5

KNOWLEDGE BASED DESIGN OF COMPLEX PRODUCTS BY THE CONCEPT OF DESIGN WORKING SPACES

H. Grabowski, R.-S. Lossack
Institute of Computer Applications in Design and Planning (RPK)
University of Karlsruhe
Kaiserstrasse 12
D-76128 Karlsruhe
email: {gr,lossack}@rpk.mach.uni-karlsruhe.de

Abstract

In this article an approach is shown to support knowledge intensive engineering along the product-life-cycle. The approach supports the design of complex products by the concept of design working spaces. A design working space is an eucledian space and builds the framework for a methodological decomposition of complex design problems into sub-problems. The sub-problems are evaluated by a knowledge based design system which is flexible and evaluates problems along the product-life-cycle phases. With an example we describe the knowledge based design system for knowledge intensive design activities.

Keywords

Knowledge based design, knowledge intensive engineering, physically based modeling for design, preliminary design, product life cycle modeling, evaluation, complex design

1 INTRODUCTION

Product development as a whole becomes an important factor for the industrialized nations. It becomes more and more difficult to meet the customers' demand and to compete on the international markets with high quality and good value products, which have to be produced faster and faster to cut down the time to market. A product passes several product-life-cycle stages during its life time, beginning at the Product Planning phase and ending up when the product will be recycled (Figure 1).

All stages have a strong interrelationship and therefore it becomes more and more important to cover all these in-

M. Mäntylä et al. (eds.), *Knowledge Intensive CAD*

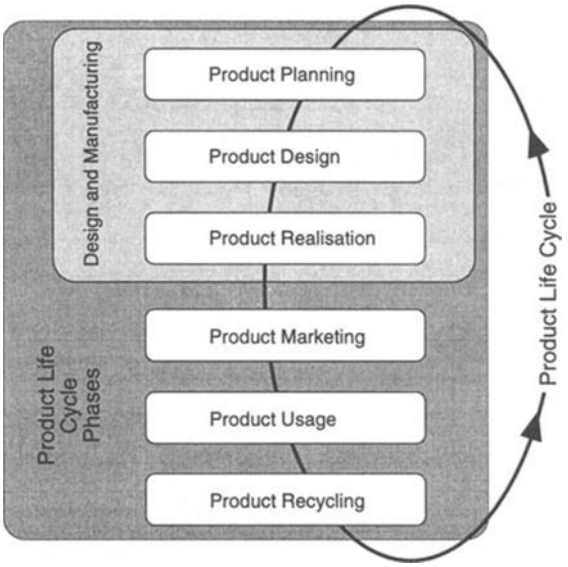

Figure 1 The Product Life Cycle

terrelationships in an integrated product model. In addition to this an integration along the product-life-cycle stages requires both a *logically integrated product* model as well as *modeling methods* to work with an integrated product model. Products become more and more complex and therefore modeling methods have to handle the product complexity as an important requirement they have to meet. An important factor in the product-life-cycle is the product design because here all characteristics of the future product is determined. First we give a short overview of the design process (Section 2) and discuss the informations that are necessary in a logically integrated product model for designing technical artifacts (Section 3). In Section 4 we introduce the concept of design working spaces as a basis for the decomposition of complex design problems to prepare the decomposed and structured problems for solving it by an integrated knowledge based system (Section 5). At the end we describe the architecture (Section 6) with which we implemented our ideas and verify the concept with an example (Section 7).

Before stepping into more details we direct some interest on a design serving as an example to explain our ideas. In Figure 2 there is shown a robot gripper designed for handling small parts, for durability and for low maintenance costs. A standard connection to the robot arm as well as the space in which the gripper has to fit is given by the product requirements. The working method of the gripper is as follows: The force with which the handled part is gripped is generated by an pneumatic energy source. The resulting force is then transmitted through a piston rod to a wedge splitting the force into two resulting forces which are applied to the grippers jaws. The applied pressure causes a movement of the piston rod towards the jaws and therefore the wedge causes a turning motion of the jaws

which results in the gripping force of the robot gripper. The detachment of the handled product is realized by a spring (not depicted) and by reducing the pressure applied on the piston. So the spring pushes the piston back to it's original position.

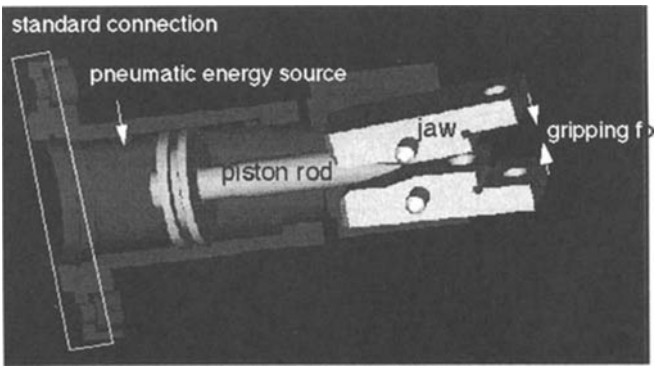

Figure 2 Product Example

2 OVERVIEW OF THE DESIGN PROCESS

The design process is considered in an idealized manner as a *successive concretion* of the description of the to-be characteristics of a technical object [2],[1], [3] a. o. This concretion process takes place on the product modeling level. Koller, Pahl, Beitz, Roth and other design methodologists define this process to lead from the *incomplete to the complete*, the *abstract to the concrete*, the *rough to the precise*, *the provisional to the definitive* and from possible *alternatives to the optimal solution*.

