in: International Journal of Systems Automation: Research and Applications (SARA), 1, 1991, pp. 347-367

Handling Functional Constraints of Technical Modeling Systems in a KBMS Environment

S. Deßloch, C. Hübel, N.M. Mattos, B. Sutter

University of Kaiserslautern, Department of Computer Science

P.O. Box 3049, D-6750 Kaiserslautern, Germany

phone: (631) 205-3284

e-mail:dessloch@informatik.uni-kl.de

Abstract

In this paper, we introduce the technical modeling approach which attempts to incorporate more semantics of the design process into the internal object representation. The most important feature of technical modeling can be characterized by the term functional constraints. They restrict the design process in such a way that the design yields a consistent component part fulfilling the functional objectives.

Special emphasis will be given to the applicability of Knowledge Base Management Systems (KBMS) to support technical modeling. Therefore, we introduce a concrete KBMS and present the modeling of technical objects and operations by an example taken from the area of shaft design. In particular, we present our ideas for modeling, and organizing the functional constraints in the knowledge base.

1. Introduction

Computer support in product planning, design, and manufacturing has become of increasing importance over the last few years. For this reason, a lot of research activities focus on the development of **integrated engineering systems** aimed at a continual, efficient, and integrated support of the design process as well as at the organization of the operational process /GAPR88, SGL89/. The most important issue in supporting an integrated design process is the construction and management of an **integrated product model**, which comprises all relevant information from the phase of functional design up to the preparation of the manufacturing process. In order to get a more structured view of an overall product model, it is commonly separated into several partial models (functional model, technical (shape) model, geometrical model, technological model, and drafting model /GAPR88/).

Though the information about the design objects contained in each of these partial models is represented in a rather different way, there exist partial overlaps (seen from a more abstract point of view), and, additionally, various relations and complex dependencies among the partial models. Considering a given design object, the technical element 'drilling hole', for example, fulfills a particular aspect represented within the functional model. This particular drilling hole element is represented as such in the technical model and additionally influences the representation of (associated) elements within the other partial models. Thus, the technological model of the considered design object contains tolerance information as well as some manufacturing hints concerning the particular drilling hole element. Similarly, the actual geometry, i.e., the dimension of this drilling hole, is determined by the geometrical model. Besides the relations among the partial models, one can identify very important dependencies among the information covered in a single partial model. For example, the technical element 'drilling hole' strongly depends on the particular features of the technical element pin or screw that have to be placed in the drilling hole. These features, in turn, depend on the functional issue the pin or screw have to satisfy. All these dependencies, also denoted **technical constraints** /DB89, SG88, WH88/, have to be considered during the design process in order to guarantee the design of a consistent part fulfilling all functional specifications.

Nowadays, the designer mostly applies geometric modeling systems, i.e., he basically models only the geometrical view of the design object. Technical information which is inherent to the design process and well known to the designer may be just lost because it cannot be represented by such modeling systems /vR89/. Here, we suggest a technical modeling approach to introduce more semantics directly into the internal object representation. **Technical modeling** means designing a component part by means of technical objects (e.g., bearing, shaft, shoulder, pin, drilling hole) that are manipulated by object specific operations (like creating or placing) regarding the technical constraints relevant for the design process /PS90, Pa89, SR88, KVY89/. The power of the technical modeling approach is significantly determined by the capability of technical modeling systems to specify what a technical object is, how a technical operation works, and the most important issue, which technical constraints have to be considered.

From a system development point of view, the question is which basic data, information, or knowledge management component can be used for supporting the specification and the control of technical functional constraints. In our opinion, Knowledge Base Management Systems (KBMS /BM86/), a new generation of systems incorporating artificial intelligence (AI) techniques and database system (DBS) techniques to allow an effective and efficient management of large knowledge bases, are suitable candidates for this approach /DHMM89/.

The goal of our paper is to show the adequacy of KBMS techniques to support the technical modeling process, especially the administration and enforcement of the technical constraints. In the following chapter, we introduce technical modeling and the previously mentioned technical functional constraints in more detail. In chapter 3, we introduce a concrete KBMS, that was developed at our university and describe the modeling of technical objects using the knowledge model of our KBMS. In chapter 4, we take a closer look at dynamic issues of technical operations, while chapter 5 focuses on the organization of the corresponding functional constraints. Finally, we give a conclusion to the main ideas and provide an outlook to our further research efforts.

2. Short Overview of Technical Modeling and Technical Functional Constraints

A design process is generally divided into several design phases, e.g., definition of the functionality, specification of the physical principles, shape design, and detailing phase, during which the designer determines technical objects according to the function, they have to fulfill. The technical modeling approach presented here is well suited for the support of the shape design (assembly group design) and the detailing phase of the component part design.

The notion technical modeling implies the manipulation of objects in a technical object structure using application oriented operations and regarding the mentioned technical constraints. The technical objects relevant to the assembly group design are assembly groups and component parts. **Assembly groups** are parts of a machine which fulfill a partial function. An assembly group consists of other assembly groups or a number of component parts, which are not composed of other parts (i.e., are viewed as atomic units). Component parts are either chosen from premanufactured parts (standard parts or supply parts) or manufactured on demand to fulfill a technical function (drafted parts). Every drafted part is specified by a task description created during the assembly group design.

In general, a **component part** is completely specified by functional elements, primary elements, and secondary elements. This is especially valid for rotationally symmetric parts like shafts. A **primary element** denotes a rotationally symmetric or a prismatic form element (e.g., a shoulder) for describing a section of a component part, including the technical information (e.g., material, tolerances). A **functional element** represents an area of a component part realizing a subfunction (e.g., torque transmission by a feather key). **Secondary elements** are chosen from a number of variants and are associated with a primary element. Like primary elements, they possess their own information (e.g., size and surface of a chamfer), and moreover, they may be standardized. The entirety which is implied by the functionality of one primary element, up to one functional element and an arbitrary number of secondary elements is denoted **technical aggregate**. For example, the shaft in Fig. 1 consists of four technical aggregates where the second aggregate is composed of a shoulder (primary element), a feather key (functional element), and a chamfer (secondary element).

The overall design problem of the assembly group design is given by a functional objective and environmental restrictions /Gr88/. In the assembly group design, the task description of the overall design is divided into task descriptions (functional constraints) for the subassembly groups or the component parts. In the component part design, the function of the component part determines hierarchically the functional objective of the primary, secondary, and the functional elements in the same manner.

Therefore, technical modeling implies the consideration of the existing relationships among the technical objects, i.e., a formal description of the functional objective and the environmental restrictions. In our approach, the technical functional constraints describe these dependencies. Such dependencies cover universally valid constraints due to physical effects or to design logic and specific technical functional constraints of a component part or among the elements of an assembly group. In the top-down design process, the constraints of a superior assembly group are propagated to the subassembly groups. This is even valid for the component part design. However, we have to take into consideration a kind of back propagation. The full specification of one component part may determine functional and geometrical restrictions to other component parts.

