ELSEVIER

Computer-Aided Design 35 (2003) 1085–1098

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Selecting and parameterising components using knowledge based configuration and a heuristic that learns and forgets

Dieter Roller[a,1], Ingo Kreuz[b,*]

[a]*Department of Graphical Engineering Systems, University of Stuttgart, Breitwiesenstrasse 20-22, D-70565, Stuttgart, Germany*
[b]*DaimlerChrysler AG, Research and Technology REM/EC, 096 HPC T725, D-70546 Stuttgart, Germany*

Received 24 July 2002; received in revised form 29 October 2002; accepted 1 November 2002

Abstract

In this paper we suggest the use of knowledge based configuration in connection with CAD systems to enhance the ability of reusing models of CAD components and to automate solution processes especially of variational design problems. Configuration systems can support design engineers in choosing components and in checking consistency. We have investigated how configuration mechanisms can be applied to the design of engineering products and introduce our heuristic relevant knowledge first (RKF). This heuristic solves many of the problems which occur with configuration in engineering domains which continuously change, e.g. as a consequence of innovation. We conclude with the presentation of experiments performed with RKF.

Keywords: Coupling of CAD and configuration; Machine learning in design; Forgetting; Reuse of CAD models

1. Background and goals

The focus of modern CAD systems is mainly the reuse of models as well as an appropriate integration into production processes. For better reusability generative declaration forms have been developed, like object or structure oriented and parametric modelling methods for dimension and structure variants [1]. *Parametric libraries* have been defined (e.g. based on DIN V 4000 part 100 or DIN V 66304) and constraints have been introduced as a declarative method of description also with the intention to improve reusability [2]. In this context Klein proposed the following goals [3]: *more expressiveness*, *conceptual and system design*, *interactivity and flexibility*, *knowledge integration*, *retrieval*, *reuse and adaptation*. We particularly want to emphasise the focus on components instead of simple elements of drawings or models (like lines or surfaces) which Klein mentioned in the context of 'conceptual and system design'. Features (recent overview e.g. in Ref. [4]) are used to add semantics to these components and support further processing during production processes. With 'retrieval, reuse and adaptation' Klein summarises the problems stemming from an insufficient structuring of drawing or component databases and a lack of automated searching in these databases. In Ref. [5] it is shown, how these results can also be transferred to the domain of electrical CAD (ECAD).

In addition to the improvements concerning the reusability of model components, the products themselves should be divided into modules: In Ref. [6] it has been described how a module concept may help to reduce the variants needed to be produced whereby at the same time the number of variants that can be offered to the customers at same cost is increased. In addition to that, modularisation enables the production of more individual products leading to another competitive edge.

Configuration systems can generally help to select and parameterise modules of a system. The idea to use configuration systems for CAD problems is not new. For example Brown and Chandrasekaran [7] have already described an expert system ('configurator') for mechanical design in the 80 s.

The goal of our project is the use of knowledge based configuration methods for design tasks in so-called *technical* or *engineering* domains. Recurrent, similar tasks

* Corresponding author. Tel.: +49-711-17-47033; fax: +49-711-3052-111-460.
E-mail addresses: roller@informatik.uni-stuttgart.de (D. Roller), kreuz@daimlerchrysler.com (I. Kreuz).
[1] Tel.: +49-711-7816-304; fax: +49-711-7816-320.

0010-4485/03/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0010-4485(02)00180-X

occur especially in the field of variational design which can be further automated by our approach. The main aspects are:

- The choice and parameterising of modules should be better supported. By consequently using existing modules, the expense of a design is reduced. Modules can more economically be produced as mass products.
- A configuration system can help to ensure that all specifications for a system are met for the modules used.
- Nowadays product variety leads to generic design models where one model describes several variations of the product, which tends to be confusing. A support for the construction of individual system models should at least include an automated import of the models of the selected and parameterised modules into a system design. The goal is a complete support including the placing and orienting of the models.
- Configuration is an NP-complete task. Consequently AI techniques like heuristics are already often used in this field. To improve the acceptance of this approach the solution process should be dramatically speeded up, e.g. by learning from previous solutions.
- Available learning methods lead to conservative solutions. With learning the only motivation for a decision is that there was the same decision before. Innovations and changes in market trends are typical for technical domains. This is why a learning method for configuration in technical domains also has to be able to be retrained.
- Different target groups of a product imply different optimisation goals which lead to different solutions. A learning method should therefore consider different target groups.

2. Configuration

The subject matter of this paper is *not* to give a solution to the layout problem which is sometimes also classified under the term *configuration* [8]. In the context of the layout problem "an assignment of coordinates and orientations of components that minimizes the cost and satisfies certain placement requirements is sought" (see chapter 3 in Ref. [8]). We rather focus on the selection of form features and their parameters (components, objects) and assume that dependencies can be formulated that allow the unambiguous placing and orienting of these components. Several definitions of configuration have been proposed and sometimes configuration is also called *design* (e.g. in Ref. [9]). A definition that corresponds to our understanding of configuration in this paper has been presented by Guenter and Kuehn [10]: Configuration is 'a task of synthesis of domain objects' with the following characteristics:

> A set of objects in the application domain and their properties (parameters), a set of relations between the domain objects where the taxonomical and compositional relations are of particular importance for configuration, a task specification (configuration objectives) that specifies the demands a created configuration has to accomplish and control knowledge about the configuration process.

Several methods for knowledge-based configuration exist (chapter 2 in Ref. [10]). Hereinafter we will focus on the so-called object- or structure based approach. Technical products are typically structured in a hierarchical way which can be most adequately modelled within this approach: All configurable components are stored as 'objects' or 'frames'. Their relationships are represented by graphs where the taxonomic (is-a) hierarchy and the compositional (consist-of) hierarchy are most important. The figures in Section 5 show examples of such hierarchies of our example domain.

3. CAD and configuration

Yan, Rehmann and Borg raise in Ref. [11]: "It is well known that mechanical products are artefacts composed by designers using past and new basic building blocks/primitives […]". These 'building blocks' (components, (form-) features) are typically stored in a library for reuse in various designs. After a component has been chosen, parameters like size and position, the length of single edges or parameters need to be adjusted. In this case the construction of a product is reduced to select the components of a system and to determine their parameters. However, these are exactly the steps in development, which can ideally be supported by a configuration system. In this context the construction of a system can be seen as a configuration task.

To start a configuration, a task specification has to be defined by a user. This specification includes above all the selection of the target system concept which needs to be configured (e.g. 'Backrest Drive' in the example of Fig. 6). After that the configuration system supports the user by determining all components of the target system along the compositional hierarchy. It can also offer or even set possible numbers (e.g. *three* cylindrical gear pairs) and specialisations (e.g. motor → electric motor → EMot2[2]) together with values for their parameters. The numbers and specialisations of the components can either be set by the user or by the configuration system. An automatic selection can be performed by the configurator either as soon as exactly one possible alternative is left or by heuristics. When a next decision is made all modelled restrictions have to be taken into account at all times concerning the parts of the solution chosen so far. Restrictions can describe any type of dependencies including morphological or spatial restrictions. An example for a technical dependency is given in Fig. 9, where two $n : m$

[2] → means 'specialise'.

relations are used to model the fact that these types of motors need a worm gear to make the backrest drives self blocking. In Section 3.1 the formalism that we use in our prototype to model spatial dependencies is presented. Heuristics as well as user selections might lead to conflicts, i.e. no more components or parameters can be chosen without violating a dependency in the target system. At that point decisions have to be withdrawn. A simple strategy to perform this is 'chronological backtracking' where (recursively) the last decision is drawn back and the next alternative is chosen. Various strategies can be used by a configuration system to find out the decision that should be withdrawn [12]. The result of configuration is a (structured) bill of material that can be transformed into a CAD model.

