision rule is conceived as "IF *condition* THEN *action*". The BNF notation for decision rule follows:

```
Decision_rule ::= if '(' <expression> ')' then
                    <sequence_of_actions>  (set row, line, range, array
              [ else
                  < sequence_of_actions > ]
              end if
sequence_of_actions ::= <action> { ';' <action> }
action ::=  #object_name.operation_name
expression  ::= <token> | <operator> | <token> { <operator> |
            <token> }
```

The syntax for “expression” is as given in chapter 5.5
The decision rule is being evaluated, and the value of expression is determined as “true” or “false”. According to expression value, the appropriate sequence of actions will be executed. These actions may be any of subclasses of actions discussed in chapter 4.6. This concept makes possible to call any operation (method) of any object that is the part of the system. Instead operation call, the action may also be the change of particular objects’ attributes. The decision rule is a complex (composite) object that references parameters or attributes of any other objects through expressions. Such an approach enables modeling of very complex rules with rich semantics available. Decision rule class may therefore be treated as the kind of framework for design process knowledge representation. Design decision rules should mainly be generated before the design process execution, as control structures in planning the design process progression. The issues regarding design process planning and the representation of the design process execution will be discussed in the next chapter.

6. Design process representation classes

In previous discussions we have proposed elementary classes (basic structural entities) which model engineering data structures and we have proposed the classes for modeling the complex network of relations. Those classes model mainly static aspects of design process. Modeling the dynamic aspects of design process is much more demanded task. As we have previously emphasized, neither one of the process representation methods could not satisfy all the necessary requirements for design process modeling. Some short surveys of several design process representation methods may be found in [29] and [39].
The issues of design process flow representation have been the subject of research interest at our design theory chair for several past years. The results of initial research phases were reported in [24], [25], [30], [31] and [33]. Those papers introduce "the design plan" as the model of the design process flow and execution control. The term "design plan" encompasses the data structures and operations that constitute the model of design process execution (progression) flow in the real time. Design plan is represented with directed graph. The design plan topologies of hierarchical tree and network (with one root node) were discussed and compared. The theoretical framework for the "tree structured" design plan and formal properties of the plan elements are discussed and presented in [33]. Later research efforts attempted to improve the model by introducing the network topology of the design plan, which has the same theoretical framework as the "tree structured".
The design plan is a formulated program of actions. Execution of the plan leads to the desired goal if and only if the preconditions for the plan execution are fulfilled. Action in the plan is an activity that generates effects if the preconditions for the action execution are satisfied. Design process is here treated as a sequence of transitions from initial requirements to a final design state [18]. Those transitions are observed as operators in the overall collection of information about product being designed. Each set of operators is represented as a node in the design plan.
The results of aforementioned research efforts were used as the basis for outlining the entities and classes which model the dynamic aspects of design process – the design process flow and execution control. In the proposed approach, the progression in design process from the initial requirements to the final design state is controlled by design decisions and by applying design constraints.

Very interesting considerations about planning the design process may be found in [13]. We must ask ourselves to what extent a design process could be planned? Real design process environments generate dynamic situations - they can change while the plan is being executed in a way that makes the plan invalid [35]. As if this is not enough, real environments may also change in a way that do not invalidate a current plan, but instead, offer new possibilities for action [35].

6.1 Design plan class

The role of this class is to model and manage the design process flow.
It is defined as a model of design process decomposition and the execution sequence. The purpose of the model is to organize and to try to predict the information flow as well as the sequence of execution of process steps controlled by conditions and constraints.
A design process is represented with the design plan class - a collection of nodes and their connections in a directed graph, recorded in adjacency and incidence matrices. The connections between nodes represent the information flow and/or the sequences of node execution. The nodes of a design plan are objects, created in a process of generating a plan and stored in the object database. To "execute" a node means to perform the associated operations. The calls of operations are realized with referenced "action" objects. Nodes and relations should be classified in order to implement different behaviors. Templates of such objects are stored in object database, to be at disposal in the process of generating a design plan. In the processes of generating and executing a design plan, a full access to all basic structural elements (classes) must be provided.
To summarize the above discussion we may say that the design plan structure should fulfill two main requirements:

- to determine (or predict) the execution sequence of operators in the design process domain, or in other words to model the progression in the process,
- to provide the mechanisms for organizing and controlling the data transfer between various kinds of software tools and various sets of design process model objects

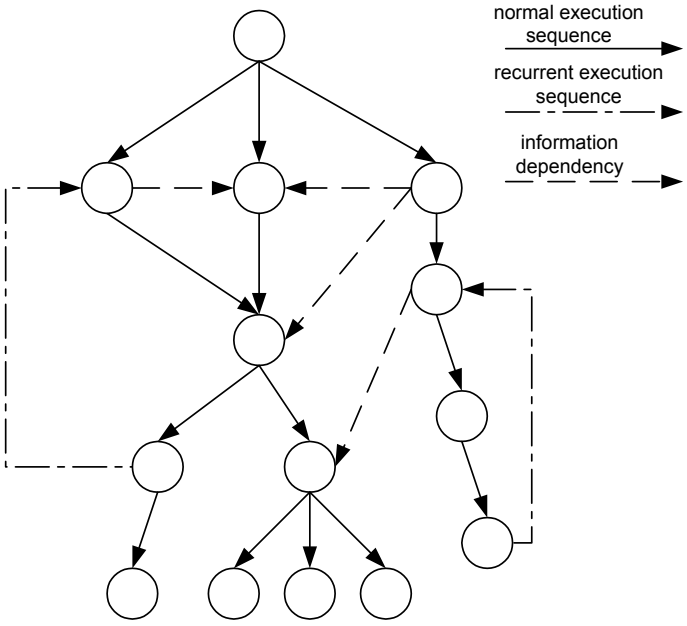

Figure 13: An example of design plan representation with directed graph

Figure 13 shows an example of graphical design plan representation. The connections in directed graph are classified as:

- "normal" execution sequence (points from current node to next node)
- "recurrent" (repeated) execution sequence (points from current node to one of previously executed nodes) – this is a kind of prototype model of iteration in design process
- "information dependency" connection – indicates that data sets (of objects that are referenced from connected nodes) are directly or indirectly dependent

The structure of design plan directed graph is recorded in the adjacency matrix and in incidence matrix. Those matrices are modeled as classes, analogous to other relations between sets of objects of the same class – according to scheme in Table 2. Adjacency and incidence matrix are the basic data structures for controlling and managing the design plan execution process. Issues of design plan execution process are in more detail discussed in [31], [33] and particularly in [32] and [24].

6.2 Class "design plan node"

In the proposed model, the design plan node is defined as a combination of set of actions A_j which transforms an object O, being in a state S^O_i, in the next (new) state S^O_{i+1} :

$$E = \{ A_n : S_j (A_j (S^O_i) = S^O_{i+1} \}$$

This definition can be extended to a set of objects, i.e. a set of actions transforms a state of each particular object in a new state. In such a process actions may generate new objects in the domain of the design process model, and furthermore, actions may change the state of newly generated objects.
The design plan node models one step of the design process, including: checking of preconditions, list of actions, checking the "postconditions" and deciding about the next step. Preconditions and "postconditions" include sets of constraints and rules, also modeled as objects. Constraints and rules include references to design parameters and attributes of all classes of objects that constitute the design process model. The design plan node execution process is represented on Figure 14. This block diagram also includes the schema of references from design plan node to other objects of the proposed system. Those references make the design plan node the most complex class in the proposed system. Through several levels of referencing, the procedures for managing the node execution process may access any particular object in the system. In such an approach the attributes of any object can be changed, as well as any object operation can be called (executed). Furthermore, by executing (calling) "constructor" action subclass, new objects can be generated.

