ICED13
INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED13
19-22 AUGUST 2013, SUNGKYUNKWAN UNIVERSITY, SEOUL, KOREA

A CONCEPT OF DIRECT KNOWLEDGE ACQUISITION FOR MULTI-AGENT DESIGN SYSTEMS

Martin KRATZER, Alexander CROSTACK, Reem KADADIHI, Michael RAUSCHER, Hansgeorg BINZ, Peter GOEHNER
University of Stuttgart, Germany

ABSTRACT
In this contribution, the paradigm of direct knowledge acquisition as a possibility to enable experts to integrate user-specific knowledge is applied to multi-agent design systems (MADS). A pure adoption of existing approaches is not sufficient due to the fact that MADS have a distributed knowledge base as opposed to centralised knowledge bases of other knowledge-based systems (e.g. case-based systems). The result is a concept which brings the intentions of design engineers by modifying the knowledge base (e.g. adding a rule) in accordance with the inherent processes in the knowledge integration component of MADS. Thus, an action of a design engineer is followed by certain operations which have been inherently carried out in order to realise the modification of the knowledge base and to keep the knowledge base consistent. In order to do so, an overview about the direct knowledge acquisition is outlined. Moreover, the structure of the distributed knowledge base is presented wherein the ProKon-knowledge forms are used. Finally, the functionality of the direct knowledge acquisition is presented in detail using the example of a key connection.

Keywords: multi-agent design systems, direct knowledge acquisition, knowledge management

Contact:
Martin Kratzer
University of Stuttgart
Institute for Engineering Design and Industrial Design (IKTD)
Stuttgart
70569
Germany
martin.kratzer@iktd.uni-stuttgart.de

1 INTRODUCTION

In knowledge engineering, knowledge acquisition represents one step of the development of knowledge-based systems in terms of elicitation, analysis and representation of knowledge (Stokes, 2001). Within the direct knowledge acquisition, as one form of the knowledge acquisition, knowledge integration components enable experts to integrate user-specific knowledge on their own at any time they want (Blythe et al., 2001). Depending on the case of application, design engineers are able to modify the properties of materials, integrate new formulas for designing a machine element, or just delete a rule. As a result, it is not necessary to wait for a knowledge engineer who is familiar with the details of software aspects of the knowledge-based system and its knowledge base (Blythe et al., 2001). Considering the aforementioned issues, Suraweera et al. (2010) conclude that knowledge-based systems are generally more expedient in industry when a knowledge integration component is used.

The aspect of supporting design engineers with knowledge integration components is part of the comparison of indirect knowledge acquisition and direct knowledge acquisition (see Figure 1). Both processes differ in terms of time required to run through the process (Gómez-Pérez et al., 2010). When following the process of indirect knowledge acquisition (see Figure 1, right path), coordination and certain decision-making steps are necessary to finally produce the same result as produced by the process of direct knowledge acquisition (see Figure 1, left path). Thus, by using direct knowledge acquisition, knowledge engineers are no longer needed for less complex activities (e.g. adding a rule) but rather only for larger modifications (e.g. integrating a new type of machine element) to the knowledge base (Maguitman, 2004).

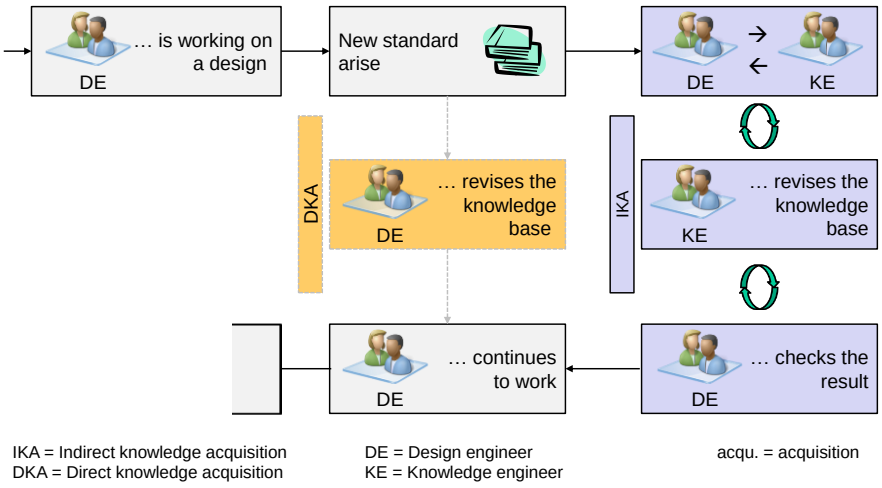


Figure 1. Comparison of direct and indirect knowledge acquisition

The approach of using direct knowledge acquisition by means of a knowledge integration component could be transferred to multi-agent design systems (MADS) as one form of knowledge-based system. The main difference between MADS and knowledge-based systems is that engineering design knowledge is stored in a distributed knowledge base as opposed to a single knowledge base as in common knowledge-based systems (e.g. rule-based systems). This means that the knowledge to be integrated has to be intelligently distributed to the correct agents in the correct form of knowledge.