To use knowledge-based configuration together with CAD we had to consider how these functionalities can be combined in our project. In principle this combination can be implemented by using a *homogeneous system concept*, *integration* or *coupling*. An overview over these possibilities is given in Section 2.2.3 of Ref. [13]. The focus of our project lies on finding suitable configuration methods for CAD and not on implementing CAD functions. A homogeneous system concept was therefore unsuitable for our prototype.

Another possibility to combine a CAD system and a configuration system is to *integrate* the configuration kernel directly into the CAD kernel. In this case the configuration system has direct access to graphical information which represents additional knowledge to that stored in the knowledge base of the configuration system. The same is true if a CAD-Kernel is built into a configurations system. The difference is the mode of control: With a configurations system with *integrated* CAD kernel the solution process is controlled by the configurator. Starting from a task specification the configurator tries to find a solution on the basis of the knowledge modelled in the configuration knowledge base. For each instantiated concept the CAD kernel is called to generate the referenced graphical model (e.g. by importing it from a library). On the other hand the construction of a graphical model controlled by an engineer is the focus in a CAD system. In this case an *integrated* configurator kernel can be used to automate at least some construction steps; i.e. decisions can be made (choose CAD models, parameterise models,…) which can be derived form the modelled configuration knowledge and the hitherto solution.

At present this direct integration often also fails especially because configuration and CAD kernels are as a rule developed independently and are produced by different companies. Moreover the data structures of CAD and configuration systems are typically very different which makes a complete integration difficult. From a software engineering point of view one should argue anyway that both functionalities are already complex and should at least remain part of separate modules. As encapsulated modules they can communicate over defined interfaces.

As long as the kernels remain separated, it must be kept in mind that a configurator typically cannot interpret graphical information to use it during the solution process. For example it cannot calculate whether two complex 3D bodies intersect each other. However, it makes no sense to model the graphical information redundantly in the configurator's knowledge base just to allow direct access for the configurator to that information. As soon as the configuration kernel is able to communicate with the CAD kernel, the CAD kernel can be called to interpret its graphical models and to perform the necessary calculations. The above mentioned test for intersection of two bodies and the placement of two bodies which should have a certain distance are examples for calculations with graphical dependencies.

CAD kernel and configuration kernel can also remain as independently developed applications as soon as an interface is provided through mechanisms of the applied operating system. An advantage of this solution is that different configurators can be combined with different CAD systems. Just the application specific instructions of the interface have to be adapted. Most operating systems support interfaces to make calls from one application to another. For example applications can perform a so-called 'remote procedure call' (RPC) in many operating systems (e.g. UNIX, Windows). Other standardised mechanisms include CORBA (Common Object Request Broker Architecture) and in the operating system Windows the interface object linking and embedding (OLE) and distributed common object model (DCOM). The CAD systems Inventor and AutoCAD from AutoDesk are examples of commercial CAD systems that support calls via the OLE interface of Windows.

In Fig. 1 the coupling of a configurator and a CAD system is depicted where all modules remain encapsulated, i.e. they occur as independent modules or applications.

The configurator kernel generates a bill of material ('configuration') on the basis of the descriptions of the components as well as the dependencies that are modelled in its knowledge base. For the calculations or verifications of dependencies in which graphical information or components are involved (e.g. 'calculate the x-position of component A in such a way that component A just does not intersect component B') can be performed by a call to the CAD kernel (see arrows 'Call for Graphical Calculation' and 'Result' in the centre of Fig. 1). Results can be used to check dependencies (like rules of the kind 'component A must not intersect component B') or to calculate parameters, especially the placing and orientation of components.

For these calculations references between the components in the knowledge base and the graphical library have to be modelled. This can be the unique keys of the components in the CAD component library which can be stored as an additional attribute of the accompanying concept in the knowledge base. User interactions during

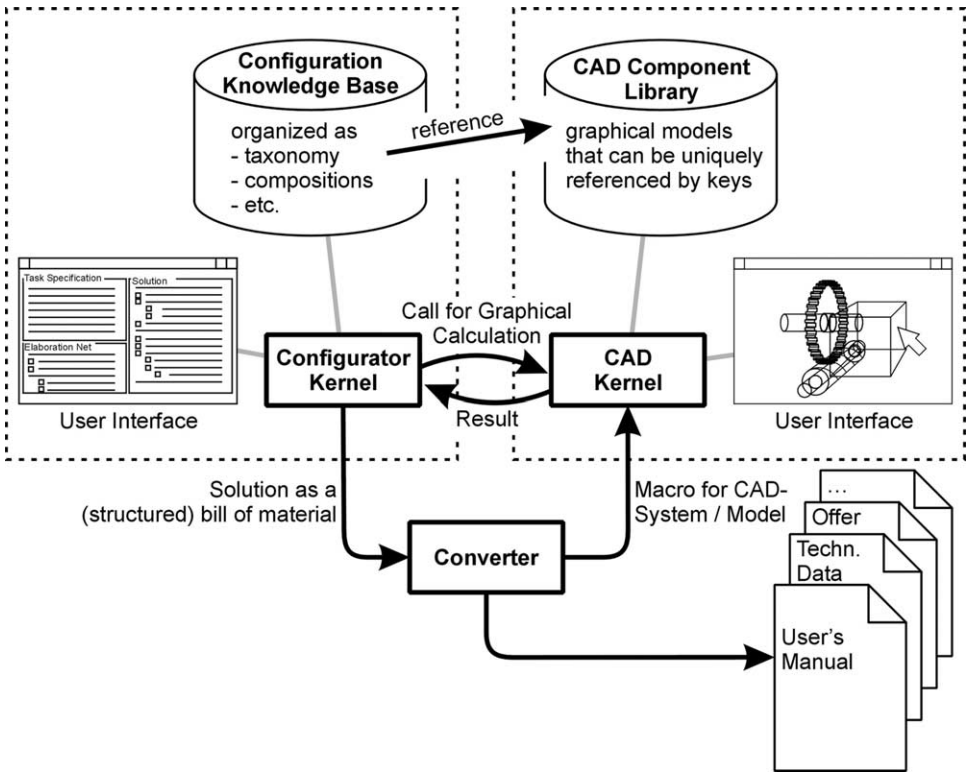

Fig. 1. External coupling of configuration and CAD.

the configuration process are controlled via the user interfaces of the configurator or CAD system.

The generation of the final graphical representation for the generated structure is also represented as an encapsulated unit 'converter' which belongs neither to the configurator nor to the CAD kernel. For the conversion of the bill of material the converter needs graphical information about the components of the bill of material. For this the references to the graphical component models have to be written into the generated bill of material by the configurator. Also the parameters of the graphical models, like height or width, should form a part of that list.

The converter can use the command interface of the CAD kernel to let it generate the graphical model: For every component of the bill of material the referenced graphical model has to be imported, parameterised, orientated and placed using the parameters which are also part of components of the bill of material.

During the configuration process it might become necessary for the converter to export an incomplete construction to the CAD system. This enables the CAD kernel to make calculations which in turn affect decisions during the configuration process. For example the number of graphical objects to be instantiated and placed can depend on the geometry and size of the objects, the strategy used for placing and particularly the available space. If steps like this are frequent in a domain the strict dividing of the modules might lead to performance problems. In all of our investigated domains (especially the 'backrest drive', see chapter 5) it was possible to model enough knowledge already in the knowledge base of the configurator. The export to the CAD system could be performed completely *after* the configuration process. Of course knowledge should be modelled with minimum redundancy to ensure its maintainability.