Typical for the design process is the successive growth of the set of design characteristics with respect to the current state of the design. Here design characteristics are defined as the instantiated solution properties of a product to be developed. The sum of all these solution properties determine the overall behavior of the product in real life. The description of the solution properties is the result of the design phases in the design process that consists of defining the *requirements* of a product, the definition of the *functional structure* and the *function flow* within a product, the description of the *physical effects* which can be assigned to the respective functions in correspondence with the modeling of the product's *effective geometry* and the *embodiment* (Figure 3).

In each phase a *general problem-solving cycle* has to be performed (Figure 3) which can be derived from the psychology of problem solving [4]. In this problem-solving cycle the designer is first *confronted* with the problem. Afterwards the *definition* of the essential problem is performed by fixing the objectives, main constraints and the environment for the intended solution. The next step is *finding* and *representing* the solution for the defined problem. After that the solution has to be *evaluated* followed by making a *decision*. Finally as the following step the problem-solving cycle is *reiterated*. Each established solution serves as a supposition for the next problem

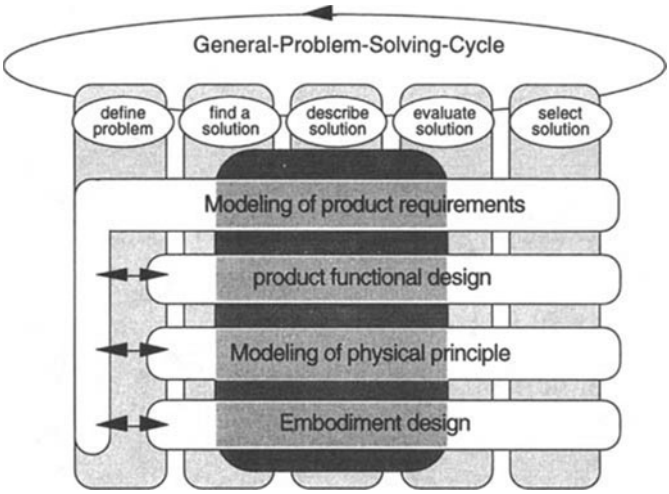

Figure 3 Modeling Layers in the Design Process for Technical Products

respectively the next design stage. In this way, the designer proceeds from the *qualitative* to the *quantitative*, from the *abstraction* to the *concretion*, from the *incomplete* to the *complete* etc. The general problem solving cycle is together with the design stages of the design process, the basis for the informations needed to solve design problems.

3 THE MODELING LAYERS IN THE DESIGN PROCESS

The result of the design process is a model of an artifact. With the three fundamental magnitudes of design, *matter*, *energy* and *information* every artifact technical system can be described on an abstract physical level [3], [1]. A technical artifacts is embedded in the environment by means of *input* and *output* and can be treated like a system. A *system* can be divided into *sub-systems*. What belongs to a particular sub-system is determined by the *system boundary*. With this approach it is possible to describe every technical system at every stage of *abstraction*. Describing a proposed technical artifact by means of a system consisting of elements, which are grouped by the system boundary related with each other by input and output, we use the term *function* or *product function*. If the input and output of the product function is described on the basis of matter, energy and information then we use the term *general function*. If the inputs and outputs represent *physical magnitudes* like force $\vec{F}$ or torque $\vec{D}$ and the relationship between the in- and output is described by a limited set of *function verbs*, then we use the term *special function*. The function verbs describe the proposed transformation between the input and output on an conceptual

level. With reference to Roth (1994) we use the set of function verbs *Change*, *Connect*, *Channel* and *Store*. In the case of relating the in- and output by a *physical law* that is described by a mathematical equation then we talk about an established *physical principle*. All technical artifacts are complex constructions, so every artifact is described by a *general function structure, a special function structure and a physical principle structure*. In accordance to the phases of the design process of Figure 3 the logical modeling layers can be been defined as follows:

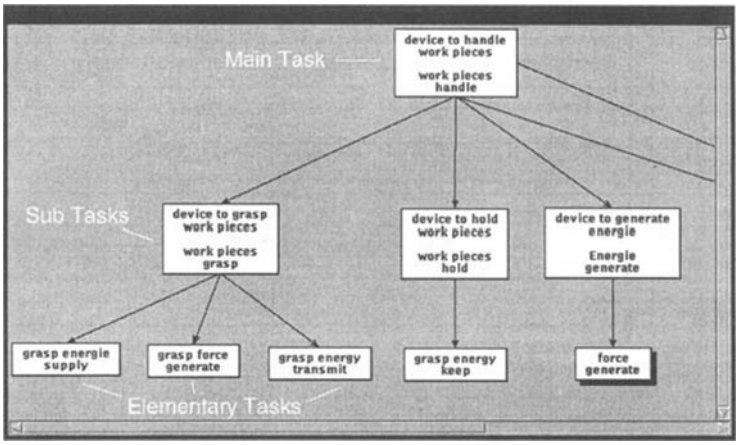

Figure 4 The Task Structure of the Robot gripper

REQUIREMENTS MODELING LAYER

The requirements modeling layer serves for the computational projection of the results won by the clarification of the design task. This modeling layer contains the *preconditions* of the design, the *to-be properties* of the future product and the description of the product's immanent *task structure*. The Task Structure is defined by a *noun* and a limited *set of task verbs*.

Figure 4 shows an example of the product task structure* of the robot gripper from Figure 2. First the designer has to establish the *main task* the product has to fulfill. In the example the designer has to develop a *device* that is able *to handle work pieces*. The designer gets the information to form the "main task sentence" from the customer or the product planning stage. In the following the main task has to be broken down into *sub-tasks*. This can be done manually or automatically. An automatic derivation has to be done by so called *process pattern* [5]. This modeling layer is finished when the designer has modeled *elementary sub-tasks*, so that a transition to the functional modeling layer.