In the ProKon project (Proactive support of engineering design processes with agent-based systems), such a MADS, called the ProKon system, is still being developed to support design engineers during the embodiment design phase. This system checks a digital product model in the CAD system in accordance with Design for X guidelines (DfX). Afterwards, a solution will be identified if the digital product model is inconsistent (Kratzer et al., 2011 and Rauscher et al., 2012). Based on this solution, the ProKon system eliminates inconsistencies by modifying the digital product model directly (Kratzer et al., 2011).

Combining the statements made by Suraweera et al., Gómez-Pérez et al. and Maguitman with experience gathered as part of the ProKon project, the ProKon system could be much more expedient in industrial environments if every design engineer has the possibility, depending on their personal rights, to modify the knowledge base and therefore the capabilities of the system at any time.

2 PROBLEM STATEMENT AND GOALS

The problem is that there is currently no concept for direct knowledge acquisition available for MADS and specifically for the ProKon system. It is not sufficient to merely apply existing concepts for direct knowledge acquisition to the field of MADS because of the aforementioned difference of the knowledge base. By just applying existing concepts, the knowledge could not be distributed to single agents correctly.

The research question is as follows: *"How should a direct knowledge acquisition being conceptualised concerning the distributed knowledge base, so that subsequently designers are able to easily integrate user-specific engineering design knowledge in a user-friendly way on their own?*"

In this context, inter alia, easily means that no additional software is needed for the modification of the knowledge base. All necessary functions are included in a single knowledge integration component. User-friendliness means that the interaction with the knowledge integration component is intuitive, so that, for example, no work-intensive trainings have to be carried out. The knowledge integration component and its programming will not be discussed in this contribution.

Essentially, the concept of direct knowledge acquisition for MADS is strongly determined by the theoretical possibilities of design engineers modifying the distributed knowledge base (e.g. adding a rule) as well as its structure. These theoretical possibilities, called application scenarios, must have an internal representation. In this approach, the internal representations are processes which are intentionally initiated by design engineers, for example when a rule has to be added. Additionally, certain mechanisms are necessary to ensure the consistency of the knowledge base after its modification.

In response to the research question, the hypothesis is formulated as follows: "*The concept of a direct knowledge acquisition for a MADS such as the ProKon system is carried out using application scenarios, their internal representations and certain mechanisms to ensure the consistency.*" The result is a description of the concept for a direct knowledge acquisition, which is shown through the explanation of the application scenarios and their internal representations. At the end, the results are used for the realisation of the knowledge integration component.

This contribution is directed at knowledge engineers who are involved in the development of a MADS. These engineers are located in the conceptual phase of the development of a MADS as a pre-stage to the actual programming phase, which is carried out by software engineers.

3 STRUCTURE OF THE CONTRIBUTION

Based on this problem statement, Section 4 presents the state of the art in terms of direct knowledge acquisition and existing knowledge integration components. Moreover, it illustrates the originality of the approach used in this contribution. Sections 5 and 6 have a prescriptive nature to give knowledge engineers implicit guidelines for conceptualising a direct knowledge acquisition for a MADS. Section 5 deals with the fundamental idea of how to modify the knowledge base of a MADS. Building on this, Section 6, as the main content of the contribution, starts with a description of the structure of the knowledge base (see Section 6.1). In Section 6.2, application scenarios and their internal representations, called operations, are described, followed by an example (see Section 6.3). Section 6.4 presents mechanisms regarding consistency checks. The obtained results are discussed in Section 7. Furthermore the upcoming evaluation is described. The contribution ends with a general conclusion and an outlook (see Section 8).

4 STATE OF THE ART

In the following, a selection of general approaches and existing knowledge integration components will be ordered thematically. As a common foundation, it can be stated that all authors are concerned with direct knowledge acquisition as the direct transferral of knowledge from an expert (e.g. design engineer) to the knowledge-based system (i.e. they all deal with direct knowledge acquisition).

Based on a survey conducted among experts from different fields, Olson et al. (1995) have developed a shell as a development environment for the user-friendly development of knowledge-based systems. This project exceeds the common development of a knowledge integration component due to the fact that this is the highest level of abstraction because putative users actually have all the possibilities of creating the knowledge-based system as they want. Overall, the fundamental functionality was demonstrated.

Blythe et al. (2001) deal with a knowledge integration component on the assumption that the graphical user interface (and its conception and realisation) is the most important aspect. Consequently, it is the task of this interface to manage complex dialogues between experts and the knowledge integration component. Furthermore, they have identified personal aspects which answer, inter alia, the following questions: How should experts start to integrate knowledge? How can one navigate through the GUI of the component? How can experts be informed of when they are producing an inconsistency? Furthermore, Blythe et al. conclude that knowledge engineers, who have used a standard knowledge engineering methodology to develop such a component, fail to satisfy the requirements of the users. This might be in contradiction to what Olson et al. describe with the use of a shell as a development environment.