If the converter is able to access the information of the graphical library directly (e.g. because a standardised file format is used) then the graphical model can be generated directly by the converter (see 'model' near the arrow from the converter to the CAD kernel in Fig. 1). The main advantage of this approach is that it is only dependant on standard formats. This mechanism of generating a file from the configuration can also be used for other 'generated documents' like specific offers, individual manuals or individual diagnosis information (see arrows near the right bottom of Fig. 1).

Both ways from the converter to a CAD application have been implemented in the prototype which will be introduced in chapter 5: It has the ability to control a CAD-System via its OLE interface and to generate models in the standard format VRML.

3.1. Spatial dependencies

As soon as all components of a system have been selected and parameterised by a configurator their graphical representations have to be placed. For this the converter needs information about the position and orientation of

the components. Since the modelling itself and especially the modelling of spatial dependencies is not a focus of this paper, we will only outline this problem briefly.

The placing of components can be understood as a layout problem which 'involves the placement of components in an available space such that a set of objectives can be optimised while satisfying optional spatial or performance constraints' [8]. In their contribution Cagan, Shimada and Yin give a survey of actual papers in this domain. They describe the common layout problem as an optimisation problem which is complex especially because the search space is not continuous but fractal. Various AI search and optimisation methods have been applied to solve this problem and the authors conclude that a hybrid approach should be used, where multiple methods are combined. Finally some forms of representation for bodies and their suitability for intersection tests are addressed in their article.

The approach which is introduced in this section does not try to solve the common layout problem. The targeted application field is restricted to domains in which a product is divided into modules and their graphical representations only need to be parameterised, placed and oriented. This approach includes a method to model spatial dependencies for mechanical components in a way that a configurator or a converter can interpret them.

An obvious approach is to model the coordinates (x, y, z) and solid angles (a_x, a_y, a_z) of the components as parameters that have to be configured by the configuration system. This approach has been used by Kopisch and Guenter [14] for a two dimensional problem: They used the configuration prototype KONWERK to arrange seats, kitchens, toilettes etc. in the cabins of Airbus A340 aircrafts.

To enable the configuration system to place components, knowledge like dependencies between component positions and strategies for placing have to be modelled. For the backrest drive example (see chapter 5) we have drawn up the following concept: Named reference points (see the Xs in Fig. 2) and a reference direction (arrows) are defined for each component.

To model the arrangement of Fig. 3 the following four dependencies need to be defined which can be checked and evaluated by the constraint mechanism of our configuration system:

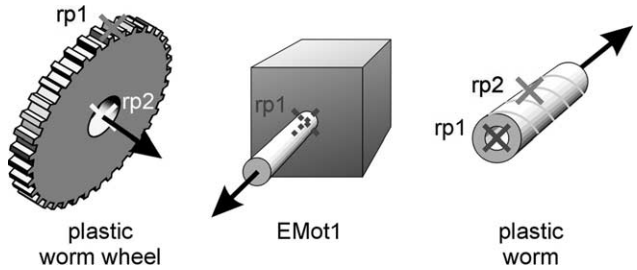

Fig. 2. Reference points and directions for the modelling of spatial dependencies.

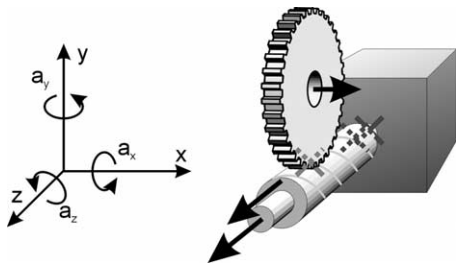

Fig. 3. The arrangement to be modeled.

worm wheel.rp1 = worm.rp2

electric motor.rp1 = worm.rp1 + (0, 0, worm wheel.radius − worm.length 0.5 + 2)

angle(worm wheel, worm) = (a_x := 0°, a_y := 90°, a_z := 0°)

angle(electric motor, worm) = (a_x := 0°, a_y := 0°, a_z := n°).

The above dependencies are generic in so far as abstract concepts are used like 'electric motor' or 'worm' in the equations. This ensures that the dependencies are valid regardless which specialisation ('EMot1',…, 'EMot5' or 'plastic worm', 'steel worm') is used. The reference points and directions have to be modelled for each specialisation (like in the example of Fig. 2). If modelled like this the dependencies (equations) also need not to be altered if further new leaf concepts (electric motors,…) are added to the domain. The second equation is an example of a parametric modelling of a dependency: The relative z position of the reference points of the electric motor and the worm are dependent on the radius of the worm wheel and the length of the worm to ensure that the motor and the worm wheel do not intersect (gap of 2 mm). If any degrees of freedom appear, strategies or heuristics can be used to determine values for the parameters concerned. In the above example the a_z in the last dependency can be chosen freely, which will affect the angle of all subsequent elements of the power train. A sensible heuristic in this case can be a default value of zero.

In many domains the placing of components is trivial for humans as soon as all components are selected and parameterised whereas the modelling of the spatial dependencies could be complicated and difficult to maintain. Due to this it is sometimes sensible to restrict the converter to selecting and parameterising the components.

3.2. Problems with configuration in CAD domains

In Ref. [7] three different classes of problems for configuration tasks are distinguished ('design' here again is a synonym for 'configuration'): class 1: *creative design*, class 2: *innovative design*, class 3: *routine design*. CAD tasks can mostly be called innovative! As methods for innovative configuration are not thoroughly investigated yet and no tools for this problem class exist, we have restricted

our investigation to the problem class 'routine design'. This class contains all CAD tasks that are connected with variational design [5].

Even if configuration is restricted to routine design, not all decisions that appear during a configuration process can be calculated or 'tried out' by a configuration system. The reason for this is the huge search space which occurs even for the simplest examples because of the exponential number of combinatorial possibilities to combine modules. In Section 2.2.1 of Ref. [15] this effect is called 'combinatorial explosion'. The simple backrest drive example described in chapter 5 already implies more than 44 million possible combinations of the modules (ignoring their different parameters). Our prototypic configurator is able to build and test about 3000 combinations per second and would therefore need more than 4 h to scan the complete search space. This example shows that it is most often pointless to scan the complete search space of a configuration domain, i.e. to try all possible combinations.

Possibly the most important application fields for computer aided design are engineering domains, i.e. the construction of technical systems. The growing use of electronic components in such systems causes the innovation cycles in all engineering domains to adapt to the rapid changes of these components. As a consequence, configuration knowledge is subjected to frequent changes as it must be continuously updated to take technical innovations into consideration. However, knowledge bases that contain heuristics or strategies for the configuration of such systems are usually fixed and not able to model innovations in an adequate way.

4. Relevant knowledge first

'Machine learning in design' is already used to improve the design process and the quality of the design solutions (see section 3.7 in Ref. [16]). In Section 4 we introduce a method called relevant knowledge first (RKF) which also contains a machine learning component. Besides *learning* RKF also uses a *forgetting* function which enables the configuration process to take changes (innovations) adequately into account. We will point out how RKF supports configuration in engineering domains by solving the above mentioned problems and finally classify RKF into the 'dimensions of learning in design' defined by Grecu and Brown [16].

- RKF has been introduced during the AAAI Workshop 'Configuration' [17] and during the ECAI 2000 Workshop 'PUK2000' [18]. It can be used to reduce the problems stated in the last section by helping to make the right decisions during an actual configuration task. Decisions during configuration using a structure based approach include
  - the determination of the number of sub-components for all components (composition, decomposition or integration step)
  - determination of the specialisations of all components (specialisation step)
  - determination of all parameters for all components (parameterising step).