FUNCTIONAL LAYER

Functional modeling serves for finding and describing the functions of a design solution as well as the functional

*The English sentences in the example are badly formed because the design system DIICAD Entwurf is developed for a German syntax structure

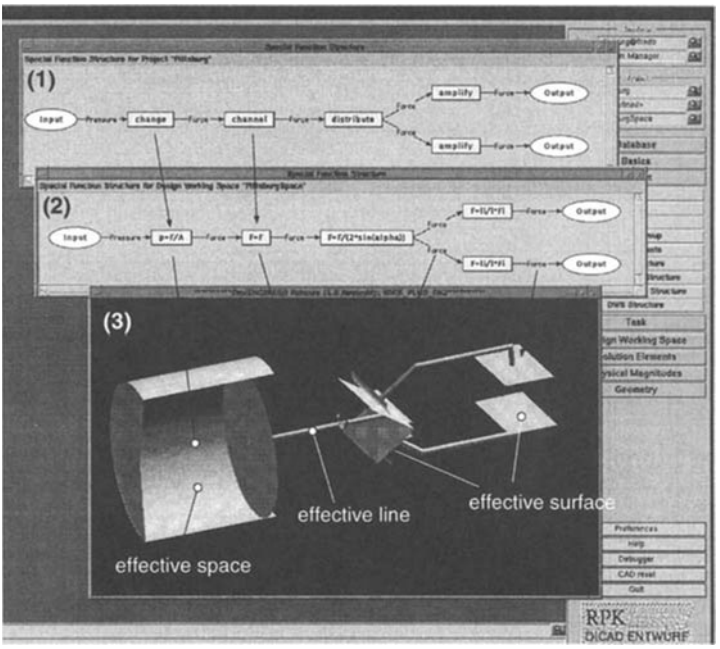

Figure 5 Modeling Structures of the Robot gripper

interrelationships within the future product. Functional modeling allows the definition and manipulation of functions on different levels of abstraction. The logical transition and by that the concretion of the functional model to the physical principle model is supported by the specification of the vectorial functional structure. Figure 5 (1) shows the established special function structure of the robot gripper (Figure 2). On this level there is seen that the energy type *pressure* (input) is *changed* in the energy type *force*. After that the force is *channeled*, *distributed* and *amplified* (output).

PHYSICAL PRINCIPLE MODELING LAYER

The physical principle design serves for the description of the solution principle of a special function. It covers all informations of the product's *physical* solution. These informations contain the *physical effect* that is described by a *mathematical equation* and geometrical informations as *effective lines*, *effective surfaces* and *effective spaces*. In Figure 5 (2) there is depicted the established physical principle structure and the derived effective geometry (Figure 5)(3) of the robot gripper.

SHAPE MODELING LAYER

The shape modeling is the most concrete one of the product modeling layers. In this layer the design process is completed by the geometrical definition of all design features, parts and assemblies.

To model complex products there is still a lack of modeling methods which helps to structure the product informations in order to reduce the complexity of the subtasks. With respect to this problem and focusing on the evaluation problem we introduce the concept of "design working spaces" which is described in the next section.

4 DESIGN-WORKING-SPACE

A *design-working-space* is an *Euclidean space* available for the designer *to solve* his *design task*. The design-working-space is defined by an envelope (geometric system boundary) and its constraints (in-/outputs) (see definition of the dws). The fundamental idea comes from system theory and is therefore not limited to the geometric scope. The *main purpose* of *design-working-spaces* in this context is to *integrate the design stages* and to *decompose* a given design problem into solvable subproblems.

In Figure 6 there are three design-working-spaces which have to fulfill a special product function, like "change force into torque" (Figure 6) (2) or "amplify force by force" (Figure 6) (3). The system boundary of a design-working-space is clearly defined by its maximum envelope, effective geometry and by the physical in- and outputs like the physical magnitudes force $\vec{F_1}$ and torque $\vec{D_1}$; the envelope and the effective geometry is represented by free form surfaces. The envelope describes the maximum space inside which a special problem has to be solved. The effective geometry is described by effective spaces and effective surfaces which transmit for example forces. The relationship between the design-working-spaces is established exemplary by the general magnitude energy E and the special magnitudes force $\vec{F_2}$ and torque $\vec{D}$.

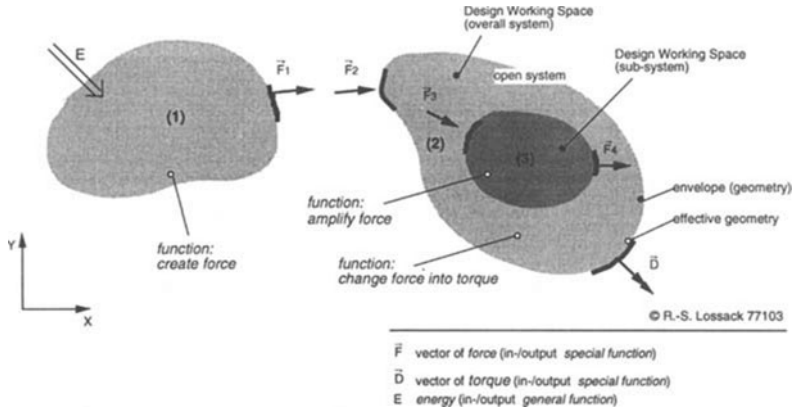

Figure 6 Concept of the Design Working Space

Design-working-spaces will be builded up by the following rules:

1. A design-working-space consists of a *set of elements* and of a *set of relationships* between the elements.
2. *Elements* of a design-working-space are *informations of the design stages*, like product requirements, functions or physical principles. *Relationships* between the elements are general or special magnitudes like energy, information, matter or force, torque etc.
3. Every design-working-space can be subdivided into *independent sub-spaces*. If elements of different sub-spaces will be grouped together then this sub-spaces are called *overlapping design-working-spaces*.
4. Every design-working-space and every sub-space is defined by a *system boundary*. The system boundary is specified by its envelope, making available the maximum of geometric space for the design and its effective geometry. The effective geometry defines the point at which the physical phenomena takes place.
5. A *system boundary* of a design-working-space has *one or more in-/outputs*. If a design-working-space has *no* in-/outputs then we talk about a *closed design-working-space*, on the other hand about an *open design-working-space*.