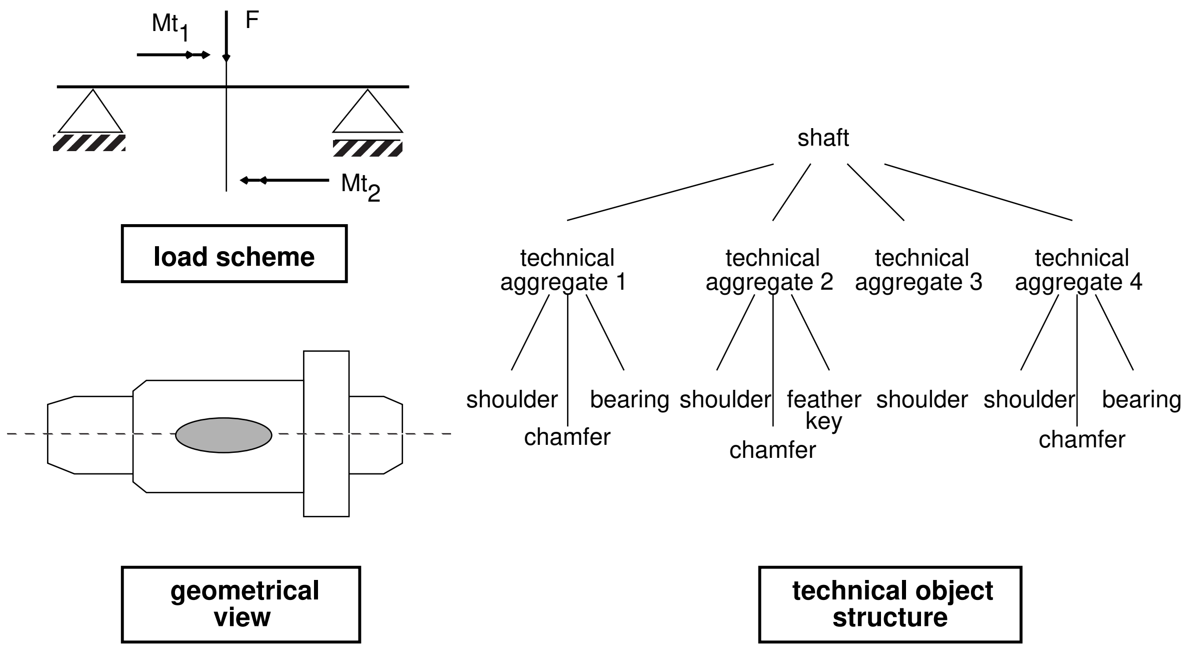

Fig. 1: Load scheme, geometrical view, and technical object structure of a shaft design

Technical objects in conjunction with the technical functional constraints determine the operations for these objects. For example, the technical object 'drill hole' is defined by some functional declarations and given catalogue or standard information which are represented by functional constraints. The execution of a technical operation can be characterized as follows:

- Receipt of constraints of the superior technical objects (e.g., task description)
- Manipulation of the complex object structure with regard to these constraints (e.g., in order to position a bearing at a shaft, the current shoulder has to be marked as bearing place)

- Activation of computation or selection methods
- Propagation and back propagation and redelegation of constraints to associated technical objects.

These considerations illustrate that the constraints are described in object as well as operation semantics. Therefore, the constraints can be classified as follows (ref. to /PS91/):

- Constraints which are determined by the hierarchical decomposition carried out during the top-down design process. For example, the shaft design is determined by a load scheme which is defined in the previous phase of the assembly group design. Thus, these load scheme points influence the component part design since they habe to be taken into consideration during this process.
- Constraints which describe the logical conditions among the technical objects. For example, logical conditions (described by means of predicates and formulas) determine or restrict the domain of technical parameters (e.g., diameter, tolerances).
- Constraints which influence the execution of an operation. To these constraints belong among other things, procedures to select a technical object from a catalog and various computational programs (procedure information). Note that the actions following such a selection procedure depends on the kind of object that was chosen.

Now the question arises also which descriptive methods we need to represent these constraints in a technical modeling system in an explicit manner. Regarding the above discussion, it should be clear that we need various mechanisms to represent such constraints. These mechanisms, which will become clear in the following sections of this paper, are necessary in order to:

- integrate some kind of **procedural information** (e.g., calculation procedures)
- represent **heuristic information** (e.g., to accelerate selection methods)
- **attach** and to **automatically activate programs** (e.g., for change propagation)
- integrate **reasoning mechanisms** (e.g., to control the design process).

For these reasons, we believe that conventional information modeling cannot satisfy the requirements of technical modeling. In our approach, a KBMS system is applied to build up a technical modeling system. However, AI technology alone is not sufficient to support all aspects involved in technical modeling. For example, the selection of an appropriate feather key implies the consultation of standard catalogues which cannot be maintained in main memory. In addition to storage and organization of such large amounts of data, technical modeling must be supported by error recovery features and multiple user operations, since engineering design is commonly organized as a cooperative process among several designers. For these reasons, our approach employs a KBMS in order to fulfill the outlined requirements especially for handling the functional constraints.

3. KRISYS Overview

In this section, we briefly introduce the KBMS which is being used as basis for the realization of our technical modeling approach. The system architecture of this KBMS, called KRISYS, is divided into three hierarchically ordered layers /KR89, Ma88b/. The goal of the lowest layer is to efficiently cope with storage and retrieval of knowledge. At this level, most of the issues are related to traditional DB techniques, however applied to large KB: storage structures, access techniques, efficiency, integrity features, transaction support, etc. On top of this layer, an object-centered view of knowledge representation and manipulation is provided for the knowledge engineer. At this (middle) layer, the knowledge model of KRISYS, called KOBRA, is implemented. To keep the end-user or application programs independent from the internal knowledge representation, the highest layer constitutes an external interface where KB contents are viewed in a more abstract manner. This interface is achieved by the knowledge manipulation language of KRISYS, called KOALA /DLM90/.

In this paper, we will concentrate on the modeling aspects of KRISYS. For this reason, the contents of the paper might ignore some important considerations, reflecting implementation aspects of KBMS (see /Ma89/ for details). In KRISYS, the effective support of these modeling requirements is achieved by a mixed knowledge representation framework (provided by the KOBRA model) which focuses equally on declarative, operational, as well as organizational aspects of knowledge. KOBRA allows for an accurate representation of the whole descriptive characteristics of the application domain, i.e., objects, properties, relationships, and constraints. It offers constructs to represent procedural characteristics of the application world such as behavior of the domain objects (methods), reactions to real world events (demons), and situation-action rules to express the problem solving know-how. However, the most important constructs provided by the knowledge model of KRISYS are the mechanisms for knowledge organization. KOBRA supports the most significant abstraction concepts (classification, generalization, association and aggregation), enriching the semantics of its knowledge model with their different built-in reasoning facilities /BMW84,Ma88a/.