As soon as one of these determinations cannot be calculated the user has to provide the decision. For similar tasks similar decisions are provided by users leading to similar results. This observation led to the so-called case based approach. Here selected solutions ('cases') are stored in a case base. Every task specification is compared with the task specification of the stored cases, and then the most similar is chosen and adapted to the actual exact task specification. Obviously the number of questions to the user is reduced by this approach and cases are widely used in the context of Artificial Intelligence in Engineering Design (e.g. Ref. [19]). However, this procedure is conservative: Imagine for example a typical configuration result for a bicycle back in the year 1900. This case would be worthless for today's bicycle configuration tasks. Consequently in RKF no fixed cases are used. However, we observed that it is useful to assess the relevance of knowledge in separated contexts in order to be able to focus on an actual configuration problem. For the relevance there are two deciding factors which correspond to the antagonism between conservatism and innovation: on the one hand knowledge is very likely to be useful again if it has already been useful for similar tasks. On the other hand *new* information should be preferred, in order to obtain innovative solutions and to avoid conservatism. As an assessment for the relevance of knowledge we calculate its so-called *relevance for a given context* from the time of the last access and its utility which is learnt by reinforcement learning mechanisms for that context. A context implicitly specifies a set of 'similar tasks' and will subsequently be called 'task-class'. Therefore every real task has to be assigned to a task-class as part of the task specification before RKF can be used for a configuration process.

With RKF the search for solutions is supported by preferring *relevant* objects of a knowledge base. Every time a decision has to be made, the relevance for all pieces of knowledge *in question* is calculated for the context of interest and *the most relevant* object is used with highest probability. Therefore a probability distribution is calculated that is proportional to the relevance of these objects. For subsequent search processes the relevance is increased for successfully used objects (reinforcement learning/training) and decreases for all other objects (forgetting, fading). For this, solutions have to be assessed by the users. For all objects used during the solution processes their relevance is increased in relation to this assessment. Objects which get low or even no rewards gradually become irrelevant on

the basis of aging ('forgetting' or 'fading'). These principles of RKF will now be described in more detail:

The relevance of an object o at a point in time t in the context of a task class c is calculated as functions of the time since the last use of the object ('age', during forgetting-phases) and the rewards received from the users (during a training phase) whereby training compensates forgetting.

$$\mathrm{rel}(o,t,c) := \begin{cases} \mathrm{train}(o,t,c), & \text{learn if } o \text{ was part of the solution} \\ \mathrm{forget}(o,t,c), & \text{forget if } o \text{ was not part of the solution} \end{cases} \quad (1)$$

The functions 'forget' and 'train' describe in which way something is learnt or forgotten. The field of cognitive psychology provided a starting point for our investigations in these processes. In this context Anderson [20] states investigations that led to the 'power law of learning' and the 'power law of forgetting': experiments have shown that forgetting and learning processes are not linear over time but show an exponential character. These power functions show desirable characteristics for learning and forgetting in technical domains and lead us to the following relevance function:

$$\mathrm{rel}(o,t,c) := \begin{cases} \mathrm{rel}(o,t-1,c) + \dfrac{(1-rel(o,t-1,c))}{b_\mathrm{t}}\,\mathrm{reward}(o,t,c) & \text{if } \textbf{train}, \text{ if } o \textbf{ is} \text{ part of the solution,} \\ & \text{start with } \mathrm{lastForgetRel}(o,t,c) \\ \mathrm{lastUseRel}(o,t,c)\cdot b_\mathrm{f}^{-\mathrm{age}(o,\,t,\,c)} & \text{if } \textbf{forget}, \text{ if } o \text{ is } \textbf{not} \text{ part of the solution} \end{cases} \quad (2)$$

with

$\mathrm{age}(o,t,c) \in \{0,1,\ldots\}$, time (number of configuration runs) since a last use of o before t in context c

$b_\mathrm{f} \in (1,\infty)$, base of the forget function for the adaptation of the 'velocity' of forgetting to the characteristics of a domain. The larger b_f the faster objects are forgotten

$b_\mathrm{t} \in (1,\infty)$, base of the learning function for the adaptation of the 'velocity' of learning to the characteristics of a domain. The larger b_t the slower the learning

c task class (context), e.g. a target group

$\mathrm{lastForgetRel}(o,t,c) \in [0,1]$, last relevance value of o before t when the last forgetting phase ended in the context of task class c

$\mathrm{lastUseRel}(o,t,c) \in [0,1]$, relevance value after the last use of o

o object of a knowledge base

$\mathrm{rel}(o,t,c) \in [0,1]$, relevance of o at the point in time t in the context of the task class c

$\mathrm{reward}(o,t,c) \in [0,1]$, reward for an object o at the point in time t in the context of the task class c

$t \in \{0,1\ldots\}$, the number of configuration runs is used as time

The learning portion (upper part of Eq. (2)) implements a reinforcement function for reinforcement-learning where the user's assessment of a design solution (reward) is used as the reinforcement signal (see standard reinforcement-learning model with one state in Ref. [21]). Learning takes place every time a solution has been generated and an assessment for the solution has been provided by a user. The simplest scheme we have investigated for giving rewards was 'a user accepted the design solution (100%) or not (0%)', in the most sophisticated one, a user has to give separate rewards for all components of a solution. The simple assessment scheme lead to slower learning as expected, but learning was still possible in our experiments. With the definition for relevance given by Eq. (2) only a linear amount of additional memory is needed because only the time of the last use and the relevance value at that point in time has to be stored for each object of a knowledge base per task class. The independent consideration of different task classes enables learning even if conflicting goals exist in a domain: per definition conflicting goals must not exist within a single task class. In order to find task classes there are two indicators: as a consequence of the above consideration the task classes can result from the combinations of various global optimisation objectives. For example let the optimisation objectives 'price' and 'performance' be decisive in a domain. The set of task classes can then be automatically generated by combining the objectives for price and performance 'high', 'low' and 'don't care'; e.g. 'low price/high performance', 'don't care about the price/high performance' etc. As a second indicator task classes can be used, which emerge directly from the respective configuration domain: The target groups for a product to be configured.

If we try to determine the start relevance $\mathrm{rel}(o,t_o,c)$ at the moment t_o when o is stored into a knowledge base, we

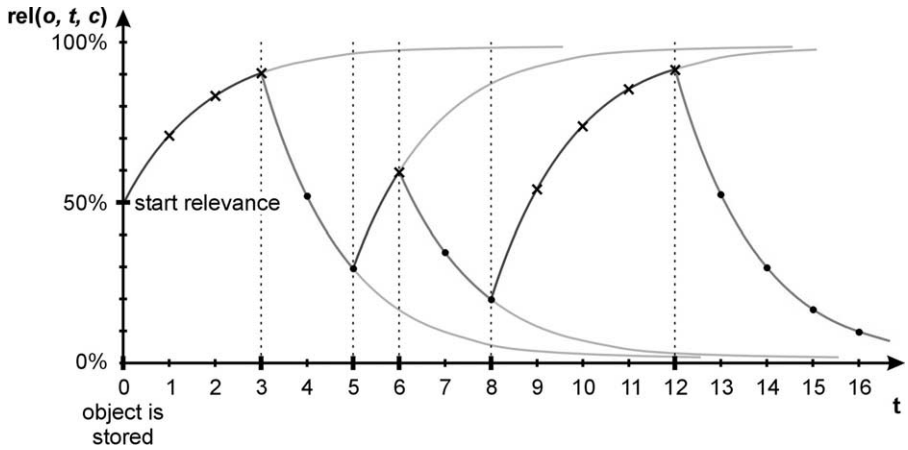

Fig. 4. Relevance function with $b_t = b_f = 2$.

recognise that many parameters of the original function in the model of cognitive psychology are not available. For example Waltz [22] describes how the start relevance depends on the estimation of the importance of the information. In the best case an expert can assess a new object and determine the start relevance for all task classes. If such an assessment is not available we must assume however, that all information in a knowledge base is of the same importance because a computer cannot make 'emotional' assessments. In this case we suggest a start relevance $\mathrm{rel}(o, t_o, c) = 0.5$ based on the experiences made during our experiments. With this value a new object has the same chance to become relevant (approach 100%) or irrelevant (approach 0%).