Rafea et al. (2003) have developed a knowledge integration component for a knowledge-based system to support the organisation of irrigation and fertilisation processes in agriculture. The task of this component is to support experts in integrating new knowledge, in deleting knowledge and in modifying the knowledge base, for example by changing knowledge elements. The knowledge base consists of a strong network of knowledge elements that support the maintenance of consistency (Rafea et al., 2003).

Furthermore, Chaudhri et al. (2007) pursue a project in which knowledge from one standard work in the areas of physics, chemistry and biology each has to be implemented within one knowledge-based system. In their work, they examine "the blank slate problem", which deals with the problem that experts do not know where to start when they want to integrate knowledge. According to this, experts do not know how to align new knowledge with existing knowledge within the knowledge base (Chaudhri et al., 2007).

Generally speaking, there have already been a couple of attempts to apply direct knowledge acquisition and to develop knowledge integration components respectively. They are mostly academic solutions with no application in industry. This aspect is strongly related to the argument put forward by developers of knowledge integration components that it is not possible to develop such a component without training experts in advance. This generally prevents decision makers in industry from adopting such systems. Although all projects are about direct knowledge acquisition and the realisation of subsequent software components, they are referring to common knowledge-based systems with a single knowledge base. A way to directly integrate knowledge by users into distributed knowledge bases of MADS is not described.

5 CONCEPT OF DIRECT KNOWLEDGE ACQUISITION FOR MADS

Because of the fact that a direct knowledge acquisition for distributed knowledge bases of MADS does still not exist, first, a concept has to be developed. This concept is based on two principles. First, there is the principle of a case-based cycle, which is inspired by case-based reasoning (CBR), as a conceptual description of integrating knowledge into a knowledge base (see Figure 2). This cased-based cycle consists of the four characteristic steps retrieve, reuse, revise and retain inter alia described by Mantaras et al. (2006).

At the beginning of the direct knowledge acquisition following the case-based cycle, design engineers first want to have an overview of the already integrated knowledge before they actually modify it. Therefore, they retrieve existing knowledge forms (in the ProKon project this abstract form is called the ProKon knowledge form, PKF) by using a query (see Figure 2, Task 1). Based on this query, a knowledge landscape has to be built up in order to offer a sectioned overview of the knowledge domain to the design engineer to avoid the blank slate problem (Task 2). For example, if the design engineer wants to implement a new rule for Design for Manufacturing regarding shafts, the surrounding knowledge landscape will be prepared. After this retrieval, design engineers either modify an existing PKF or they insert a new PKF within the existing knowledge domain (both correspond to the reuse step, Task 3). Following this reuse, the changes in terms of general consistency and insularity (not completeness) will be revised (see Task 4). Subsequently the design engineer will be instructed about what else has to be changed in case of an inconsistency (see Task 5).

The second principle is called initialising. This describes the distribution of the knowledge from the modified knowledge base over the existing agents in the MADS (see Task 6). There are certain rules for regulating in detail which knowledge is assigned to which agent. This aspect is not the concern of this contribution.