Constructs for representing objects

The representation of all these kinds of knowledge are incorporated in one basic concept, called **schema** (not to be confused with a DB-schema!). A schema (others call it frame, unit, or object) is uniquely identified by a name (i.e., object-identifier) and contains a set of attributes to describe its characteristics. Schemas are persistently stored by the lowest layer of the system, which also guarantees efficient access to KB contents by means of an application buffer /Ma88b, Ma89/. Attributes are used for the representation of properties of a schema and of its relationships to other schemas (**slots**) as well as for the description of behavioral aspects of this entity (**methods**). In order to characterize schemas in more detail, attributes can be further described by **aspects** (possible-values and cardinality specification, default-value, user-defined aspects, etc.).

In Fig. 2, we give an example of a schema representing a particular shaft being designed. The figure illustrates three properties of this object described by the slots diameter, and length, and load-scheme-points, which are governed by integrity constraints expressed by the aspects possible-values and cardinality, i.e., changes of the slot values violating the given restrictions are automatically rejected by KRISYS. Note that the aspect possible values not only rules the value class of the attributes but also can be used to ensure the referential integrity of a relationship. In the example, we have represented technical objects (like a particular shaft) and functional objects (like / power-transmission points) separately in accordance with the different partial models involved in our application. In order to assure that the values of the slot load-scheme-points, which represents a relationship between a technical and a functional object, always reference an actual point, we have asserted 'INSTANCE-OF points' as the value of the possible-values aspect.

```
shaft47
 diameter 4.25
   possible-values (real > 0)
   unit (inches)
   cardinality [1 1]
 length 10.17
   possible-values (real > 0)
   unit (inches)
   cardinality [1 1]
load-scheme-points: p1
   possible-values (instance-of points)
      :
```

Fig. 2: The representation of a shaft object

Constructs for structuring knowledge

Organizational knowledge is specified by means of the **abstraction concepts** which are incorporated into the model by means of special, system-controlled attributes. Each schema can be related to other objects by means of any abstraction concept. Thus, classification/generalization as well as association and aggregation form a directed acyclic graph (sometimes called lattice) rooted in a system-defined schema. Since each schema can be a node in each of these graphs, the KB can be seen as the superposition of three graphs. The same object can, for example, represent a class with respect to one object and a set or even an instance with respect to another. In other words, KOBRA supports an **integrated view of KB objects**, i.e., there are no separate representations for sets, classes, instances, or complex objects. Therefore, the difference between data and meta-data, which is usually apparent in existing data models, is eliminated in KOBRA so that meta-information is integrated into the KB. The semantics of the abstraction concepts are guaranteed by **built-in reasoning facilities** provided by the system [Ma88a]. Inheritance, for example, is the reasoning about the attributes of an object, which is automatically performed via the generalization and classification relationships.

Fig. 3 shows the modeling of the technical object structure using the general abstraction concepts as supported by the knowledge model of KRISYS. An assembly group is aggregated by other assembly group or by component parts. A component part has three subclasses, namely standard part, supplied part, and drafted part. Furthermore, a drafted part is aggregated by technical aggregates which are in turn an aggregation of primary elements, functional elements, and secondary elements. Each of these element classes has several subclasses according to the technical object in the design process (remember that we are only regarding the shaft design). The representation of a shaft actually being designed (e.g., 'shaft47') is performed by an instantiation of the previously mentioned technical objects, when the properties of such a design are given values. For example, the 'shaft47' in Fig. 3 has the slots previously presented in Fig. 2 and some further methods which will be introduced later on.