5 DECOMPOSING AND EVALUATING DESIGN PROBLEMS BY DESIGN WORKING SPACES

During the design process it is typically that the designer is permanently changing the scope of design to evaluate the design which depends on a certain context [4],[3]. An example of a gear box design is shown in Figure 7. During the design process the designer changes the view on the design by clipping and zooming the drawing area. The necessity is obvious because on one hand the overall gearing box on the other hand the detail of the bearing is important but both have to match the requirements regarding size, function, general arrangement, spatial compatibility etc. In all this, technological, economical and other considerations are of paramount importance, so that the designer needs to have modeling methods to structure the scope of design to evaluate the design according to different criteria. The scope of design covers all aspects including the product task, the product function, principle, effective geometry and must not limited to the embodiment stage as shown in Figure 7.

Modeling methods which meet the required functionality discussed above involve a large number of corrective steps in which analysis and synthesis constantly alternate and complement each other. This explains why the familiar methods underlying the *search for solutions* and *evaluation* must be complemented with methods facilitating the *identification of errors* (design faults) and optimization. The *collection of information* on materials, manufacturing processes, repeat parts and so on is complex and requires a considerable effort: many actions have to be performed simultaneously, some steps have to be repeated at a higher level of information. Without having structuring methods it is not possible to evaluate complex products and it is impossible to evaluate complex products with respect to different criteria.

For that, a complex problem has to be defined, broken down into sub-problems until these sub-problems can be evaluated. Figure 8 shows the idea. There is given an overall problem which cannot be evaluated because it is too complex. The idea is to *identify* and to *mark off* the *critical sub-spaces*. A sub-space is a critical sub-space if the complexity of an overall problem can be reduced in a large amount. After having solved all critical sub-spaces the designer can go back to the overall problem to solve it again.

With the help of Figure 8 we summarize the elementary steps that have to be performed in this process. First there is given the *overall problem* that has to be broken down into *sub-problems*. This step has to be repeated until an *elementary problem* has reached. This elementary problem is evaluated and solved. After that the sub-problem should be evaluated on a higher level by evaluating the relations (constraints) between the solved elementary problems while regarding the elementary problems as a whole. These steps have to be repeated until the overall solution is evaluated.

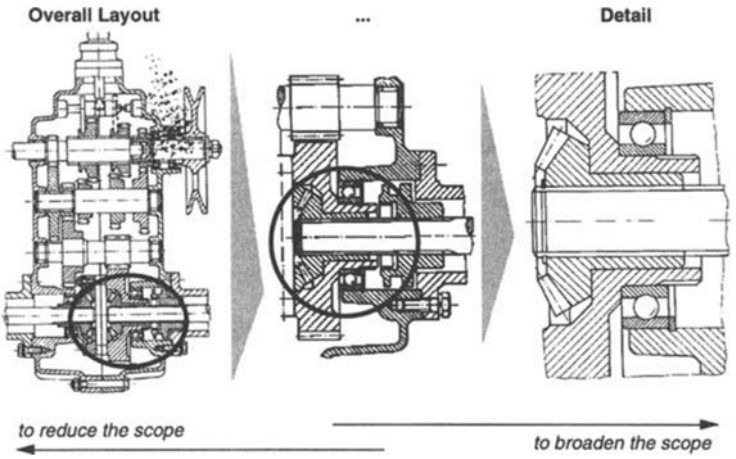

Figure 7 Broadening and Reducing the Scope of a Gear Design with the Example of a Technical Drawing

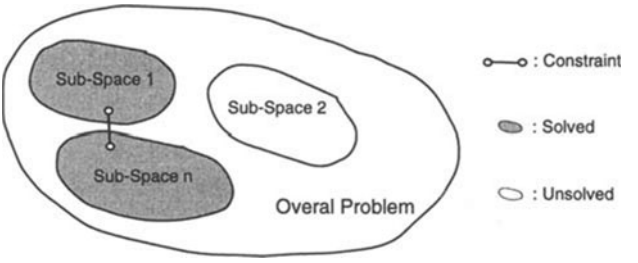

Figure 8 Decomposition an overall problem space into sub-problem-spaces

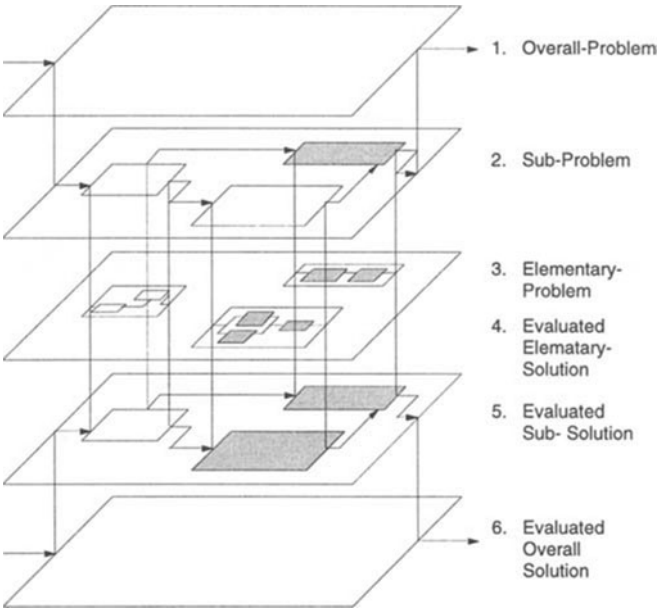

Figure 9 General Evaluation Process for Evaluating Designs

6 THE ARCHITECTURE OF THE DESIGN SYSTEM DIICAD ENTWURF

The following section describes a part of the integrated product model used in "DIICAD Entwurf"† for the evaluation of design principles‡ with design working spaces. The section is divided in two subsections. Subsection 6.1