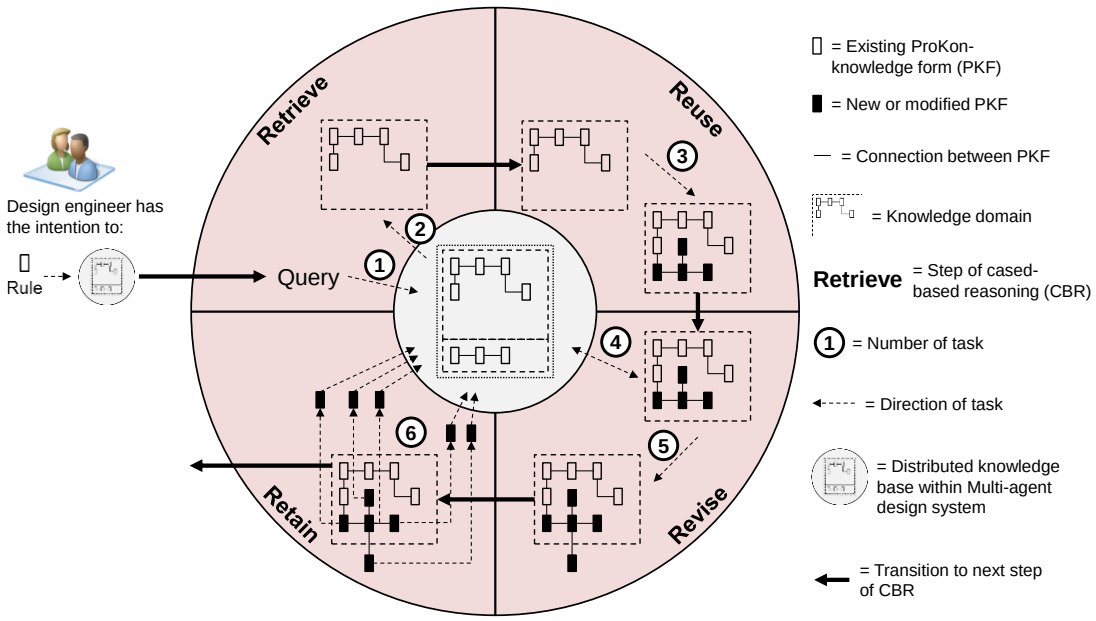


Figure 2. Overview of the direct knowledge acquisition

Tasks 1 and 2 comprise the functions of defining queries and preparing the knowledge landscape. Task 6 describes the distribution of the consistent knowledge over the agents. The realisation of these three steps is mostly a matter of software engineering. In contrast, Tasks 3 to 5 contain the modification and checking of the knowledge base as the actual content of the following sections. First, the structure of the knowledge base as the central aspect will be described (see Figure 2).

6 DETAILED CONCEPT

Based on the continuous further development of standards as a common situation in the engineering design process, the knowledge base of the ProKon system has to be updated by design engineers. This update represents the intention of the design engineer whereby each of these modifications is called application scenario. The realisation of this application scenario requires a number of additional process steps, called operations, to implement the modification as well as to ensure the consistency of the knowledge base. An application scenario or just a part of such an application scenario is for example the updating of a single formula. The inner representation (i.e. the operations) of this part, inter alia, comprises the replacement of mathematical operators, variables or constants.

Because of the importance of the distributed knowledge base, its structure will be discussed in Section 6.1. Both the application scenarios and their internal representation, the operations, are described in detail in Section 6.2. The necessary logical sequence of the different operations is shown in Section 6.3 using the example of the addition of a condition.

6.1 Structure of the knowledge base

The structuring of acquired knowledge and the interaction with design engineers during acquisition have to be supported in a standardised manner. For these purposes, ProKon-knowledge forms (PKF) were developed within the ProKon project based on the well-known ICARE forms developed by Stokes (2001). By means of the six developed types of PKF the whole knowledge domain concerning a machine element can be conceptually described. These PKF are rules, formulas, entities, parameters, tables and conditions (Kratzer et al., 2012).

Because DfX guidelines are generally based on rules, the rule form is one of the central forms. Rules can thereby be separated into main and operational rules (see Figure 3). Main rules comprise all necessary rules (operational rules) which have to be fulfilled with regard to a specific DfX guideline (e.g. Design for Manufacturing, DfM). Each of these operational rules contains one or more conditions which are stored in a separate PKF due to modularity reasons. The parameters which are used, amongst others in the conditions, are stored in the parameter PKF. This PKF is the central list of all used parameters, thus it defines parameters on a system-wide level. It contains the definition and the information on whether it is a constant or a variable parameter. The table PKF includes constants and assigns them to concrete values and their units. Necessary variables can be calculated using formulas.

Within the entity PKF, a knowledge domain is described in an ontological way. The parameters, classes, functions etc. are thereby assigned to each other using standardised relationships.
The relations between the PKF may be many-to-many (m:n) or one-to-many (1:n) relationships. For example, the main rule contains one or more operational rules, which are possibly part of different main rules. Although a table PKF contains more than one parameter, each constant is assigned to only one table PKF.

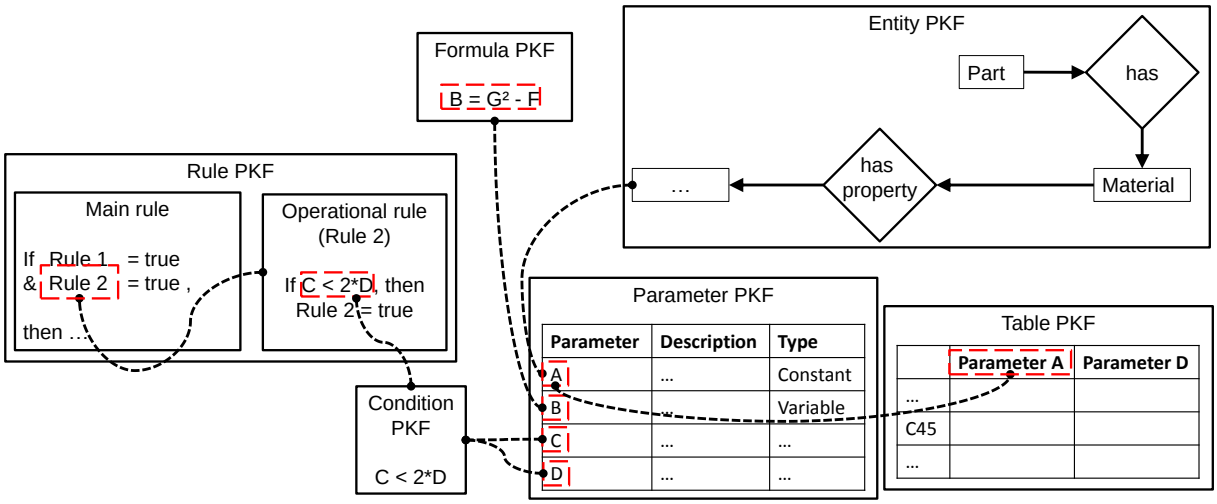


Figure 3. Relationships between ProKon knowledge forms (PKF) in the knowledge base