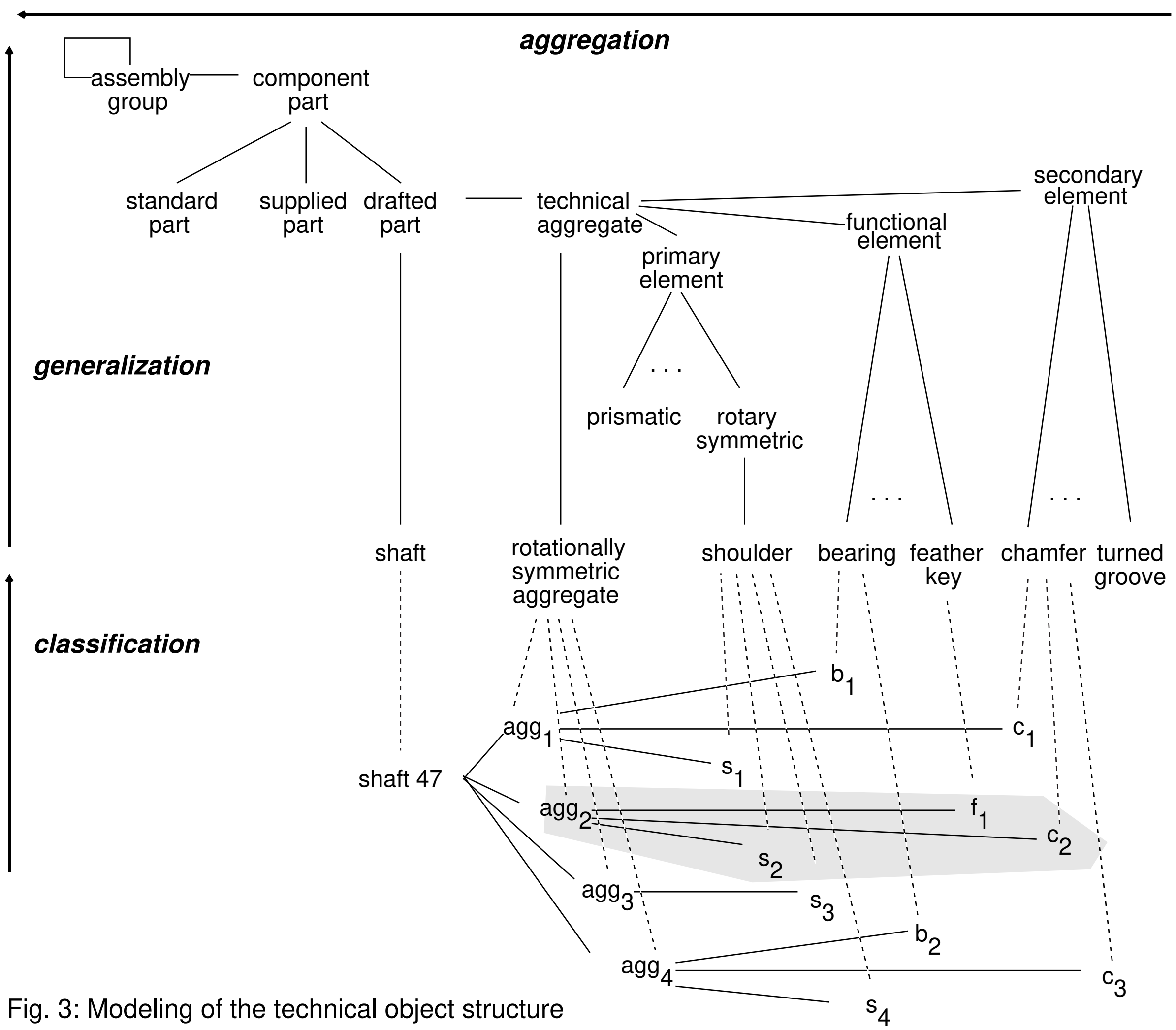

Fig. 3: Modeling of the technical object structure

The support of all these abstraction concepts is one of the aspects that differentiate KRISYS from existing expert system tools like ART /Cl85,Wi84/, KEE /FK85,Fi88/, KNOWLEDGE CRAFT / Ca87,FWA85/, LOOPS /BS83,SB86/, etc. as well as from existing DBS. Whereas KRISYS puts special emphasis on a complete support of the abstraction concepts, these systems neglect the existence of some of them (generalization by DBS, association by DBS, KEE, and LOOPS, and aggregation by all of them), forcing a substantial amount of real world semantics to be maintained in the application programs. Thus, KRISYS can use the semantic of these concepts as the basis for drawing conclusions about the objects and for maintaining the integrity of the knowledge base, thereby not weakening the expressiveness and the semantic power of its knowledge model.

Constructs for manipulating knowledge

As already mentioned, behavioral aspects of an entity are represented by means of **methods** (sometimes called 'stored procedures') which can be activated by sending messages to the objects. Methods can be used to implement algorithms that are necessary to perform particular tasks, e.g., to compute the strength values of some technical elements. For this purpose, they may manipulate the attributes of the object in which they are stored, send further messages to other objects, as well as require the execution of an inference process.

KOBRA provides two further concepts for the specification of operational knowledge: **demons** and **general reasoning facilities**. Demons are special kinds of objects (i.e., schemas) containing procedural information that can be attached to attributes of any other object in order to be automatically activated when the attributes are accessed. Similar to all other objects in a KRISYS KB, demons are represented as schemas which are organized in a hierarchy having the predefined schema DEMONS at the top. This schema has several subclasses: (GET, PUT, SEND, etc.), each containing two predefined instancemethods: <subclass name>-BEFORE and <subclass name>-AFTER. As an instance of one of these subclasses of DEMONS, a user-defined demon inherits the corresponding method attributes (GET-BEFORE, GET-AFTER, PUT-BEFORE, ... SEND-BEFORE, SEND-AFTER), in which the user stores the code of the respective attached procedure (Fig. 4). Thus, KRISYS enables the user to specify the time of demon activation. For example, a procedure stored in a GET-BEFORE method will be always activated before the actual access to the corresponding attribute occurs. The same happens in the case of a PUT-AFTER demon which will be activated after the execution of the update operation on the corresponding attribute. This flexibility allows the use of demons for many different purposes. For example, demons applied to represent implicit information should be activated before accessing an attribute (get-before), those used to check constraint violations should be activated after an update (put-after), etc.

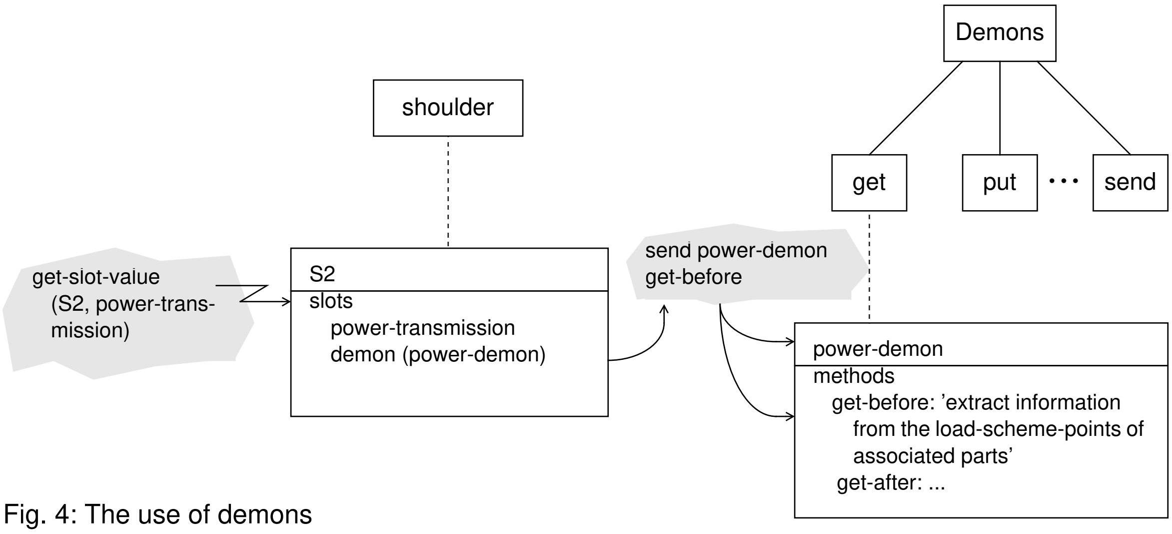

Fig. 4: The use of demons

The linkage between an attribute ('power-transmission' of the schema 's2' in Fig. 4) and the corresponding demon is done by the specification of the name of the demon schema ('power-demon' in Fig. 4) as value of the aspect DEMON of this attribute. Whenever an access to the attribute is made, KRISYS checks whether there is a DEMON aspect defined for this attribute, activating the corresponding attached procedure by sending a message to the schema specified there. Remember that the kind of access (get, put, etc.) determines the attached procedure to be evaluated. That is, a procedure specified in a GET method (either before or after) will be invoked if a get-access has been issued to the attribute, a PUT procedure is invoked by issuing a put-access, etc. Fig. 4 illustrates the use of a demon to extract information stored in different objects or attributes of the KB when it is required. That is, when the slot 'power-transmission' of the shoulder 'S2' is accessed, the demon 'power-demon' is automatically activated to extract information such as power, torque, etc. from the corresponding objects. Since demons are implemented as methods of special system objects, they have acess to the attributes of other objects, may send further messages, activate inference processes, etc.

The general reasoning facility of KRISYS is provided by user-defined rules whose contents, i.e., the conditions (if-part) and actions (then-part) of the rule, are specified by means of KOALA predicates. Rules can be flexibly grouped together into rule sets to better organize the KB. For example, all rules applied to choose the most appropriate feather key for a particular shaft under design are grouped together into the rule set 'choice-of-feather-key'. KOBRA supports inference strategies for forward and backward reasoning which are activated with respect to such rule sets. That is, all rules of the set specified at the activation are evaluated in the 'direction' required (i.e., forward or backward). In order to influence the course of inference processes, the user can specify flexible control parameters as conflict resolvers, search strategies, termination conditions, etc.