†*DI*alog oriented *I*ntelligent *CAD* System, *Entwurf* is a subsystem for Design Space Modeling

‡Design principles have been discussed at some lenth in literature. We refer to Pahl and Beitz [1], who distinguish among other things Principles of force transmission like "Flowlines of force and principle of uniform length", "Principle of direct and short force transmission path", "Principle of matched deformations" and "Principle of balanced forces"

describes the object oriented product model serving as the knowledge base. Subsection 6.2 describes the implemented rule base for the design principle "Force transmission"[§] which is used in the the example.

Figure 10 shows the Architecture of the design system. The System is devided in three parts: the *Graphical User Interface*, the *Design Modeler* and the *Object Oriented Database*. The Graphical User Interface, the Design Modeler and the Application Programming Interface (API) to the Database and the Geometric Modeler is written in itcl[¶], except from the knowledge base that is written in CLIPS[‖]. As geometric modeler we used Pro/ENGINEER that is controlled by the Design Modeler with ProDEVELOP**.

The Design Modeler consists of two parts (Figure 10). One part is the *Design Working Space Modeler* and the other one the *Knowledge Based System*. The Design Working Space Modeler consists of 5 Modules to model the informations of the modeling layers discussed in section3. The Knowledge Base is logically divided in a *Knowledge Acquisition* and an *Interpretation* Module. Logically means that these two "modules" can be distinguished but they are incorporated in one rule clause (see Section 6.2).

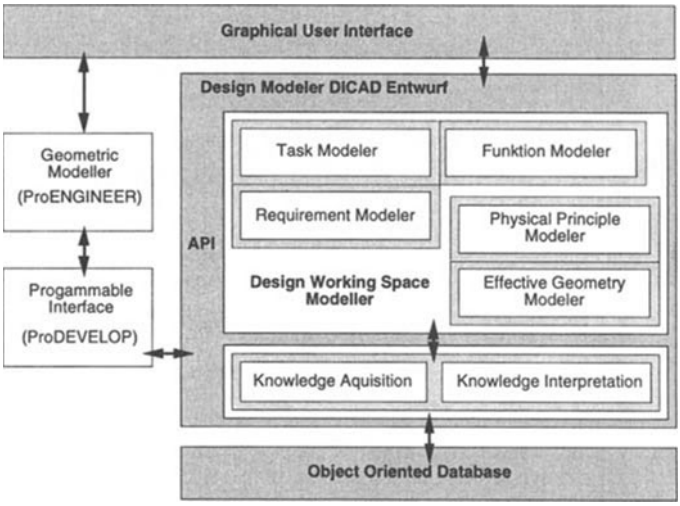

Figure 10 Architecture of the Design System DIICAD Entwurf

[§]According to Pahl and Beitz [1] Force transmission means "Principle of direct and short force transmission path"

[¶]itcl provides object-oriented extensions to Tcl, much as C++ provides object-oriented extensions to C

[‖]CLIPS is an expert system tool developed by the Software Technology Branch (STB), NASA/Lyndon B. Johnson Space Center. CLIPS is designed for writing expert systems.

**ProDEVELOP is the programmable interface to Pro/ENGINEER

6.1 The Information Model

The following section describes the information model implemented in DIICAD Entwurf. The classes consist of object attributes, object relations, object methods and class functions. Only the important object methods and class functions will be described.

An object relation is always bidirectional, e.g. if a relation named *requirement* was defined from a *DesignContext* object to a *Requirement* object there will be an implicit relation named *requirement* from the *Requirement* object to the *DesignContext* object. Such a relation can be read as "a *DesignContext* object *has* a requirement (which is a *Requirement* object) and a *Requirement* object *is* a requirement of a *DesignContext* object".

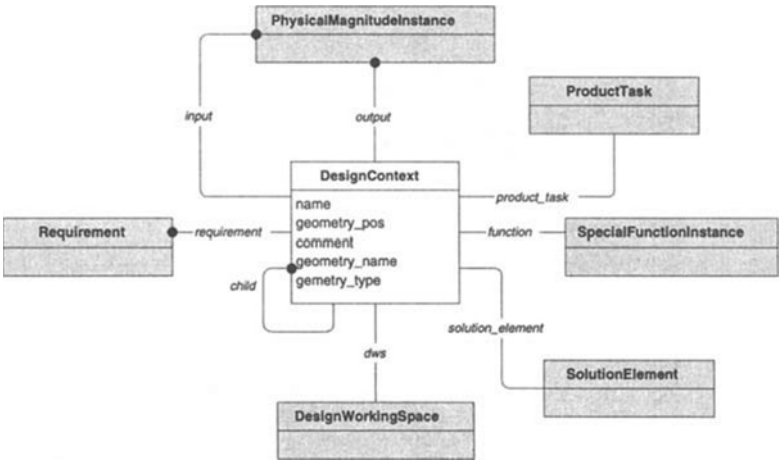

Figure 11 Information Model of the Design Context

The *DesignContext* is the central class of the model (Figure 11). It is mainly used to group instances of other classes to form the context of a design problem. *DesignContexts* are able to group themselves to split the problem into subproblems that can be solved independently. A found solution can be joined together to form the solution of the higher context. A *DesignWorkingSpace* object is always related to a *DesignContext* and defines the geometric boundary of a design problem.