The following diagram shows an example of the relevance function from Eq. (2):

In this example object o has been stored at $t = 0$. It was part of the solution at $t \in \{1, 2, 3, 6, 9, 10, 11, 12\}$ and received a constant user reward (crosses in Fig. 1). Therefore the intervals (0,3], (5,6] and (8,12] are training phases. During the remaining intervals (forgetting phases) the information ages and from $t = 12$ onwards it ages infinitely (dots in Fig. 4).

As soon as a decision has to be made between similar objects, the most relevant object is selected with highest probability. With this statistical procedure training in 'wrong directions' can be avoided since the second or third most relevant objects still 'have a chance' to prove their worth.

The probability for choosing object o from a set of possible objects $\mathbf{O}$ is calculated by:

$$P(o) := \frac{rel(o)}{\sum_{i=1}^{n} rel(o_i)}, \; n = |\boldsymbol{O}|, \; o, o_i \in \boldsymbol{O} \tag{3}$$

With this method the problems named in Section 3.2 are solved in the following way.

Only those objects that are part of a *valid solution* can receive rewards, i.e. a solution where no constraints, rules or other dependencies are violated. They are becoming more and more relevant and consequently they become more and more likely also to be part of future solutions which in turn makes them relevant more quickly. Finally one object with the highest reward per set of possible objects $\mathbf{O}$ becomes most relevant and the others become irrelevant (approaching 0%). As a consequence all the most relevant objects (per task class) are part of a consistent solution which will be found with high probability. The separated determination of relevance for different task classes ensures that different solutions are learnt, because task classes imply different rewards. The selection of objects is performed randomly, so it is possible that irrelevant objects can still be chosen, but with lower probability or as a consequence of revision if dependencies have changed. An example of speeding up of the solution process by avoiding backtracking with RKF is given in the example of Section 5 (see Fig. 10).

RKF also prefers objects that are *new*. As a consequence new objects are also chosen with high probability as well as the 'well-tried' objects. In the case that they are not suitable, the decision to choose them has to be withdrawn (backtracking). One could say that new objects get a chance to prove their worth. But also well-tried objects can have a reduced relevance by no longer being useful—with ageing at the same time—e.g. as a result of market trends resulting in new reward values. This effect is also shown with the example of Section 5 (see Fig. 8).

RKF does not change whether a configuration is innovative or not. But if a configuration system allows an insertion of new components (innovations) during a configuration process, RKF will consider them adequately as new components (see above). Additionally the stochastical part in selections with RKF ensures that new (but similar) solutions can be found again and again for a given task.

In our experiments with RKF these results could be confirmed. One example domain of these experiments will be introduced in the next chapter. But the experiments have also shown limitations of the heuristic.

If an object has to be chosen from a set of n objects at least n runs are necessary to get rewards for all of the objects.

During these steps non optimal results are gained. One possibility to solve this problem is not to begin with a start relevance of 0.5 but with a relevance equal to a first representative user reward for each object (if such values are available, e.g. from an expert of the domain).

The determination of the parameters b_t and b_f of the relevance function is also a problem: For example if learning and forgetting is too fast (small b_t, large b_f) one object of each set of available objects becomes relevant so fast, that its probability to be chosen is near 100% before all objects could obtain at least one reward. For the experiments presented in the next chapter we have determined these values empirically.

Another problem is that RKF will only learn optimally if rewards for each component of the solutions are available. This might be a problem in complex systems where solutions consist of hundreds of components. A solution for this can be the use of simplified assessment schemes. One example is to use one rating value for the complete solution which is used as the reward of all components of the solution.

A fourth problem exists because RKF does not learn dependencies *explicitly*. With RKF only the objects which are to be used in context of a task class c are learnt. Dependencies are only learnt in so far as the set of *objects* which become relevant fulfil all dependencies.

The solution of these problems or the determination of cases, where RKF cannot be used, will be subject matter of our future research. However, even with the current version of RKF, good results could be obtained in our experiments.

We will now classify the learning portion of RKF into the 'dimensions of learning in design' defined by Grecu and Brown [16] as follows:

1. Triggers of learning: *success*. Whenever a solution has been found and an assessment of the solution is available time t is increased and the recursive reinforcement-learning function is used (Eq. (2)).
2. Elements supporting learning: *feedback after completing the design task*. The user's assessment ('reward') is used as the reinforcement signal.
3. What gets learned: *preferences and successful designs*. Objects of the solution which are desirable in the context of a task class get high rewards leading to the effect that non-explicit preferences are learned. Besides this only consistent solutions can get an assessment. Therefore consistent sets of components will be learned implicitly. They are not stored as cases but as relevance values which correspond to 'probabilities to be chosen' in a given task class.
4. Availability of knowledge for learning: *direct communication*. An assessment of a user is needed.
5. Methods of learning: *reinforcement-learning*.
6. Local versus global learning: *local*.
7. Consequences of Learning: *design improvements and improvement of the design process.* The designs can be improved because optimisation goals of a task-class are implicitly learned which leads to the result that only appropriate objects are chosen. The design process is improved, because less backtracking is necessary as a consequence of learning consistent sets of objects.

RKF is not completely classified by this scheme because forgetting is also crucial for our method. Forgetting is not recordable with this scheme.

5. Example for using configuration and RKF

We have developed the configurator prototype 'RKF-Configurator' to test the effects arising from the use of RKF. It implements knowledge based configuration for structure based technical domains. Supported modelling techniques for the configuration knowledge base include a taxonomical hierarchy with single inheritance, compositional hierarchies, $n:m$ relations and JAVA-based constraints (special user defined JAVA methods). During the configuration process depth-first search and chronological backtracking are used in a build and test mode. RKF is the only heuristic we use at this time. To generate CAD models the converter is able to combine VRML models (files) which are referenced by the single components of the knowledge base to one VRML model of the system. The converter contains an algorithm which is able to place and orient these models following the specifications of the four dependencies from Section 3.1 (just before Fig. 3). This VRML file generation has been used in the following example. We have also combined the RKF-Configurator with the CAD system AutoDesk Inventor by a converter which we have implemented in Java. It reads the solutions from the RKF configurator (as JAVA objects) and sends Visual Basic (macro language) commands to the CAD program by OLE to generate a CAD model. The imported components are not yet automatically placed in this case.