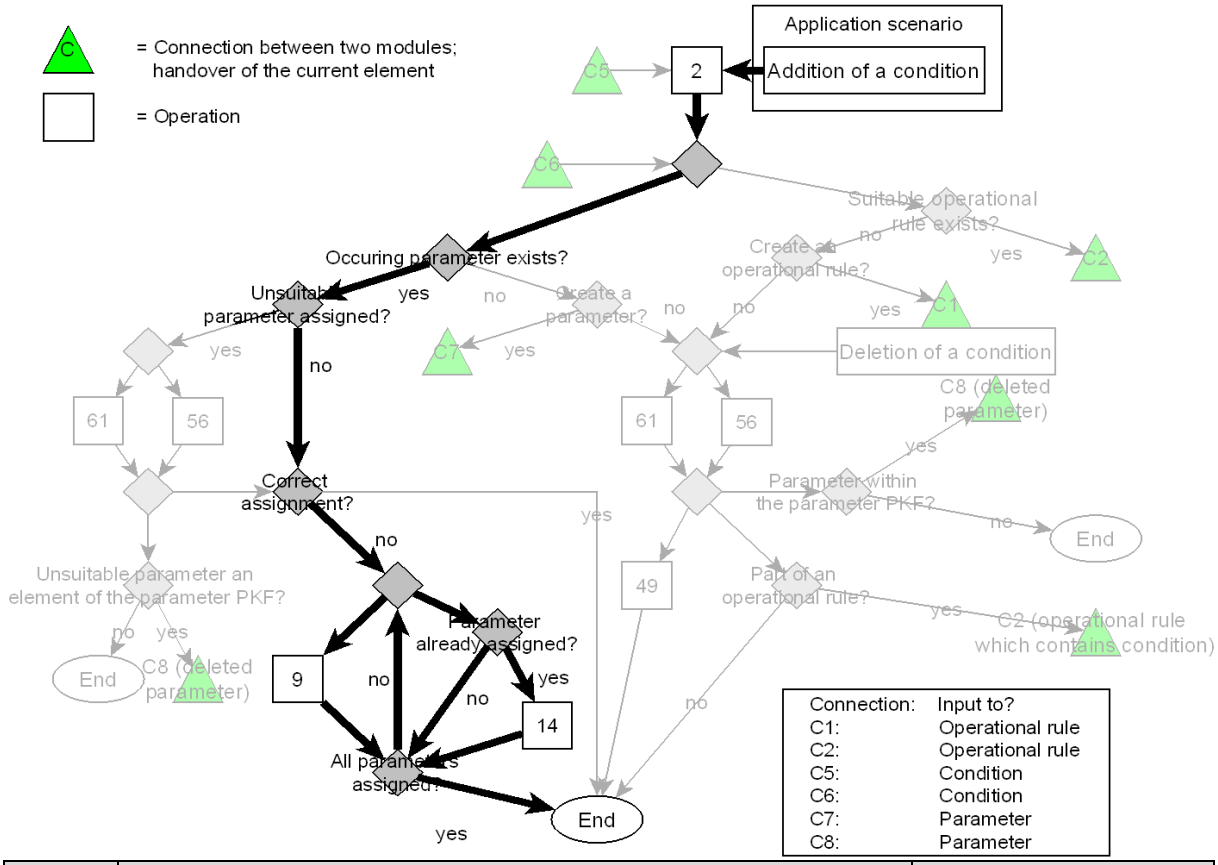
In order to describe the content of each PKF, 12 elements were identified, whereas a PKF is just a framework for gathering these elements. These elements are constants, variables, conditions, formulas, mathematical operators, Boolean operators, operational rules, main rules, tables, classes, instances and values. For example a formula consists of constants, variables and mathematical operators.