A *ProductTask* object defines the task the product has to fulfill. A *ProductTask* is defined by a *noun* and a *verb* (Figure 12). The noun can be chosen freely, the verb has to be one of a list of predefined *ProductTaskVerb* objects. The context of the design problem is defined by the related *DesignContext* object.

A *Requirement* object is always related to a *DesignContext* (Figure 11). It defines the product requirements that have to be fulfilled by the solution of a design problem (Figure 12). The related *Person* object specifies the claimant of the requirement. If a physical magnitude is specified as a quantitative requirement, a *Units* object is related to

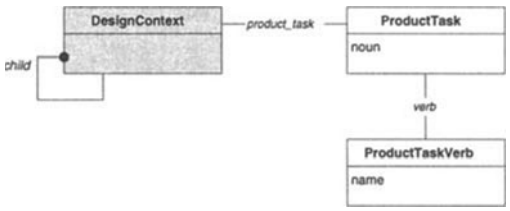

Figure 12 Information Model of the Product Task

define the minimum and maximum values. The requirements for the solution of a design problem can be classed by their scope. The *RequirementScope* defines the accepted requirement scopes, for example costs.

The *PhysicalMagnitude* objects define the physical magnitudes by the attributes of name, units, symbol and type.

The *PhysicalMagnitudeInstance* (Figure 14) defines the interface of a *DesignContext*. A PhysicalMagnitudeInstance can be related to a *DesignContext* as its *input* or *output*. This relation defines the "direction" of the "function flow". The *PhysicalMagnitudeInstance* has a relation to a *PhysicalMagnitude*. The *PhysicalMagnitude* describes the physical magnitude itself whereas the *PhysicalMagnitudeInstance* object describes the interface to a *DesignContext* with its relation and attributes.

A *Unit* object defines a unit by its name and the mathematical combination of base units. Only base units are known to the system but if a *PhysicalMagnitude* object has to be defined which has no base unit, it is possible to define a new unit by the combination of base units with the *Unit* object.

A *SpecialFunction* defines a special function consisting of an input, verb and an output. The input and output are relations to *PhysicalMagnitudes*, the verb is a relation to a *SpecialFunctionVerb* that is a list of accepted verbs for a special function and describes the relation between *PhysicalMagnitude* input with the result of its *PhysicalMagnitude* output.

A *SpecialFunctionInstance* is part of a design solution for a problem defined by a *DesignContext*. Its verb and type of the inputs and outputs are defined by its related *SpecialFunction*. The type of a function distinguish between a main and an auxiliary function.

A *PhysicalPrinciple* object defines a physical principle by its name and physical law, which is specified by a mathematical equation. The related *SpecialFunction* objects define the supposition for the use of the physical principle.

An *EffectiveGeometry* defines the geometry where the physical phenomena, described by the *PhysicalPrinciple*, takes place. The designer may modify the parameters of the effective geometry within its constraints and use methods to derive the geometry of the *SolutionElement*.

The *SolutionElement* object defines the geometry of a solution for a design problem (Figure 15).

A *Task* object defines a task for a designer and grants explicit permission to view and modify those parts of the design problem that it refers. A Task can suppose other tasks to be completed first, i.e. *Task* objects perform sequential control upon the design process.

All the people involved in the design process are known to the system as *Person* objects. A person is defined by its name, login, password and authorization.

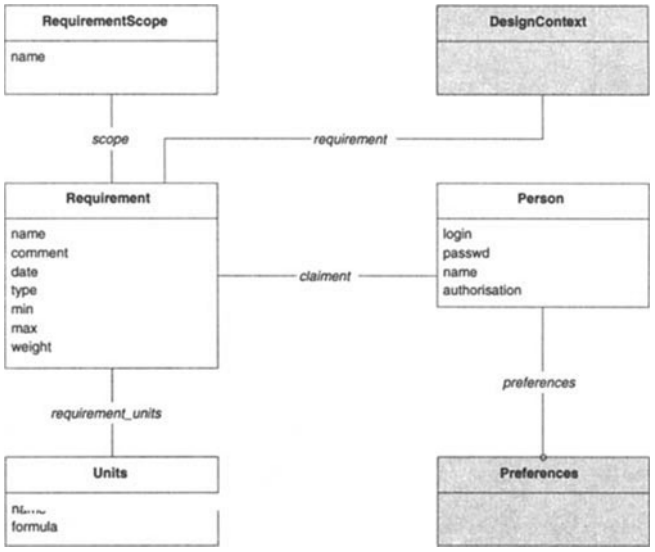

Figure 13 Information Model of the Requirement class and its related classes

Each *Person* object (general databases class) has a related *Preference* object that holds user specific preferences, e.g. the path to the working directory or the desired view on the geometry.

Like the *Preference* object a *Coordinator* object (general databases class) is related to every *Person* object. The *Coordinator* object coordinates the designers within one project. It has relations to the current *Project* object, *Task* object and *DesignWorkingSpace* object of the related *Person* object.

A *MaterialType* object defines a material type. It is used to class *Material* objects.

A *Material* object (general databases class) defines the Material type and all of its properties.

6.2 The Knowledge Base

This section describes the complete rule base for the design principle "FORCE-TRANSMISSION". The rule base contains more design principles than only "FORCE-TRANSMISSION" but this design principle is the simplest one, can described completely and shows the idea of the concept.