4. Performing a Technical Operation

After illustrating the representation of technical objects and the mechanisms that can be used to express integrity constraints, this chapter will discuss the execution of operations on technical objects with special regard to the technical constraints. The reader should imagine the design process of a shaft, for which several shoulders have already been specified according to the above described representation, and the designer has just called an operation to place a feather key (compare with Fig. 1). At the user interface of a technical modeling system, such an operation is performed in a single step, in which the designer gives the necessary input information, e.g., the shoulder in which the feather key should be located and its material. The operation may, for example, be realized as a method of the appropriate object (i.e., the technical aggregate), which is activated by the application program realizing the user interface (see Fig. 5, step 1). Internally, however, this operation is performed in several different steps as described in the following.

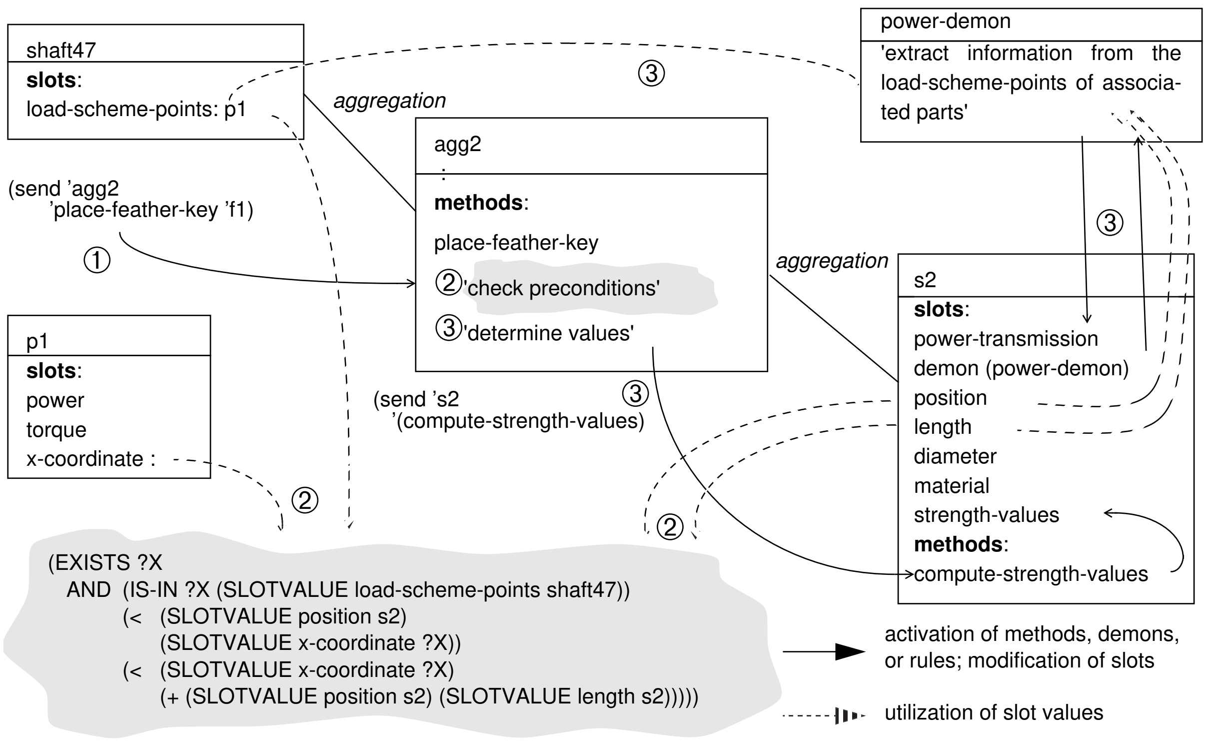

Fig. 5: Placing a feather key: checking preconditions and determining necessary information

Checking preconditions

Before starting the whole placing process, it is necessary to check whether a power feed has been specified for the corresponding shoulder. This task demands the analysis of the environment in which the shaft will be applied (i.e., other external technical objects) to pass on information about such external objects, described in the load scheme, to the shaft itself. We have represented the load scheme, which consists of several points of interest (i.e.,transmission points such as 'p1' in Fig. 5) using the slot 'load-scheme-points'. The values of this slot refer to other objects containing additional information, like the power feed which is established when the shaft itself is specified in the assembly group design (see Fig. 5 for details). For the placing operation in our example, an abstraction process concerning this information is then performed in order to generate the kind of knowledge to be used at this stage. In order to determine whether a power feed is intended for the shoulder 's2', we have to check the power feeds in the load scheme of 'shaft47' and compare the x-coordinate of the power feeds to the position (start and end) of the shoulder. To support an explicit representation, we employ a **predicate-based description** of the precondition (see Fig. 5, step 2), which may be utilized by a reasoning process as well as in the code of a method. This is accomplished by the use of KOALA /DLM90, KR89/, the predicate-based language provided by KRISYS for the description of rules and queries. In the case of a successful check of all preconditions, the placing operation continues with the next step.

Determining the necessary information

In order to choose the most appropriate feather key for the shaft being designed, information about the corresponding shoulder and the technical conditions has to be provided for the further steps. Such information (e.g., diameter, power feed, material, etc.), which is represented by different objects or attributes of the involved objects is made available by distinct **demons**, which are automatically activated when the rules or methods access the desired information. For example, the slot 'power-transmission' of the shoulder 's2', which should contain the power transmission value directly relevant to the shoulder, is connected to the demon 'power-demon'. When the slot is accessed, the demon is automatically activated and extracts the information from the load scheme of the related shaft (see Fig. 5, step 3).

However, the selection of a feather key does not only involve information that can be extracted from different sources. Much more important is the consideration of strength values of the shaft and the feather key itself, which are provided by **methods** of each of the corresponding technical objects (see for example the method 'compute-strength-values' of schema 's2' in Fig. 5). Such methods are also activated by the rules during the next step of the operation after the information about the shoulder has been provided by the demons.

Choosing the feather key

The next step applies **heuristic rules** which, in turn, exploit all the above mentioned information to perform the actual selection process. The heuristic rules express a lot of knowledge of the designer about the dependencies existing among shaft, feather key, power feed, etc. For example, the height of the feather key must be selected from standard catalogs, considering also the diameter of the shaft; in order to determine the length of the feather key, the height, the power value, and the strength values must be taken into consideration. In Fig. 6, the rule set 'choice-of-feather-key' has as its elements all the distinct rules which are needed to perform the selection. During this selection process (step 4), the values of the slots of our feather key 'f1', which are unknown at the beginning, are determined or calculated by the activation of rules, which may also trigger the execution of demons or utilize the above described methods to carry out mathematical computations.