6.2 Overview of the application scenarios and operations

After describing the structure of the knowledge base by using different PKF, the question arises as to which of the PKF have to be handled during the modification of the knowledge base (see Figure 2, tasks 3-5). As already mentioned, every intention to modify the knowledge base is called application scenario followed by logical sequences of necessary operations to implement the modification as well as to ensure the consistency of the knowledge base. First the possible application scenarios and operations will be presented. Second it will be verified whether each of these application scenarios and operations is useful.
By using the three intentions addition, modification and deletion, which can be applied to each of the six types of PKF, all possible application scenarios can be identified. In the case of addition and deletion, a distinction can be made on the one hand between the creation of an element itself and its assignment to another PKF. On the other hand it has to be distinguished between the deletion of the element itself and the deletion of its assignment. For example, a parameter will be created in the parameter PKF and then assigned for its usage to a condition PKF. If the parameter will be deleted in its parameter PKF, all of its assignments have to be deleted too (cf. Figure 3).
The modification of a PKF comprises several different kinds. For example, modification can be the restructuring, transformation, renaming or replacement of an element. An example for such a modification is the transformation of a variable into a constant or vice versa. Another example is the replacement of a mathematical operator between the parameters in a condition (see Figure 3, Condition PKF "<").
The theoretically possible number of different operations can be determined by the number of PKF, the intentions and the elements occurring within the PKF. The question is whether all of these operations can occur. For example, a parameter always has to be renamed in the parameter form, in the condition form and in the entity form. Depending on if it is a constant or variable, the parameter also has to be renamed in the formula or the table PKF. Otherwise, a Boolean operator cannot be renamed. It only can be replaced. After a close examination of all theoretically possible operations, only 64 useful operations may actually occur within the reuse and revision steps.

6.3 Logical sequences of operations

After describing the application scenarios and operations in detail, the logical sequence of the latter, to fulfil the intentions, can now be presented. It can be stated that not every of the 64 useful operations always occurs in consequence of each application scenario, so that different logical sequences of operations can occur. Such a logical sequence can be developed using the "brainstorming" method. When using this method, the following questions have to be answered: Which elements might be affected by the application scenario? Which operations are necessary? Which PKF should also be examined? The results of these questions will be explained using the example of the application scenario "Addition of a condition" (Figure 1, highlighted path). Thereby, Figure 1 is a generic representation including all identified and different logical sequences of operations. Following a concrete example, not all of the paths have to be considered. Within Figure 1 squares represent operations. These are described in the table below the figure. Triangles represent connections to further operations which influence additional PKF, for example, the creation of a condition makes it necessary to assign the condition to an existing operational rule or to a new operational rule which has still to be created. The flow charts belonging to them are not depicted here.



No.	Description	Comment
OP 2	Creation of an empty condition PKF	Not the condition itself
OP 9	Assignment of a parameter to a condition PKF	
OP 14	Assignment of a mathematical operator to a condition PKF	Only used with OP 9
OP 49	Deletion of a condition within the condition PKF	
OP 56	Deletion of the assignment of a parameter to a condition PKF	
OP 61	Deletion of the assignment of a mathematical operator to a condition PKF	Only used with OP 56

Figure 1. Flow chart and table depicting operations of an exemplary application scenario

The starting point in this example is the addition of a condition (e.g. in case of a key connection following DIN 6892 (DIN, 2012) the rated key length $l_c \leq 1.3$ * shaft diameter d) and the first operation (OP 2), which includes the creation of an empty condition PKF. After its creation, the condition itself has to be defined using the mathematical operators ≤ and * and the parameters l_c and d. For the definition of the condition itself, at first, it has to be checked, if all parameters exist. In this

example it will be assumed that l_c and d are already parts of the parameter form, thus no more parameters have to be created. In the next step, the condition has to be checked whether unsuitable parameters are used with regard to the content. At next, the condition is checked whether all necessary parameters have been assigned correctly. These two steps only become necessary if an existing condition will be modified, because a new condition does not contain a parameter, thus, there are no unsuitable ones and not all necessary parameters are assigned to the condition.
Subsequently, the condition will be defined. For this, the required parameters are assigned to the condition by using OP 9 and the required mathematical operators are assigned by OP 14. For defining the condition of the used example, the first parameter l_c is assigned to the condition. Because there is no further assigned parameter, no mathematical operator is needed so far. Since the condition is not complete, the parameter d as well as the constant 1.3 will be assigned to the condition too. Additionally the mathematical operators ≤ and * will be assigned.
As soon as all necessary parameters and mathematical operators are assigned to the condition, the condition itself is complete, but examination is required in order to clarify whether the condition is used or should be used within one or more operational rules. These rules may also have to be created or in case of existing ones they have to be modified. This creation or modification could be also an application scenario, if it is the original intention of the design engineer. In this case it is only an operation. This illustrates the complex nesting of the process steps into one another.
In addition to this nesting, recursions may also occur. Following the example above, recursion means that the addition of another condition becomes necessary during the creation of the operational rule. Based on the structure of the knowledge base, nesting and recursion of the application scenario may occur more than once, thus a simple modification of the knowledge base may result in a major effort for the purposes of ensuring its consistency.