The module FORCE-TRANSMISSION describes all rules that are able to detect whether the design principle "FORCE-TRANSMISSION" can be fulfilled or not. The rule base extracts the the facts from the product model by the API over a piping mechanism. If it is not possible to derive or to extract all facts from the product model

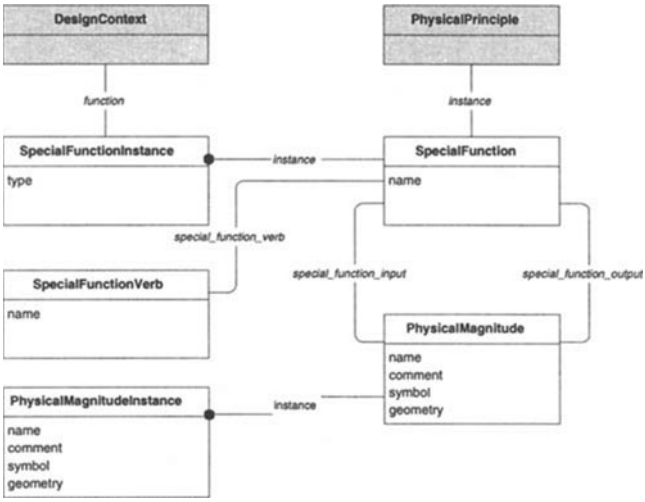

Figure 14 Information Model of the Product Function and their Structure

an user interaction takes place. In this case the user is asked to enter the missing facts which are instantiated in the product model (knowledge acquisition). According to a certain design principle an incomplete design is becoming complete.

```
(defmodule FORCE-TRANSMISSION
   (export ?ALL)
   (import MAIN ?ALL))
```

Module for the Evaluation of FORCE-TRANSMISSION

```
(defrule FORCE-TRANSMISSION::ask-for-physical-principle-effectgeometry ""
  (declare (auto-focus TRUE))
  (special-function (input force)(function-verb channel)(output force)
  (attribute  ATTRIBUTE))
  (solution-element (type ?)(exist FALSE))
  ?f <- (physical-principle (effect EFFECT)(effectgeometry NAME BOOL))
  =>
  (bind ?bool (ask-command "is_effective_geometry_a_straight_shaft"))
```

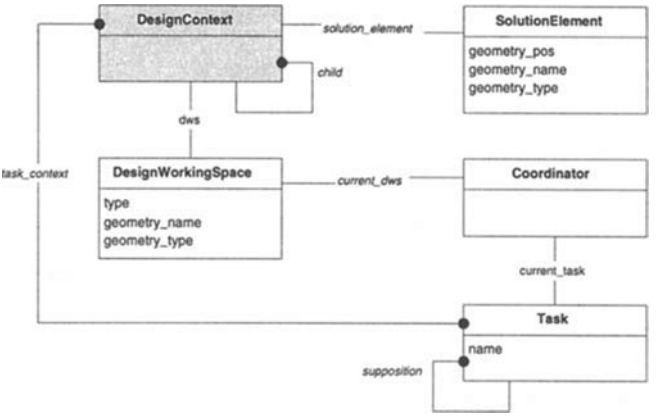

Figure 15 Information Model for the Design Working Space and the Solution Elements

```
(if  (neq ?bool {}) then
  (if ?bool then
    (modify ?f (effect actio-reactio)(effectgeometry shaft TRUE))
    else
    (modify ?f (effectgeometry unknown FALSE)))
  else
  (bind ?answer (yes-no-question "Is there effective geometry and is\
   it a straight shaft?"))
  (switch ?answer
          (case TRUE then
                (modify ?f (effect actio-reactio)\
                (effectgeometry shaft TRUE)))
          (case FALSE then
                (modify ?f (effectgeometry unknown FALSE)))
          (case QUIT then
                (assert (clear all))))))
```

Rule for the Evaluation of the Effective Geometry of a Physical Principle

```
(defrule FORCE-TRANSMISSION::ask-for-straight-shaft ""
  (declare (auto-focus TRUE)(salience 10))
  (special-function (input force)(function-verb channel)(output force)
```

```
  (attribute  ATTRIBUTE))
  ?f <- (solution-element (type TYPE)(exist TRUE))
  =>
  (bind ?bool (ask-command "is_solution_element_a_straight_shaft"))
  (if  (neq ?bool {}) then
    (if ?bool then
      (modify ?f (type straight-shaft))
     else
      (modify ?f (type not-straight-shaft)))
   else
    (switch (yes-no-question "Is the geometry a straight shaft?")
            (case TRUE then
                  (modify ?f (type straight-shaft)))
            (case FALSE then
                  (modify ?f (type not-straight-shaft)))
            (case QUIT then
                  (assert (clear all))))))
```

Rule to Evaluate a straight shaft

```
(defrule FORCE-TRANSMISSION::ask-for-physical-principle-effect ""
  (declare (auto-focus TRUE))
  (special-function (input force)(function-verb channel)(output force)
  (attribute  ATTRIBUTE))
  ?f <- (physical-principle (effect EFFECT)(effectgeometry unknown FALSE))
  =>
  (bind ?answer (yes-no-question "Is the effect actio-reactio?"))
  (switch ?answer
          (case TRUE then
                (modify ?f (effect actio-reactio)))
          (case FALSE then
                (modify ?f (effect unknown)))
          (case QUIT then
                (assert (clear all)))))
```

Rule for the Evaluation of a Physical Principle

```
(defrule FORCE-TRANSMISSION::force-transmission-best-fulfilled ""
   (physical-magnitude (exist ?bool1)(against ?bool2)\
   (short-distance ?bool3))
   (or (physical-principle (effect actio-reactio)\
   (effectgeometry shaft TRUE))
       (physical-principle (effect ?)(effectgeometry shaft TRUE))
       (physical-principle (effect actio-reactio)\
       (effectgeometry unknown FALSE))
       (physical-magnitude (exist TRUE)(against TRUE)\
```

```
        (short-distance TRUE))
        (solution-element (type straight-shaft)(exist TRUE)))
    (not (designing-principle (type force-transmission)(fulfilled TRUE)))
    =>
    (if (eq ?bool1 ?bool2 ?bool3 TRUE) then
        (printing-out \
        "Direct Force Transmission : Best Fulfilled With a Straight Shaft")
    else
        (printing-out "Direct Force Transmission : Fulfilled"))
    (assert (designing-principle (type force-transmission)(fulfilled TRUE))))

(defrule FORCE-TRANSMISSION::force-transmission-not-fulfilled ""
  (special-function (input force)\
  (function-verb channel)(output force)(attribute ?))
  (or (solution-element (type not-straight-shaft)(exist TRUE))
      (physical-magnitude (exist TRUE)\
      (against FALSE)(short-distance FALSE)))
  =>
  (printing-out "Direct Force Transmission: Not Fulfilled")
  (assert (clear all)))
```