Connecting the feather key to the shoulder

After having successfully chosen an appropriate feather key, it may now be placed in the corresponding shoulder. At this point, complex integrity constraints have to be considered in order to prevent collisions between different technical functional elements. To check these constraints, the rules again make use of demons to get the information about the location of other power feeds, bearing places, etc. from the load scheme. After this, the feather key is connected to the technical aggregate, terminating the placing operation. However, during this last step, demons are again activated propagating further changes inside and outside the technical model. For example, the length of the shoulder may have to be modified because of the size of the chosen feather key, which is automatically performed by the 'collision-demon', when the relationship between the aggregate 'agg2' and the feather key 'f1' is actually fixed (Fig. 6, step 5). As a consequence, neighboring shoulders have to be shortened or lengthened accordingly in order to keep the whole shaft under the required specification.

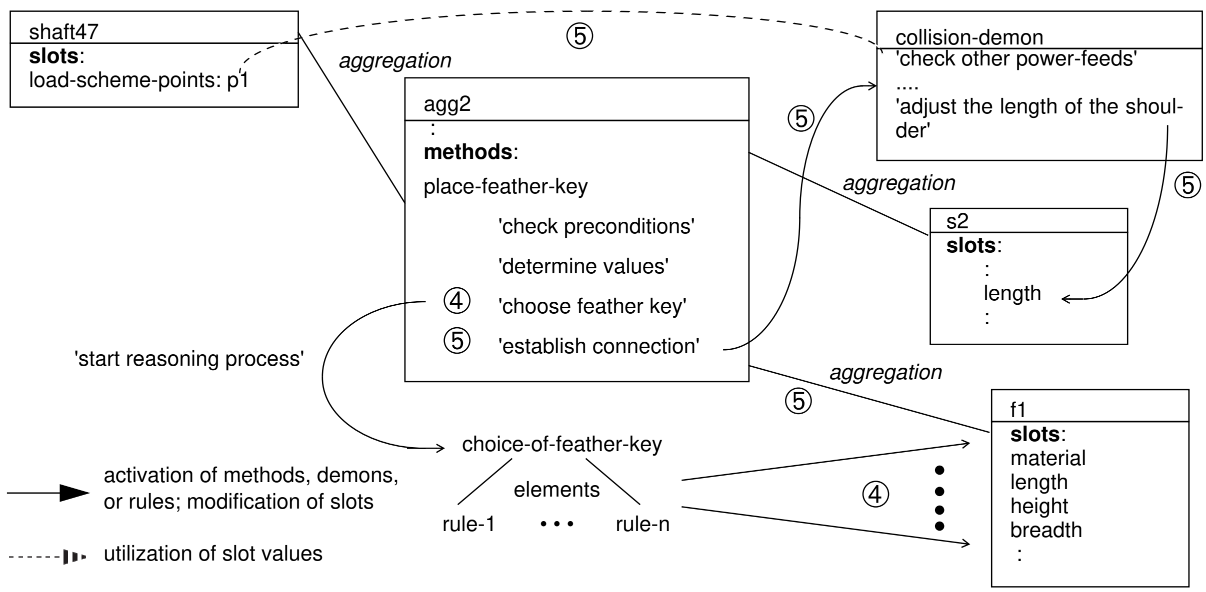

Fig. 6: Placing a feather key: chosing a feather key and connecting it to the aggregate

5. Organization of Constraints

The operation described above apparently involves several types of technical constraints motivating the use of various AI techniques for their maintenance. Such constraints characterize dependencies related to technical objects which govern the changes occurring in the structure and properties of the object during the design process by means of technical operations. Therefore, they have to be represented in our KB in such a way, that their relation to the corresponding operation (e.g., placing) as well as to the involved technical objects (e.g., 'feather-key' and 'technical-aggregate') becomes apparent. Moreover, they have to be properly organized in order to support extensibility of our system (e.g., the definition of a new type of functional element requiring the definition of new constraints or the reorganization or use of existing ones). The heterogeneity of the different techniques involved in the realization of the constraints makes this issue even more important.

Grouping related functional constraints into constraint sets

For an overall organization of technical constraints we have used the abstraction concept of association /Ma88a/: Several (possibly heterogeneous) objects may be regarded as elements of an object set, which, in turn, may be a subset of another set, etc. A partial view of the organization is shown in Fig. 7. All constraints involved in the placing of functional elements are regarded as elements of the set 'place-f-el-constraints'. According to the different types of functional elements, this set is divided into several subsets (e.g., 'feather-key-constraints'), which again may have further subsets describing, for example, the preconditions or the selection process of the feather-key. These sets may have different kinds of elements. For example, the set 'choice-of-feather-key' (which already appeared in Fig. 6) contains rules as its elements, while 'connect-constraints' is a set of demons. The above described sets of constraints are of course only a small subset of the technical constraints that are represented in our technical modeling system. They form a subhierarchy of the set 'technical-constraints', which contains all constraints represented in our system. By means of this organization, all different kinds of constraints related to the same or similar operations and objects are collected and explicitly represented within one conceptual framework, therefore providing an easy way to inspect, modify, or extend the existing technical constraints.

Activating constraint sets

For activating the different sets of constraints, every constraint set possesses a method called 'activate'. These methods were individually specified; for this reason, they carry out different operations according to their subsets or elements. For example, the 'activate'-method of 'choice-of-feather-key' simply starts