6.4 Derivation of mechanisms concerning consistency checks

Bearing in mind Figure 4, the complexity increases the probability that inconsistencies may appear in the knowledge base. Some aspects for avoiding inconsistencies are already included in the flow chart, for example, additional mathematical operators are only required if more than one parameter is used. Moreover, there are more inconsistencies which may appear by running through the logical sequences of operations. Thus, in this section, these possible inconsistencies will be discussed and the actions for their avoidance are illustrated. After analysing all possible application scenarios, considering the operations that can take place during the process of editing the knowledge base, three main types of inconsistency could be identified:

- **Type 1 – Incompleteness of PKF:**
  A PKF does not contain all necessary elements in accordance with the generic structure (e.g. a condition within a rule is missing). The first type can be directly detected in the user interface depending only on the user input and without checking the knowledge base. To avoid this type of inconsistency, a suitable functionality of the user interface is needed.
- **Type 2 - Redundancies of PKF or elements in PKF concerning the naming of them:**
  In this type, detection of the inconsistency just by means of the user interface is not possible. The possible inconsistency will be detected after the knowledge base has been checked. This check will validate the knowledge base and aims at identifying different inconsistencies, for instance having a parameter which is calculated by more than one formula.
- **Type 3 – Wrong content in PKF:**
  This type describes PKF with wrong content considering standards, books and knowledge of experts (e.g. F = m * v instead of F = m * a considering Newton's law). For this type, the inconsistency is caused by wrong content in the user input. Detection and avoidance by the knowledge integration component will not be possible because the component cannot identify the inconsistency on its own.

In the following, each type will be explained with an example to give a clear idea of the differences. To avoid inconsistency of the first type, different mechanisms in the user interface have to be realised, for example an enforcement concept which requires the usage of one or more input field(s) that must be filled out by the design engineer. Based on the already discussed example of the addition of a condition the design engineer wants to select one parameter and a mathematical operator. The storing of this PKF is prohibited as long as the user selects a second parameter. Although the aforementioned obligation in terms of using mathematical operators during the definition of a condition is an example

of type 1, it must also be checked whether a condition consists of a minimum of two parameters. Another possible example of the first type is the case of assigning a formula to a parameter which is already defined as a constant and should not be calculated. Such cases can be avoided by providing the design engineer with a limited choice of all available parameters that are not saved in the system as constants and thus can be calculated.
An example of the second identified type of inconsistency is the creation of a parameter with a name already used. This would lead to redundancy in the knowledge base and can be detected when performing the redundancy check (searching in the knowledge base to identify elements with the same type and the same name) before saving the new element. To avoid such inconsistency, the design engineer will be informed and will be asked to select a new name.
The last type, which cannot be detected by the system itself, is related to the content of the edited information. For example, during the addition of a new formula by the design engineer, a wrong mathematical operator is used. The system cannot find such inconsistency because it does not know how the correct formula is defined. A second example of the third type is a rule in which the assigned conditions are mutually excluded. Detection of this logical inconsistency demands a great deal of effort because it requires a lot of text reorganisation processes and this is not in the scope of this concept.

7 DISCUSSION OF THE RESULTS

In this section, the results presented in Section 5 and 6 are critically discussed. Moreover, some implications are presented in order to name further application fields. First, it must be discussed whether the research question is rigorously answered by the hypothesis. An unambiguous answer in terms of the functionality is not yet possible because the real functionality of the concept must be shown in an evaluation of a prototype. It will only be possible to state that the concept was correctly developed after a successful evaluation. In addition, besides the functionality in terms of usefulness, applicability of the final knowledge integration component could also not yet be proven. Apart from that, rather rudimentary evaluation criteria such as consistency and insularity can more likely be proven but in a subjective manner. Considering the knowledge base (see Section 6.1), consistency means that the connections between the PKF are logical. For example, main rules may always be connected with operational rules. Insularity is just a dilution of completeness due to the fact that completeness cannot be proven at all. Insularity is achieved if all PKF are connected, so that there are no stand-alone PKF. By analysing a first working prototype of the ProKon system which is using the same knowledge base as the knowledge integration component of course, consistency and insularity considering the knowledge base could be confirmed. The flow chart (see Section 6.3), with their logical sequences of operations, are also evaluated in terms of consistency and insularity. A first run through the flow chart showed that an exact distinction of the Tasks (Tasks 3 to 5, cf. Figure 2) is not always possible. The tasks appear in micro cycles, for example, after the creation of the empty condition by OP 2 as a part of Task 3, initial consistency checks (e.g. “Unsuitable parameters exist?” as part of Task 4) are performed before the condition itself will be finally defined (final part of Task 3). Unfortunately, at this time, there is no evidence that the realisation of the flow chart within a knowledge integration component leads to a user-friendly and easy integration of user-specific knowledge. As an implication for research and practice, the presented flow chart is applicable not only for MADS but also for most knowledge-based systems. The overall strategy of formulating sequences for modifying the knowledge base is the same, as well as the possibility to jump through the flow chart from one point to a completely different point by means of connections. The structure of the knowledge base is not significant, thus independent from the PKF systematic. Complex networks with a high grade of crosslinking could be built up and presented more easily by means of reusable operations and connections.