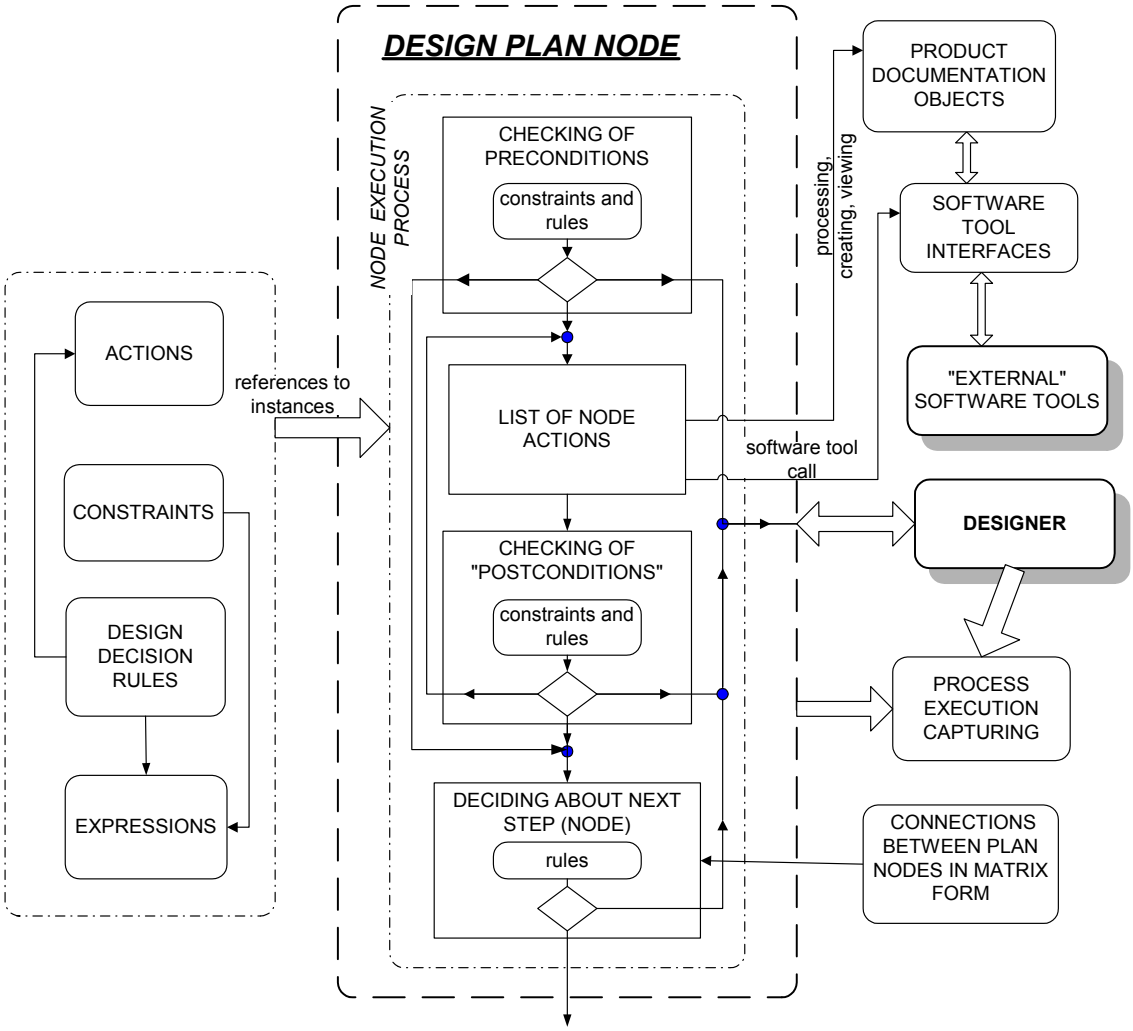

Figure 14: Design plan node references and execution process

Proposed definition (model) of design process stage is in one part similar to "design workspaces" proposed in [15] and [16]. The "design workspace" is focused primarily on geometric modeling. The model proposed in this work is primarily oriented to improve the design process organization and to integrate various software tools used in the process. In other words, this research work is focused primarily to represent the design process in the context of information processing.

According to definition, actions in the design process stage change the state of one or more objects. The necessary condition for action execution is, consequently, the existence of the particular object. Besides, the object should be in the state required by the action to be able to execute. Therefore, the design plan node should include the operations for checking the preconditions necessary for action execution. To plan the process means to determine the goal state of each particular stage. These goals could be considered as the set of conditions that must be satisfied. Consequently, the design plan node should contain the operations for checking the goal state ("postconditions").