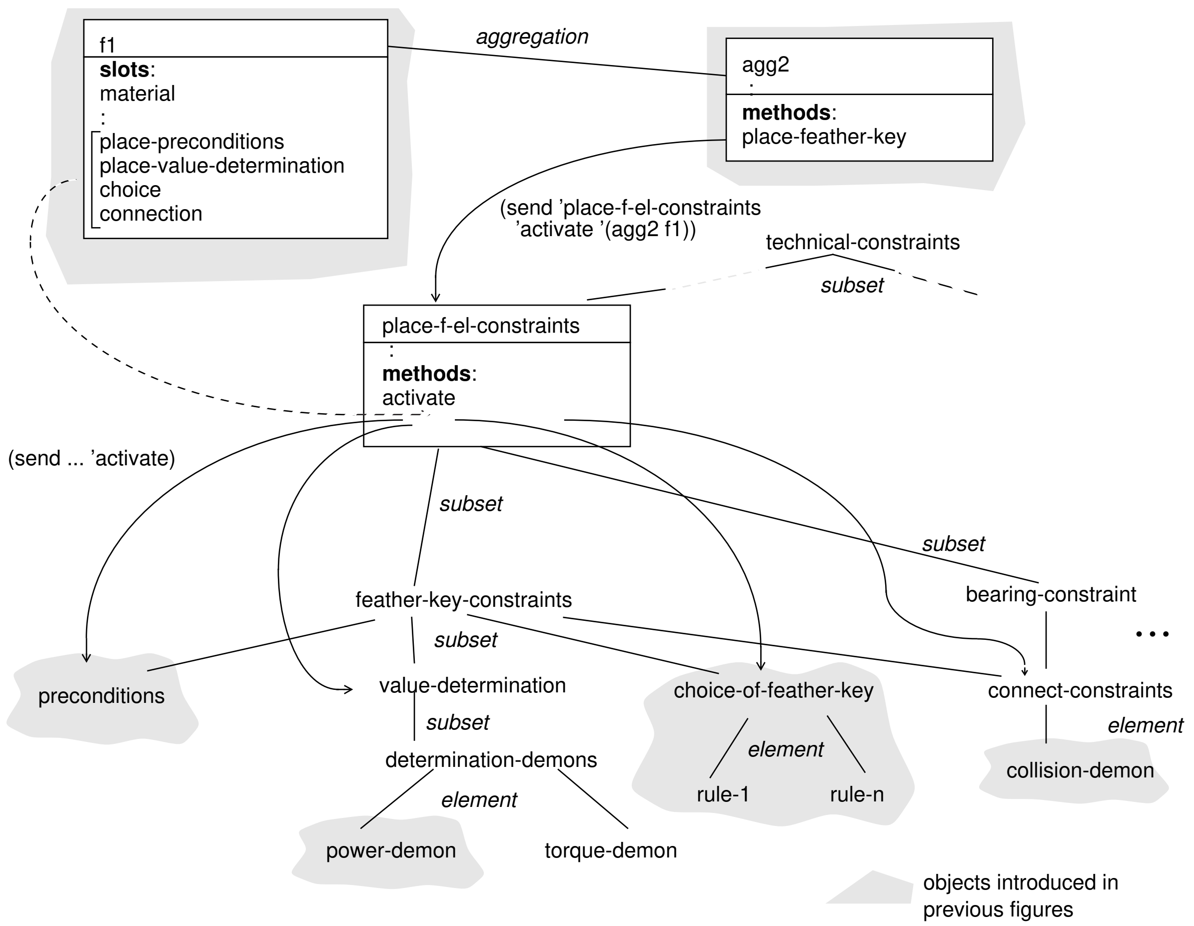

Fig. 7: Organization and activation of technical constraints

a forward-chaining process involving all the rules relevant for this task (i.e., its elements), while the activation of 'value-determination' sends messages to the technical objects involved (e.g., the shoulder s1) in order to compute the strength values (see step 4 in Fig. 6). (Please note that the 'activate'-method of a set of demons, for example the 'connect-demons' or the 'determination-demons', does not directly perform any task at all, since demons are implicitly activated by an access to certain object attributes. Sets of demons are therefore defined only for organizational purposes). This way of activating the constraints makes it easy to modify the realization of constraints, if necessary, because the procedural interface of the constraint sets (i.e., the method "activate") is always the same. For example, if we decide additionally to use rules for the determination of values, we may add a new set containing these rules as a subset of 'value-determination' and modify the "activate"-method, so that it also activates the new rule-set. This change is completely hidden from the objects or operations relying on these constraints.

Realizing a 'generic' approach for specifying technical operations

The above described type of activation also supports a more generic version of the placing operation, which is valid for all different kinds of functional elements and involves checking preconditions, determining attribute values, choosing functional objects, and carrying out collision checks. This generic operation may itself be regarded as a kind of constraint, which is implemented as the 'activate'-method of the set 'place-f-el-constraints'. This method is then activated by the different 'place'-operations (e.g., 'place-feather-key') within our technical objects, and again utilizes object-specific information to activate the right sets of constraints: the names of the constraint-sets relevant for different tasks and related to specific types of objects are individually stored in the involved technical objects as values of certain slots (e.g., 'place-preconditions', see Fig. 7), and are used for the determination of the relevant constraint sets by the 'activate'-method of 'place-f-el-constraints'. Since the names of the constraint-sets are specific to the type of technical objects involved (e.g., feather key, functional element), the values of the slots can be fixed in the appropriate classes and are then inherited by their instances (here 'f1').

Performing queries on functional constraints

Besides the applicability of the association concept for organizational purposes, the representation of technical constraints and sets of constraints as objects of the KB provides an additional advantage. Constraints may be described by arbitrary attributes, which can be used to formulate queries. For example, a lot of constraints used to ensure the manufacturability of the design object may be related to the machines available for manufacturing, which may be used to describe the constraints by means of an additional attribute 'related-machine'. In order to update the functional constraints upon the replacement of a machine by a better one, the KB-engineer simply has to query the constraint hierarchy using the 'related-machine'-attribute to find out, which constraints are affected.

From the description above, it should have become clear that the way such constraints are organized and activated in our KB offers a high degree of modularity, flexibility, and extensibility, which is a prerequisite for dealing with changes and evolution within our application domain and with the complexity involved in the relevant operations.

6. Conclusions and Outlook

A technical modeling system provides the functionality necessary to design artifacts in terms of technical objects (e.g. bearing, shaft, etc.) using object specific operations. A central issue in the realization of such a system is the consideration and enforcement of functional constraints, which represent (probably very complex) dependencies among technical objects or between objects of different partial models of the product model.

In this paper, we have characterized different types of functional constraints and sketched the problems arising from the need to maintain them in a technical modeling system. Possible solutions by exploiting the KBMS KRISYS were discussed, giving special attention to the modeling of the technical object structure and the organization of the constraints. Further, we have pointed out the extensibility of our approach in order to add new technical elements or operations (including the inherent constraints) in the modeling system.

Although we have only given an overview of the issues introduced in our approach, the ideas presented in this paper give rise to a few concluding remarks. Our example should have made clear that the use of AI techniques provides the support necessary to make technical modeling a successful approach. **Heuristic methods** allow the specification of intelligent selection processes. **Object-oriented methodologies** support the necessary integration of procedures in the technical object descriptions, providing the specification of calculation methods and of the actions of static elements. **Demons** and **rules** may be applied for several purposes: to specifiy complex integrity constraints and interactions between distinct partial models, to control the course of design processes, etc. Finally, in order to guarantee the **extensibility** of the system, an **expressive** and, above all, **explicit representation** of all aspects involved in the technical design (i.e., objects, their interactions, dependencies, and the course of the process) has to be employed to support easy changes and extensions of the technical model as well as of the technical operation involved in the design process.

Our future research efforts in this area will mainly concentrate on two issues: realizing some more complex techincal operations in the presented environment and improving the facilities for the representation and maintenance of the functional constraints. It is our purpose to implement complex operations with KRISYS in order to be able to construct a technical modeling system with an appropriate user interface.

As the reader may have noticed, the KB-designer has to utilize several distinct mechanism of KRISYS for the implementation of constraints (e.g., rules, demons, predicate-based conditions). However, KRISYS does not provide a general notion of a constraint which is independent of its implementation. This has to be achieved by the application (i.e., through the introduction of constraint sets and an appropriate definitions of the 'activate' methods). Therefore, one of our main goals is the development of a general modeling construct for constraints, that somehow integrates or subsumes the existing facilities with respect to their expressiveness, but allows a uniform, high level description of constraint characteristics.

Additionally, we also consider the exploitation of facilities and techniques known from constraint languages /SS80, Bo81/ and constraint logic programming /Co90/ for our general notion of constraint. In this context, the interesting question arises in how far the representational and operational power of such a constraint system could actually affect the user interface of the technical modeling system. For example, the ability of constraint logic programming systems to handle underconstrained variables would allow incomplete descriptions of technical objects or a more general characterization of attribute values in terms of intervals, etc. An evaluation of the possible impact of a more powerful constraint mechanism will therefore also be a topic for future work.