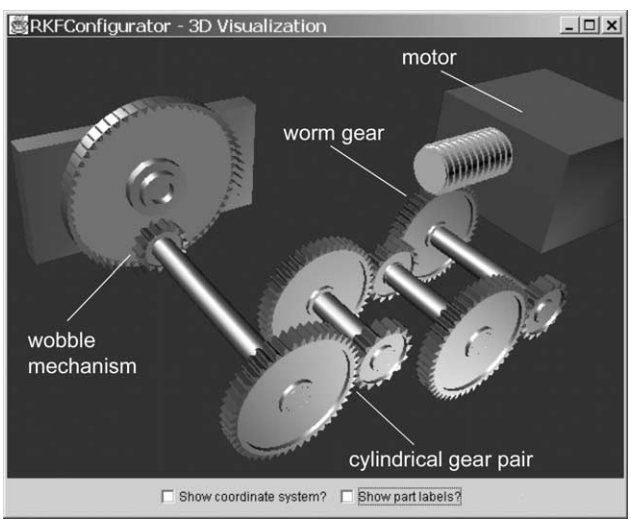

Fig. 5. VRML model of a motor and a gear in the backrest of a driver seat.

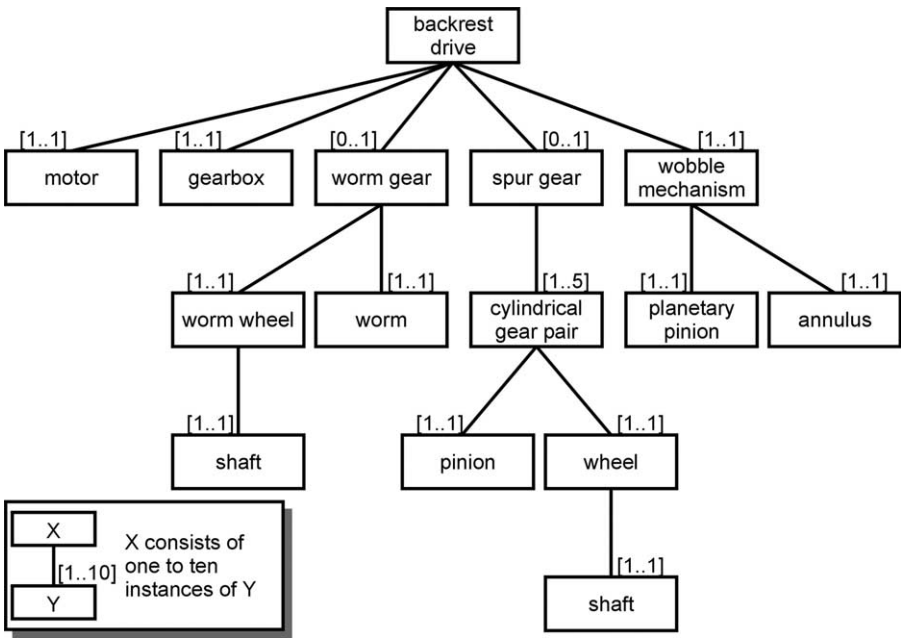

Fig. 6. Compositional hierarchy of the domain in our example.

More than 1100 experiments have been performed each consisting of 200 configuration runs in different domains with this prototype to test the effects arising from the heuristic RKF. One of these is presented hereinafter.

Our example is part of a mechanical engineering problem, which has been modelled with our prototypic implementation: a backrest drive needs to be configured using RKF from its individual components such as the motor, worm gear, pinions, shafts etc.

An example configuration of this domain is depicted in Fig. 5 which has been generated by the converter of our prototype. Fig. 6 shows the compositional hierarchy of this domain which has been modelled in the knowledge base of the configurator as frames (objects).

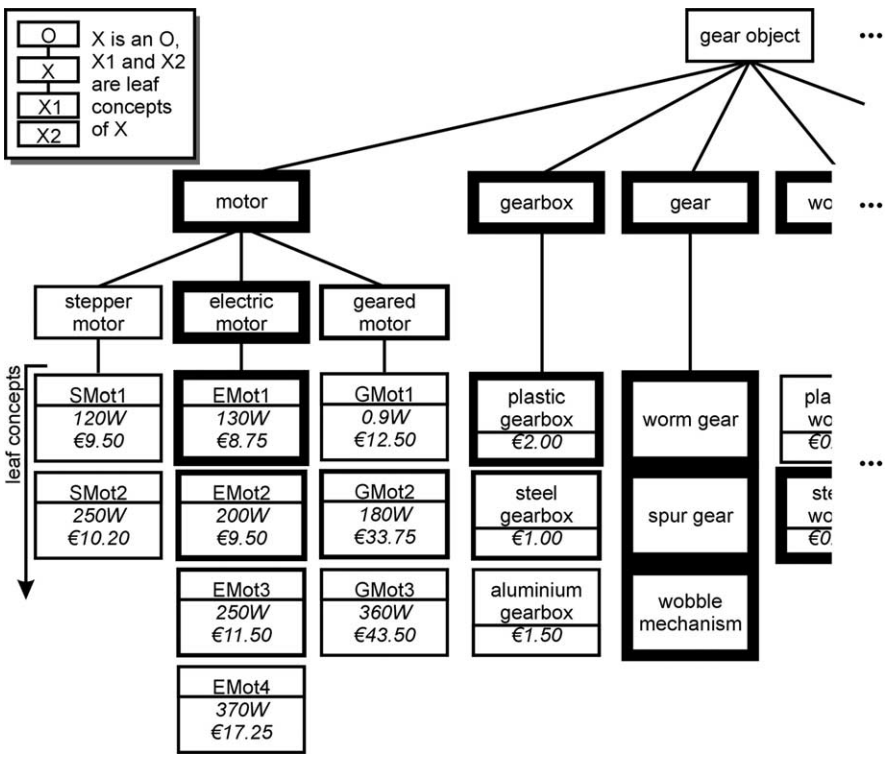

Fig. 7. Part of the taxonomy of the backrest drive domain.

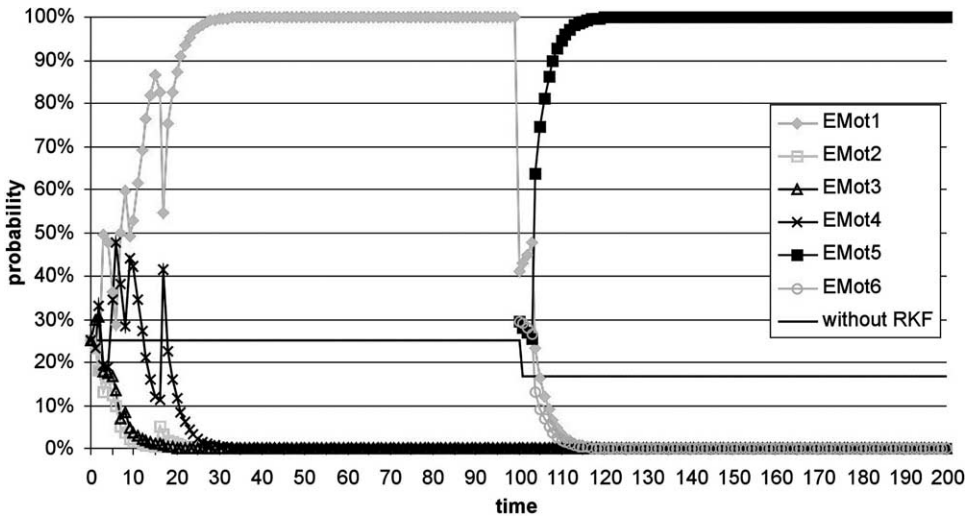

Fig. 8. Probability for choosing electric motors depending on their relevance for the task class 'standard driver seat'. At $t = 100$ EMot5 and EMot6 were added.