In the proposed system, preconditions and goals are modeled as required initial and required final state of objects being processed in the particular design plan node. Such a model of the

design process stage contains a set of actions which perform the transformation form initial state to final state of objects. The examples of transformations could be:

- changes of object attribute values
- establishing or changing the relationships between objects
- the call of object operation(s) (methods) that changes that particular object state or changes the state of other objects
- the call of "external software tool"
- the generation of new objects, including the determination of their initial state
- querying: views and analyses of sets of attributes and objects

The design plan node controls (manages) the processing of different object classes. The main issue here is how to record the sequence of processing, and to record which objects should be processed. The plan node contains sets of references (pointers) to objects that should be processed. The sequence of processing could be recorded in a form of table where each reference has its ordinal number. Let us name this table "The node execution agenda". The proposed approach enables easy changes of execution sequence, as well as references to objects that should be processed.

pointers to objects that should be processed	ordinal number of processing
p [1]	3
p [2]	2
p [3]	1
p [4]	5
p [5]	4

Table 4: An example of "agenda" for processing the objects referenced from design plan node

Such a structure makes possible to develop the procedures for dynamic modifications of the design plan at the execution time. For example, an action (procedure) from one node could change the "execution agenda" in other node, or it can even change its own agenda. A kind of dynamic planning can be implemented, but constrained to internal process in particular nodes, without impact on decision process on choosing the next node (which is much more complex). This is a very important issue for further research, because such possibilities could bring the proposed model much closer to the reality, i.e. the human abilities of planning and adapting a finished plan to the new (unplanned) situations that can occur while executing.

7. Implementation and exploitation of the proposed design process model

The techniques of object oriented modeling and design have been intensively explored in last ten years, and it seems that they finally reached a high level of maturity. These days such techniques are dominated by the Unified Modeling Language (UML) [6], which is going to be a standard modeling language for object-oriented development. Therefore, the UML is chosen to be an implementation environment for the proposed design process model, which is completely developed and documented in UML.

UML model of the proposed system is (with aid of software tool) mapped to object database dictionary which contains definitions of all design process model classes listed in first three columns of Table 1. In other words, object database dictionary constitutes a framework of the object-oriented design process model. The database dictionary can serve as the open toolbox – the designer can easily create new instances of existing classes. Furthermore, "templates" of design methods and procedures can be modeled as composite classes and integrated in the design process model framework.

The prototypes of the proposed classes are realized in C++. The object-oriented database maps objects that has been created in C++ or Java onto objects in the database. This kind of direct mapping enables the database schema to be generated automatically [34]. Even in the case of highly complex object models, there is no difficulty writing them to the database. The database "knows" what an object is and also recognizes the relationships (references, pointers) between objects. These are simply stored together with the objects themselves, and are therefore reproduced easily at any time the data is required. For instance, if one object is retrieved from the database, all referenced objects (from that object) can be automatically retrieved, if demanded. "One-to-many" relations are implemented with "set of pointers" structure – one object has a set of pointers to all related objects.

The process of using a proposed system in modeling the computer-based support for the particular design process is shown on Figure 15. This block diagram also proposes a scheme for integration of various types of software tools used for design process support.