Acknowledgements

T.Härder and R. Paul have contributed to clarify and improve the description of some important issues presented in this paper.

Literature

BM86 Brodie, M.L., Mylopoulos, J. (eds.): On Knowledge Base Management Systems (Integrating Artificial Intelligence and Database Technologies), Topics in Information Systems, Springer-Verlag, New York, 1986.

BMW84 Borgida, A., Mylopoulos, J., Wong, H.K.T.: Generalization/Specialization as a Basis for Software Specification, in: Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds.): On Conceptual modeling (Perspectives from Artificial Intelligence, Databases, and Programming Languages), Topics in Information Systems, Springer-Verlag, New York, 1984, pp. 87-114.

Bo81 Borning, A.: The Programming Language Aspects of ThingLab, A Constranaaniannt-Oriented Simulation Laboratory, in: ACM TOPLAS, Vol. 3, No. 4, pp. 353-387, Oct. 1981.

BS83 Bobrow, D.G., Stefik, M.: The LOOPS Manual, Xerox PARC, Palo Alto, CA, 1983.

Ca87 Carnegie Group Inc.: Knowledge Craft CRL Technical Manual, Version 3.1, Carnegie Group, 1987.

Cl85 Clayton, B.D.: Inference ART Programming Tutorial, Volume 1 Elementary ART Programming, Volume 2 A First Look At Viewpoints, Volume 3 Advanced Topics in ART, Los Angeles, CA, 1985.

Co90 Cohen, J.: Constraint Logic Programming Languages, in: Comm. of the ACM, Vol. 33, No. 7, pp. 52-68, July 1990.

DB89 Dahshan, K.E., Barthes, J.P.: Implementing Constraint Propagation in Mechanical CAD Systems, in: Intelligent CAD Systems II (v. Akman, P.J. W. ten Hagen, P.J. Varkamp eds.), Springer-Verlag, London, Paris, Tokyo, 1989.

DHMM89 Deßloch, S., Härder, T., Mattos, N., Mitschang, B.: KRISYS: KBMS Support for Better CAD Systems, in: Proc. 2nd Int. Conf. on Data and Knowledge Systems for Manufacturing and Engineering, Gaithersburg, Oct. 1989, pp. 172-182.

DLM90 Deßloch, S., Leick, F.-J., Mattos, N.: An Approach to Knowledge Base Languages, internal report, University of Kaiserslautern, submitted for publication.ature Based Modeller, in: Third International Conference on Applications of Artificial Intelligence in Enginering, LA, August 1988.

Fi88 Filman, R.E.: Reasoning with Worlds and Truth Maintenance in a Knowledge-based Programming Environment, in: Communications of the ACM, Vol. 31, No. 4, April 1988, pp. 382-401.

FK85 Fikes, R., Kehler, T.: The Role of Frame-based Representation in Reasoning, in: Communications of the ACM, Vol. 28, No. 9, Sept. 1985, pp. 904-920.

FWA85 Fox, M., Wright, J., Adam, D.: Experience with SRL: an Analysis of a Frame-based Knowledge Representation, Technical Report CMU-CS-81-135, Carnegie-Mellon University, Pittsburgh 1985.

GAPR88 Grabowski, H., Anderl, R., Pätzold, B., Rude, S.: The Development of Advanced Modeling Techniques - Meeting the Challenge of CAD/CAM-Integration, in: Proc. 4th CIM Europe Conf., May 1988.

KR89 The KBMS Prototype KRISYS - User Manual, Version 1.0, Kaiserslautern, West Germany, 1989.

KVY89 Krause, F.-L., Vosgerau, F.H., Yaramanoglu, N.: Implementation of Technical Rules in a Feature Based Modeller, in: Third International Conference on Applications of Artificial Intelligence in Enginering, LA, August 1988.

Ma88a Mattos, N.M.: Abstraction Concepts: the Basis for Data and Knowledge Modeling, ZRI report 3/88, University of Kaiserslautern, in: 7th Int. Conf. on Entity-Relationship Approach, Rom, Italy, Nov. 1988, pp. 331-350.

Ma88b Mattos, N.M.: KRISYS - A Multi-layered Prototype KBMS Supporting Knowledge Independence, in: Proc. Int. Computer Science Conf. - Artificial Intelligence: Theorie and Applications, Hong Kong, Dec. 1988, pp. 31-38.

Ma89 Mattos, N.M: An Approach to Knowledge Base Management - requirements, knowledge representation, and design issues -, Doctoral Thesis, University of Kaiserslautern, Computer Science Department, Kaiserslautern, 1989, Lecture Notes in Artificial Intelligence (subseries of Lecture Notes in Computer Science), Springer, to be published.

Pa89 Paul, R.: A Contribution to the Product Modeling of a Delimitated Object Class (in German), Doctoral thesis, Technical University of Magdeburg, GDR, 1989.

PS90 Paul, R., Sutter, B.: Technical Modeling - An Approach to an Integrated Product Data Management (in German), in: Proc. of GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft, Kaiserslautern, March 1991.

SB86 Stefik, M., Bobrow, D.G.: Object-Oriented Programming: Themes and Variations, in: AI-Magazine, Vol. 6, No. 4, Winter 1986, pp. 40-62.

SG88 Serrano, D., Yassard, D.: Constraint Management in MCAE, in: Intelligent CAD Systems II (v. Akman, P.J. W. ten Hagen, P.J. Varkamp eds.), Springer-Verlag, London, Paris, Tokyo, 1989.

SGL89 Spur, G., Germer, H.-J., Lehmann, C.: Impact of Geometric Modeling for Computer Integrated Manufacturing, in: International Symposium on Advanced Geometric Modeling for Engineering Applications (IFIP & GI), Berlin, FRG, 1989.

SR88 Shah, J.J., Rogers, M.T.: Expert Form Feature Modeling Shell, in: Computer Aided Design, Vol. 20, No. 9, November 1988.

SS80 Sussman, G.J., Steele, G.L.: Constraints - A Language For Expressing Almost-Hierarchical Descriptions, in: Artificial Intelligence, Vol. 14, No. 1, pp. 1-39, Aug. 1980.

vR89 von Rimscha, M.: Feature Modeling and Assembly Modeling - A Unified Approach, in: International Symposium on Advanced Geometric Modeling for Engineering Applications (IFIP & GI), Berlin, FRG, 1989.

WH88 Wang, J., Howard, H.C.: Design-dependent Knowledge for Structural Engineering Design, in: Third International Conference on Applications of Artificial Intelligence in Enginering, LA, August 1988.

Wi84 Williams, C.: ART the Advanced Reasoning Tool: Conceptual Overview, Inference Corporation, Los Angeles, 1984.