The ranges indicate how often a concept can be a part of another. For example the spur gear in the above example can consist of one to five cylindrical gear pair modules, which again consist of exactly one pinion and one wheel. A wheel always consists of the wheel itself and a shaft. For a task specification one of these concepts has to be chosen as the root of the wanted system. In our example this will normally be the concept 'backrest drive'. Examples of task classes in this domain are 'premium car driver seat', 'premium car passenger seat', 'standard driver seat', 'taxi driver seat' etc.

Fig. 7 shows part of the taxonomy of the example domain. The above concepts are ancestor nodes of those concepts connected with lines below. For better clarity the alternatives of leaf concepts in Fig. 7 have been represented in a list, one below the other (without further lines). The thickness of the boxes indicates the learnt relevance; i.e. for the task class 'Standard Driver Seat' this form of drive gets the highest rewards for the electric motor EMot1, a plastic gearbox, a steel worm and so on. The figure also shows how RKF supports a fast depth first search in this hierarchy: Paths of high relevance develop because whenever a daughter concept was useful (received a reward), the respective father concept was also useful. Whenever a specialisation of a concept has to be chosen during the configuration process the RKF heuristic will search along these relevance paths with high probability. In our example a motor has to be chosen, which is a component of 'backrest drive'. RKF will find a specialisation for motor in the context of 'Standard Driver Seat' task class by specialising it to 'electric motor' and finally to EMot 1 with high probability corresponding to their high relevance.

In other task classes of the same domain the relevance values can be completely different. For example the relevance of the stepper motors for the 'premium car driver seat' task class can be much higher than that of the electric motors. The configuration system will learn that these are used more often in that task class (without learning the underlying reason: the seat position memory requires this sort of motor). This example shows that dependencies can be learnt implicitly in the context of the task classes even if parts of the dependencies are not explicitly modelled.

Fig. 8 shows the probability course of EMot1 through EMot6 to be chosen in an experiment that has been performed in the context of the task class $c =$ 'Standard Driver Seat' in the backrest drive domain. At first ($t < 100$) only four electric motors were part of the set of available motors. The rewards given for these four motors were: $\text{reward}(\text{EMot1}, t, c) = 100\%$, $\text{reward}(\text{EMot2}, t, c) = 10\%$, $\text{reward}(EMot3, t, c) = 10\%$ and $\text{reward}(EMot4, t, c) = 50\%$ for $t < 100$ provided they were part of the solution. Consequently the configuration system should learn that EMot1 is the 'best' motor for this task class.

Due to the fact that they all have the same relevance of 0.5 (the starting relevance) in the beginning, the probability to be chosen is equally 1/4 for all motors at this time.

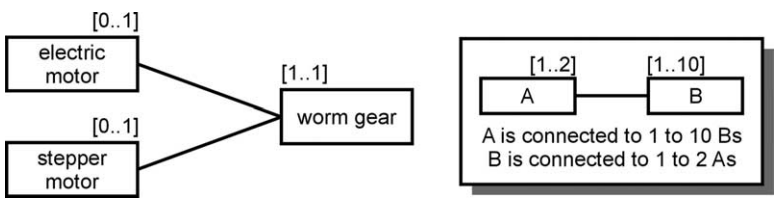

Fig. 9. Additional dependencies.

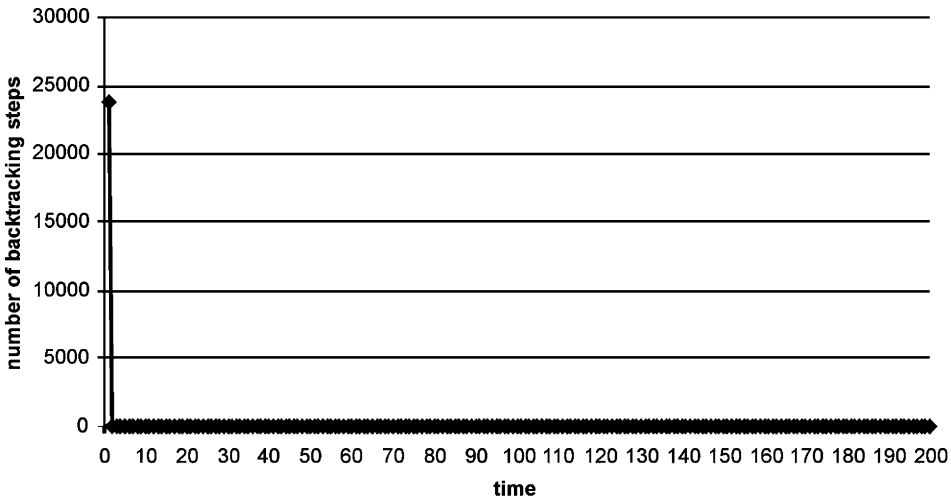

Fig. 10. Backtracking becomes quickly unnecessary.

During the first few configuration tasks EMot 3 was selected randomly but because it received no good rewards it became only slowly more relevant. As soon as EMot4 or EMot1 were chosen these became relevant more quickly because of good rewards. Finally EMot1 which received the maximum reward of 100% 'won', i.e. the configuration system has learnt that this is the best motor for this task class.

The flexibility of RKF appears as soon as new components are added. In the example of Fig. 8 two new electric motors (EMot5 and EMot6) were added to the taxonomy of the knowledge base after 100 configuration tasks ($t = 100$). They start again with a relevance value of 50% resulting in a probability value of about 30%. The probability of the so-far most relevant motor EMot1 falls to a value of about 40% although the relevance of EMot1 is unchanged just because another two motors were added which have a relevance value that is not near 0% (see Eq. (3)). At this time also the reward scheme was altered: EMot1 through EMot4 received low rewards (1%) from then on and EMot5 received the highest reward (100%). EMot6 also received low rewards (10%). RKF now learnt that EMot5 replaces EMot1 and EMot6 does not 'prove its worth': EMot1 was chosen four more times but because of the low reward it did not become more probable quickly. But as soon as EMot6 was chosen randomly this motor became relevant quickly because of its high rewards and as a consequence became more and more likely to be chosen again. From then on EMot1 and EMot6 were also no longer chosen and became irrelevant because of forgetting.

In the example domain two additional dependencies occur which are shown in Fig. 9. The $n : m$ relations are used to model the fact that the two motor types 'electric motor' and 'stepper motor' need a worm gear to make the backrest drive self blocking.

Our prototype implements a build and test mode which means that a solution is built using the compositional and taxonomical information and then all additional dependencies are checked. If a dependency is violated chronological backtracking is used to systematically scan the search space until a consistent solution can be found. The components of the solution get rewards and become more relevant and consequently more likely to be part of the next solution. Fig. 10 shows the number of backtracking steps that were necessary during the configuration runs of the experiment described above.

Backtracking was only necessary during the first two configuration runs. After then objects which are a valid part of the additional dependencies were already so relevant that they were chosen again and again which lead to no more conflicts. It must be said that this was an ideal run. In other experiments it lasted up to 60 configuration runs before backtracking was no longer needed.

6. Summary and conclusions

For a further enhancement of reuse we have suggested the use of a knowledge based configuration system to support the choice of components (features) and parameters. It has been discussed how a coupling of a configurator with a CAD system can be performed in several possible ways. In this context aspects of spatial configuration have also been discussed. To solve the problems that arise from changing sets of components and market trends we suggest the application of the heuristic RKF which uses reinforcement learning and forgetting.