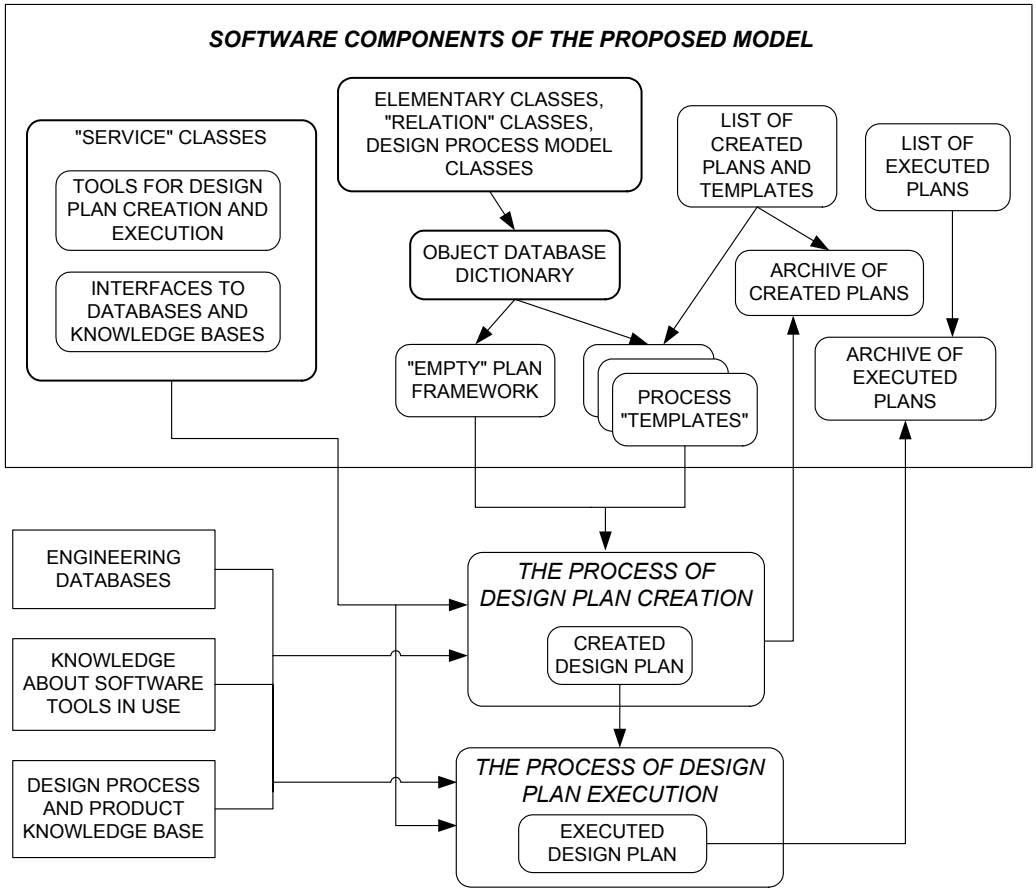

Figure 15: The "software components" and the usage process of the proposed system

Operations of creating and executing the design plan should be realized as "service" classes (the “fourth” group of classes listed in Table 1). These are very complex procedures as they must ensure the highest possible level of accuracy of the created plans. The design plan node execution process uses and calls the "external software tools". In the process of creating and executing of design plan, the designer must have on disposal all the available knowledge about "external tools" and “action” classes – which could be stored in separate database. Similarly, while planning or executing the design process, the designer should have interfaces to engineering databases (with product data), as well as to knowledge bases about design process and products being designed. Those interfaces should also be realized as “service classes”.
The design process structure can be decomposed to a hierarchy of design tasks and subtasks. Therefore, it is not necessary to define a design plan at the highest (most complex) level of design task abstraction. Computer based support for complex design tasks can be modeled as a hierarchy of plans and subplans. The design process control at higher abstraction levels can then be left to the designer. Previously completed and tested plans should be stored in a separate database (archive) to be available for reuse or to be used as the subplans in a more complex plan being created.

7.1 Case studies and future research issues

In order to decide the best strategy for further development, the future research efforts will firstly be directed towards the final implementation and evaluation of the proposed system elements and structure.
In this phase of research, case studies of the proposed system implementation have been considered only theoretically, based on the author’s experience in integrating the computer-based design support tools in design office of the medium size electrical machines factory. The design process in mentioned office includes typical variant and adaptive design tasks assigned to small design teams. It turned out that such design tasks could be efficiently modeled in the framework of the proposed system. More complex case studies are expected to be done in the next research phases.

One part of future research efforts will be directed towards exploring modeling of n-ary relations between various classes. A common representation of n-ary relation is a table in a relational database. The structure of tables and relationships in relational databases is relatively static, and not easy to update – usually requires a lot of programmers work. It is expected that modeling of n-ary relations in object environment will give more flexible structures. In the design process many of the data structures and relations are being created for the first time, in the ways that are often difficult to predict.