8 CONCLUSION AND OUTLOOK

In this contribution, a concept of direct knowledge acquisition with a MADS is presented. In contrast to indirect knowledge acquisition where knowledge engineers have to integrate knowledge themselves, direct knowledge acquisition enables design engineers to integrate user-specific engineering design knowledge in a user-friendly way. Therefore, the fundamental concept of the direct knowledge acquisition is presented with the description of the case-based cycle. In addition to that, the structure of the knowledge base, consisting of ProKon knowledge forms (PKF), is described. The fundamental

concept of the direct knowledge acquisition as well as the description of the knowledge base leads to the presentation of the concept which is based on application scenarios and operations. Operations denote the inner representation of the design engineers' intention (application scenario) of modifying the knowledge base (e.g. adding a rule). Both are covered by the flow charts. One of eight flow charts is presented in this contribution, relating to the example of adding a condition as a part of an operational rule. Furthermore, three mechanisms concerning consistency checks are derived and show how to keep the knowledge base consistent when design engineers want to modify the knowledge base. Due to the fact that Figure 4 shows the flow chart for only one application scenario, the development of the flow charts has to be completed and the flow charts have to be finalised in terms of internal completeness and insularity. A follow-on step is the actual realisation and evaluation of the results (set of operations, set of application scenarios, flow charts and mechanisms concerning consistency checks) within a software prototype. It is likely that there will be iteration loops between the actual realisation in software and the development of the concept because there are no empirical values in the state of art to support this process.

ACKNOWLEDGMENTS

The authors would like to thank the Deutsche Forschungsgemeinschaft (DFG) for its support in the research project ProKon.

REFERENCES

Blythe, J., Jihie, K., Ramachandran, S. and Gil, Y. (2001) An integrated environment for knowledge acquisition. In *Proceedings of the 6th international conference on intelligent user interfaces*, Santa Fe, New Mexico, January 2001, pp. 13-20.

Chaudhri, V., John, B., Mishra, S., Pacheco, J., Porter, B. and Spaulding, A. (2007) Enabling Experts to Build Knowledge Bases from Science Textbooks. In *Proceedings of the 4th international conference on knowledge capture*, Whistler, BC, Canada, 28-31 October 2007, pp. 159-166.

DIN (Deutsche Institut für Normung e. V.) (2012) *DIN 6892 Drive type fastenings without taper action - Parallel keys - Calculation and design*, Berlin, Beuth Verlag GmbH.

Gómez-Pérez, J., Erdmann, M., Greaves, M., Corcho, O. and Benjamins, R. (2010) A framework and computer system for knowledge-level acquisition, representation, and reasoning with process knowledge. *International Journal of Human-Computer Studies*, Vol. 68, No. 10, pp. 641-668.

Kratzer, M., Rauscher, M., Binz, H., Goehner, P. (2011) An agent-based system for supporting design engineers in the embodiment design phase. In Culley, S. J., Hicks, B. J., McAloone, T. C., Howard, T. J., Chen, W. (eds) *Proceedings of the 18th International Conference on Engineering Design (ICED 11). Impacting Society through Engineering Design*, Design Society, Glasgow, pp. 178-189.

Kratzer, M., Crostack, A., Binz, H. and Roth, D. (2012) Distribution of engineering design knowledge within the development of multi-agent design systems. In Marjanovic D., Storga M., Pavkovic N., Bojcetic N. (eds) *Proceedings of the 12th International Design Conference DESIGN 2012*, pp. 1495-1506.

Mantaras, R. de, McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M., Cox, M., Forbus, K., Keane, M., Aamodt, A. and Watson, I. (2006) Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, Vol. 20, No. 3, pp. 215-240.

Maguitman, A. (2004) *Intelligent support for knowledge capture and construction*. PhD Thesis, Indiana University, Department of Computer Science.

Olson, D., Mechtov, A. and Moshkovich, H. (1995) The Role of Rules and Examples in the Process of Knowledge Acquisition in Direct Classification Tasks. *Expert Systems with Applications*, Vol. 8, No. 1, pp. 203-212.

Rauscher, M. and Göhner, P., (2012) Automated Consistency Check in early Mechatronic Design. In Wikander, J., Hehenberg, P., Flick, A. (eds) *Proceedings of the 13th Mechatronics Forum International Conference*, pp. 800-803.

Stokes, M. (2001) *Managing engineering knowledge*. *MOKA: methodology for knowledge based engineering applications*, London, Professional Engineering Publishing.

Suraweera, P., Mitrovic, A. and Martin, B. (2010) Widening the knowledge acquisition bottleneck for constraint-based tutors. *International Journal of Artificial Intelligence in Education*, Vol. 20, No. 2, pp. 137-173.