The advantages of the proposed approach are: (1) the use of knowledge based configuration in connection with CAD systems simplifies the selection and

parameterisation of CAD components; (2) modelling techniques of knowledge based systems (e.g. frames, inheritance,…) help to support the organisation of components or features more adequately; (3) mechanisms of a knowledge based system concerning relations make it very simple to ensure dependencies between components or features in an actual CAD model; (4) due to the learning portion of RKF many selections can be performed without user interaction during configuration; (5) better solutions can be found faster because also optimisation goals are learned implicitly and backtracking is avoided; (6) the forgetting portion ensures that the heuristic can be retrained which enables the heuristic to follow continuously changing demands and changing sets of components.

However, the heuristic RKF is restricted to select elements from *discrete* sets of elements. As a consequence it does not help determining spatial parameters (position, orientation) of components. Therefore we had to use other mechanisms to solve the layout problem in our prototypic implementation: constraints have been used to model spatial dependencies. Other limitations include the fact that the relevance values of the objects should be initialised and the users have to provide assessments for the solutions. The necessary values might not be available especially in large domains. Another problem might be the determination of the constants b_f and b_t which control the speed of forgetting and learning. Work is currently being carried out to investigate further the prerequisites and limitations of RKF. In this context the use of RKF combined with other heuristics seems to be a promising approach. RKF will use the results of the 'other' heuristics as additional input as it will also learn that objects are relevant which were selected by other heuristics. For these investigations we plan to integrate the RKF heuristic into a commercial configuration kernel that implements several other heuristics and strategies.

References

[1] Roller D. Foundation of parametric modelling. In: Hoschek J, Dankwort W, editors. Parametric and variational design. Stuttgart: B.G. Teubner; 1994. p. 63–72.

[2] Chung JCH. Constraint-based variational design. In: Hoschek J, Dankwort W, editors. Parametric and variational design. Stuttgart: B.G. Teubner; 1994. p. 27–36.

[3] Klein R. The role of constraints in geometric modelling. In: Bruederlin B, Roller D, editors. Geometric constraint solving and applications. Berlin: Springer; 1998. p. 3–23.

[4] Brown D. Functional, behavioral and structural features. In: Borg JC, Farrugia P, editors. Proceedings of KIC-5: The fifth IFIP WG 5.2 Workshop on Knowledge Intensive CAD, University of Malta, Department of Manufacturing Engineering; 2002. p. 34–46.

[5] Roller D, Schaefer D. Variational Design in Electrical Engineering. An Extension of parametric modelling. In: Brunet P, Hoffmann CM, Roller D, editors. CAD tools and algorithms for product design. Berlin: Springer; 2000. p. 233–50.

[6] Kreuz I, Forchert T, Roller D. ICON, Intelligent configuring system. In: Roller D, editor. Proceedings of the 31th ISATA, Volume Automotive Electronics and New Products. Croydon, England: Duesseldorf Trade Fair; 1998. p. 219–26.

[7] Brown DC, Chandrasekaran B. An approach to expert systems for mechanical design in trends and applications. Gaithersburgh, MD: IEEE Computer Society, National Bureau of Standards; 1983. p. 173–80.

[8] Cagan J, Shimada K, Yin S. A survey of computational approaches to threedimensional layout problems. Comput Aid Des 2002;34(8):597–611.

[9] Mittal S. A knowledge-based framework for design. In: Kehler T, Rosenschein S, editors. Proceedings of the Fifth National Conference on Artificial Intelligence, Menlo Park, CA: The AAAI Press; 1986. p. 856–65.

[10] Guenter A, Kuehn C. Knowledge-based configuration—survey and future directions. In: Puppe F, editor. Knowledge-based systems: survey and future directions. Proceedings of XPS-99, Wuerzburg, Berlin: Springer; 1999. p. 47–66.

[11] Yan X-T, Rehmann F, Borg J. FORSEEing design solution consequences using design context information. In: Borg JC, Farrugia P, editors. Proceedings of KIC-5: The fifth IFIP WG 5.2 Workshop on Knowledge Intensive CAD, University of Malta, Department of Manufacturing Engineering; 2002. p. 18–33.

[12] Guenter A, Hotz L. Aufloesung von Konfigurationskonflikten mit wissensbasiertem Backtracking und Reparaturanweisungen. In: Guenter A, editor. Wissensbasiertes Konfigurieren. Ergebnisse aus dem Projekt PROKON. Sankt Augustin: Infix; 1995.

[13] Heiden TK, Rechnerunterstützte Auswahl, Konfiguration und Berechnung von Antriebselementen mit einem wissensbasierten CAD-System am Beispiel von drehstarren biegeelastischen Kupplungen. PhD diss August 1992. University of Berlin (TU), Fortschr.-Ber. VDI-Reihe 20 Nr. 70, VDI, Duesseldorf; 1992.

[14] Kopisch M, Guenter A. Configuration of a passenger aircraft cabin based on conceptual hierarchy, constraints and flexible control. In: Belli F, Radermacher FJ, editors. IEA/AIE, Paderborn. Berlin: Springer; 1992. p. 421–30.

[15] Rich E, Knight K. Artificial intellligence, 2nd ed. Singapore: McGraw-Hill; 1991.

[16] Grecu DL, Brown DC. In: Duffy AHB, Brown DC, Goel AK, editors. Dimensions of machine learning in design. AI EDAM (artificial intelligence for engineering design, analysis and manufacturing), vol. 12. Cambridge: Cambridge University Press; 1998. p. 117–21. Number 2, special issue on Machine learning in design.

[17] Kreuz I, Roller D. Knowledge growing old in reconfiguration context. In: Faltings B, Freuder EC, Friedrich G, Felfernig A, editors. Configuration, Papers from the AAAI Workshop. Technical Report WS-99-05, Orlando; 1999. p. 54–9.

[18] Kreuz I. Considering the dynamic in knowledge based configuration. In: Sauer J, Koehler J, editors. Proceedings of the ECAI Workshop on New Results in Planning, Scheduling and Design, PUK2000, Berlin, 2000.; 2000. p. 83–9.

[19] Gomes P, Bento C, Gago P. In: Duffy AHB, Brown DC, Goel AK, editors. Learning to verify design solutions from failure knowledge. AI EDAM (artificial intelligence for engineering design, analysis and manufacturing), vol. 12. Cambridge: Cambridge University Press; 1998. p. 107–15. Number 2, special issue on Machine Learning in Design.

[20] Anderson JR. Cognitive psychology and its implications, 5th ed. New York: Worth Publishers; 2000. Chapter 6 and 7.

[21] Kaelbling LP, Littman ML, Moore AW. Reinforcement learning: a survey. Journal of Artificial Intelligence Research 1996;4: 237–85.

[22] Waltz D. The importance of importance. AImagazine 1999;20(3): 18–35.

Dieter Roller is a member of the board of directors of the Institute of Computer Science at the University of Stuttgart. Furthermore he is the head of the Graphical Engineering Systems Department and full professor of computer science. Additionally he has been awarded the distinction as an honorary professor of the University of Kaiserslautern. Professor Roller holds an engineering diploma and has a PhD in computer science. As former research and development manager with world-wide responsibility for CAD-Technology within an international computer company, he has gathered a comprehensive industrial experience. Professor Roller is well-known through many presentations throughout the world as well as through over 150 publications on computer-aided product development. Furthermore he has been granted several patents. He is chairman of several national and international working groups and organiser of symposia, congresses and workshops. With his wealth of experience, he also serves as a technology consultant to various high-tech companies.

Ingo Kreuz studied Computer Science at the University of Stuttgart and graduated in 1996. In August 1997 he joined the Research and Technology department of DaimlerChrysler AG and is working on his doctoral thesis on Knowledge Based Configuration. He is responsible for the ICON project (ICON, intelligent configuration), which is a co-operation between DaimlerChrysler AG and the University of Stuttgart.