The proposed object structure is still a scaled prototype model. A lot of work remains to be done:

- refining the proposed structure, establishing class packages and hierarchies
- defining and designing attribute sets and operations in classes
- defining “use cases”, defining and diagramming collaborations between objects
- designing “main” system operations that have the threads of control
- developing interfaces to commercial object and relational databases

8. Conclusions

One can ask a question is it possible to develop a general (widely accepted) computer based design process and information management model? Obviously, it is a very difficult and complex task.

The presented research attempts to develop a kernel of object-oriented design process model. In the first phase of the research, the emphasis has been put on modeling the relations between objects, as they constitute a large proportion of the system semantics. The results are promising, there are many advantages comparing to traditional relational database technologies. Some of the proposed ideas offer more flexible data structures that are easier to create, update and maintain. Also, the object model is closer to the reality, therefore it should be easier to understand and learn for users. The very important fact is that object database recognizes and easily reproduces at any time the relationships (references, pointers) between stored objects. This fact enables the management of very complex and huge relation networks. Efficient modeling of the design process relation network topology was one of the primary goals of the presented research.

The main role of the proposed "design plan" is to integrate computer supported activities from lower complexity levels, forming the blocks for building a support for higher level activities. In other words, the proposed model may be viewed as "design information manager and process organizer".
The author's opinion is that applicability of the proposed model will strongly be influenced by a class of design task. The design process is sometimes far too complex to be easily planned. The research started from the assumption that the proposed model could be efficiently used to support design processes for variant and adaptive classes of design tasks. This has to be proven in the future research.

The research projects in the area of computer-based modeling of product development process were mainly focused on issues that are islands in the area (sometimes isolated). The attempts of development of general integrated frameworks were numerous, but it seems that there are still no good results, i.e. widely acceptable and usable systems. It is obvious that such a task is very huge and complex. It is difficult to believe that isolated research teams could come to widely acceptable solution. The presented work is also only a small portion of the job that has to be done – just "the top of the iceberg".

The benefits of object oriented methodology and UML raises new challenges. Using an UML, a distributed design community could easily collaborate in developing the design process model. Maybe this fact could initiate the process of creating a theoretical framework for integrated computer design process support that will be widely accepted in a design community. Or at least a prototype of such a framework could be developed for education purposes. One of the basic conditions for the success of such projects is the development (and the existence) of widely accepted product development process ontology.

Presented research proposes a basic framework which could serve as an open toolbox. The concept of loosely coupled groups of classes should provide a certain degree of flexibility. By using, combining and upgrading the proposed elements (classes) the designer should be able to

create and build his own partial models of design process, according to his current needs. To reach such a complex functionality of the system, the following steps are required:

1. Broad discussions and widely accepted definitions of basic classes – the definitions and the realization of the basic classes should be independent of the implemented design process model and design information management model.
2. Developing and inclusion of other design process models beside currently implemented "state-action" model.
3. Further research in modeling and generalizing the algorithms of "service classes" – the quality of which may have the key role in system's usability.
4. UML language could become (or should become?) a common communication platform for a distributed design community in such collaborative processes.

Considering a collaborative project in defining a common design process representation, the benefits of using the UML as implementation platform could be summarized as:

- All members of a development team use a common language.
- Confusion over terminology and requirements is averted.
- Transition between planning and programming is shortened due to a clearer understanding between designers and implementers.

9. References

[1] Akman V., ten Hagen P.J.W., Tomiyama T., A fundamental and theoretical framework for an intelligent CAD system, CAD, Vol. 22, No. 6, pp. 352-367., 1990.

[2] Ambrosy, S.: Methoden und Werkzeuge für die integrierte Produktentwicklung. Aachen: Shaker 1997. Zugl. Muenchen: TU, PhD Thesis. 1996.

[3] Andreasen M. M., Duffy A. H. B., Mortensen N. H., Relations in Machine Systems, Proceedings of WDK Workshop "Product Structuring", Delft University, 1995.

[4] Andreasen M. M., Wognum N., Considerations on a Design Typology, Proceedings of 3rd International Workshop IPD 2000, Otto-von-Guericke University Magdeburg, 2000.