Rule for the Evaluation of the Quality of the Fulfillment of a force transmission

```
(defrule FORCE-TRANSMISSION::ask-for-traction-or-pressure ""
  (designing-principle (type force-transmission)(fulfilled TRUE))
  =>
  (bind ?var (ask-command "is_shaft_traction_or_pressure"))
  (if  (neq ?var {}) then
    (assert (shaft-is-under ?var))
    (focus PRESSURE-AND-TRACTION-TRANSMISSION)
    else
    (bind ?answer (ask-a-question "Is the shaft under traction or pressure?"\
     traction pressure quit))
    (if (neq ?answer quit) then
      (assert (shaft-is-under ?answer))
      else
      (assert (clear all)))))
```

Rule for the Evaluation of technical traction or pressure

7 EXAMPLE

Given is an example depicted in Figure 16. The effective geometry of the robot gripper is infolded by three design working spaces which have been placed user interactively. The designer can select one of them and specify what design principle he wants to have evaluated.

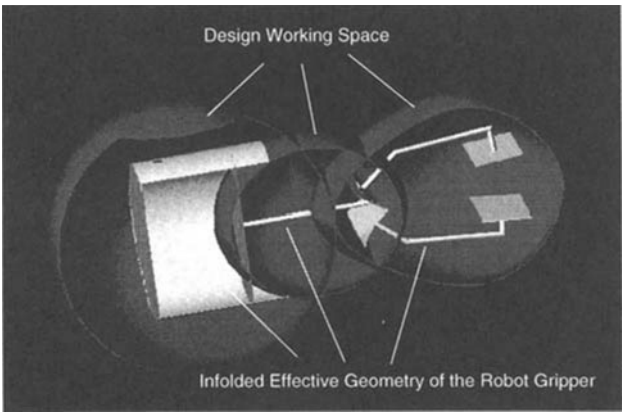

Figure 16 Result of Infolding the Effective Geometry in Design Working Spaces

Figure Figure 17 shows the user interaction with the system. Because of not being able to derive the direction of the physical magnitude he is asked to enter their relative direction which is then instantiated into the product model.

8 CONCLUSIONS AND FUTURE WORK

We have modeled and verified the concept of decomposing an overall problem by design working spaces and evaluated the decomposed design by a knowledge based system in a small application on the DIICAD product model. Modeling design solutions in design working spaces and evaluating the decomposed solutions in a "scalable way" is possible. This is realized by using a knowledge based system which is integrated in the product model. The concept has been implemented on CLIPS, Pro/ENGINEER and itcl.

Evaluating a design in an integrated product model in the described way is a promising approach. We consider it an important point that in the future basic research has to be done in developing *methods for decomposition of design working* spaces. So, our next steps will be to develop those methods to try to automate the decomposition process.

9 ACKNOWLEDGEMENTS

This work is supported by the *Deutsche Forschungsgemeinschaft (DFG)* in the project *Modeling of Design Working Spaces* of the Sonderforschungsbereich 346 *Computer Integrated Design and Manufacturing of Single Parts*. In

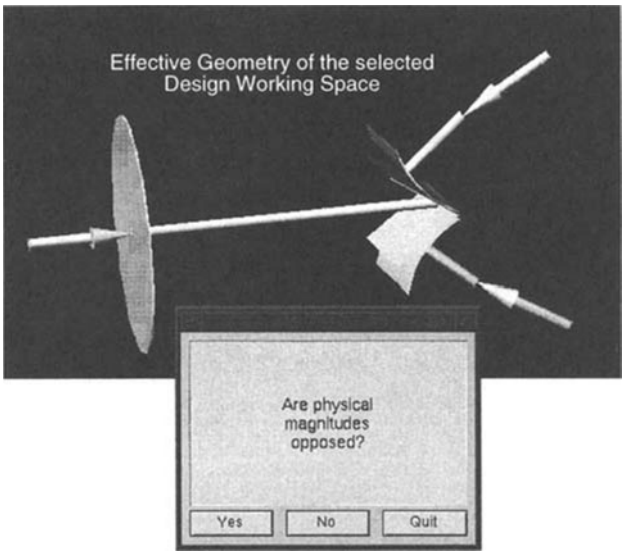

Figure 17 Example for the User Interaction with the Knowledge Based Sytem

addition to this I will thank my students cand. inf. Harald Kunze and cand. mach. Arno Michelis for discussing and implementing the concepts.

REFERENCES

Pahl, G. Beitz, W. *Engineering Design.* Springer, 1994.

R. Koller. *Konstruktionslehre fuerr den Maschinenbau (Design Theory for the Mechanical Engineering).* Springer, 1985.

K. Roth. *Konstruieren mit Konstruktionskatalogen.* Springer, 1994.

A. Rutz. *Konstruieren als gedanklicher Prozess (Design as Intellectual Process).* PhD thesis, KM, Lehrstuhl fuer Konstruktion im Maschinenbau, TU Muenchen, 1985.

Grabowski, H. Lossack, R.-S. Weis, C. A design process model based on design working spaces. In *Knowledge Intensive CAD-1.*