[5] Andreasen M. M., Wognum N., McAloone T., Design Typology and Design Organisation, Proceedings of 7th International design conference – Design 2002, Design Society, FMENA, Dubrovnik, 2002, Vol. 1, pp. 1-7

[6] Booch G., Rumbaugh J., Jacobson I., "Unified Modeling Language User Guide", Reading, Addison Wesley, 1999.

[7] Clarkson P. J., Hamilton J. R., 'Signposting', A Parameter-driven Task-based Model of the Design Process, Research in Engineering Design, Vol. 12, No. 1, pp. 18-38, 2000.

[8] Eckel B., C++ Inside & Out, Berkeley, Osborne McGraw-Hill, 1993.

[9] Eliëns A., Principles of Object-Oriented Software Development, Reading, Addison - Wesley, 1995.

[10] Eppinger S. D., Whitney D. E., Smith R. P., Gebala D. A., A Model-Based Method for Organizing Tasks in Product Development, Research in Engineering Design, Vol. 6, No. 1, pp. 1-13, 1994.

[11] Froese T., Models of Construction Process Information, Journal of Computing in Civil Engineering, ASCE, 1995

[12] Froese T., Rankin J., Representation of Construction Methods in Total Project Systems, 5 th Congress On Computing On Civil Engineering, Boston, ASCE, 1988

[13] Giaopulis A., Schlüter A., Ehrlenspiel K., Günther J., "Effizientes Konstruieren durch Generierendes und Korrigierendes Vorgehen", Proceedings of 10th ICED conference in Praha, Volume 2, Schriftenreihe WDK 23, Praha, 1995, pp. 477-483.

[14] Gorti S. R., Gupta A., Kim G. J., Sriram R. D., Wong A., An object-oriented representation for product and design processes, CAD, Vol. 30, No. 7, pp. 489-501., 1988.

[15] Grabowski H., Lossack R.-S., Weis C., A Design Process Model based on Design Working Spaces, Proceedings of the IFIP TC5 WG5.2 International Conference on Knowledge Intensive CAD, Vol. 1, pp. 245-262, Helsinki, Chapman & Hall, 1995.

[16] Grabowski H., Lossack R.-S., Knowledge based design of complex products by the concept of design working spaces, Proceedings of the IFIP TC5 WG5.2 International Conference on Knowledge Intensive CAD, Vol. 2, pp. 79-98, Pittsburgh, Chapman & Hall, 1996.

[17] Hubka V., Theorie technischer Systeme, Berlin, Springer, 1984.

[18] Hubka V., Eder W. E., Design Science, London, Springer, 1996.

[19] King G. W., Object-Oriented really is better than Structured, web page: www.cs.uwa.edu.au/Why OOP.htm, 1995.

[20] Knutilla A., Schlenoff C., Ray S., Polyak S.T., Tate A., Cheah S. C., Anderson R.C., "Process Specification Language: An Analysis of Existing Representations," NISTIR 6160, National Institute of Standards and Technology, Gaithersburg, MD, 1998.

[21] Korpela T., The Role of Object-Oriented Analysis in Modelling of Technical Processes, Proceedings of the 12th International Conference on Engineering Design ICED 99, Vol. 2, pp. 853-856, WDK, 1999.

[22] Liang W.Y., O'Grady P., Design With Objects: an Approach to Object-Oriented Design, CAD, Vol. 30, No. 12, pp. 943-956, 1998

[23] Lipschutz S., "Theory and problems of set theory and related topics", McGraw-Hill, 1964.

[24] Marjanovic D., Implementation of Expert Tools in the Design Process, Ph.D. thesis, University of Zagreb, Faculty of mech. engineering & naval arch., 1995. (written in Croatian language)

[25] Marjanovic, D. (1997) 'Design Process Representation in the Developmnet of Design Support Environment', Proceedings of International Conference on Engineering Design, ICED 97, pp. 2/705-2/708, WDK, Heurista.

[26] Meyer B., Object-Oriented Software Construction

[27] Mortensen N. H., Design modeling in a Designer's Workbench, Ph.D. thesis, Lyngby, Technical University of Denmark, 1999.

[28] Nagy R. L., Ullman D. G., Dietterich T. G., A Data Representation for Collaborative Mechanical Design, Research in Engineering Design, Vol. 3, No. 4, pp. 233-242, 1992.

[29] Park H., Cutkosky M. R., Framework for Modeling Dependencies in Collaborative Engineering Processes, Research in Engineering Design, Vol. 11, No. 2, pp. 84-102, 1999.

[30] Pavkovic, N., (1997) 'Defining and Generating Design Plans - an Approach to the First Phase in Exploitation of an ICAD System', Proceedings of International Conference on Engineering Design, ICED 97, pp. 2/297-2/300, WDK, Heurista.

[31] Pavkovic N., Marjanovic D., "Structuring a Designers Workbench with Object-Oriented Design Plans", Proceedings of ICED 99, Vol. 3, pp. 1401-1406, WDK, 1999.

[32] Pavkovic N., "Object-oriented approach to design process modeling", Ph.D. thesis, University of Zagreb, Faculty of mech. engineering & naval arch., 2000. (written in Croatian language)

[33] Pavkovic N., Marjanovic D., "Considering an object-oriented approach to the design process planning", International Journal of Technology Management, Vol. 21, No. 3/4, 2001.

[34] POET 5.0 Programmer's Guide, San Mateo, POET Software Corporation, 1997.

[35] Pollack, M.E, "The uses of plans", Artificial Intelligence, Vol 57, No1, 1992, pp. 43-68.

[36] Pulm U., Lindemann U., Enhanced Systematics for Functional Product Structuring, Proceedings of ICED 01, Vol. "Design research – theories, methodologies and product modeling", pp. 477-484, WDK, 2001.

[37] Schlenoff, C., Gruninger M., Tissot, F., Valois, J., Lubell, J., Lee, J., The Process Specification Language (PSL): Overview and Version 1.0 Specification , NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD, 2000.

[38] Schregenberger J. W., Attribution of Models, Proceedings of the 12th International Conference on Engineering Design ICED 99, Vol. 2, pp. 1191-1194, WDK, 1999

[39] Smith R. P., Morrow J. A., Product development process modeling, Design Studies, Vol. 20, No. 3, pp. 237-261, 1999.

[40] Stachowiak H., Allgemeine Modeltheorie, Wien, Springer, 1973.

[41] Steward D. V., The Design Structure System: A Method for Managing the Design of Complex Systems, IEEE Tansactions on Engineering Management, Vol. EM-28, No. 3, pp. 71-74, 1981.

[42] Suh N. P, Principles of Design, UP, Oxford, 1990

[43] Szykman, S., Racz, J. W., Sriram, R. D., The representation of function in computer-based design, Proceedings of the 1999 ASME Design Engineering Technical Conferences, Las Vegas, ASME 1999, CD-ROM

[44] Tomiyama T., Yoshikawa H., Extended General Design Theory, Design Theory for CAD, ed. Yoshikawa, Warman, Procc. Of IFIP TC5/WG5.2, Tokyo, North Holland, 1987.

[45] Ullman D. G., The Mechanical Design Process, McGraw - Hill, 1992.

[46] Vajna S., Wegner B., Optimization of the Product Development Process with Evolution Algorithms and Simultaneous Engineering, Proceedings of International Conference on Engineering Design ICED 97, Vol. 3, pp. 3/67-3/70, WDK, Heurista, 1997.

[47] Wallace D. R., Abrahamson S. M., Borland N. P., Design Process Elicitation Through the Evaluation of Integrated Model Structures, Proceedings of ASME Design Engineering Technical Conference, DETC 99, Las Vegas, 1999.

[48] Van der Aalst W.M.P., Kumar A., A reference model for team-enabled workflow management systems, Data & Knowledge Engineering, Vol. 38, Issue 3, pp. 335-363, 2001.

[49] Watton J. D., Rinderle J. R., Improving Mechanical Design Decisions with Alternative Formulations of Constraints, Journal of Engineering Design, Vol. 2, No. 1, pp. 55-68, 1991.

[50] Werner H., Muth M., Weber C., Functional Modeling Using an Object-Oriented Design System, Proceedings of International Conference on Engineering Design ICED 97, Vol. 3, pp. 3/234-3/238, WDK, Heurista, 1997.