Moving Integrated Product Development to Service Clouds in the Global Economy
J. Cha et al. (Eds.)

doi:10.3233/978-1-61499-440-4-192

Knowledge Object - a Concept for Task Modelling Supporting Design Automation

Fredrik ELGH[a,1], Joel JOHANSSON[a]
[a] *School of Engineering, Jönköping University, Sweden*

Abstract. The ability to design and manufacture highly customer adapted products brings a competitive edge to manufacturing companies acting on a business-to-business market as suppliers to OEMs. A vital means for success in quotation and order preparation is advanced system support in design, process planning and cost estimation based upon the automation of engineering tasks. A design automation system encapsulates these tasks which are to be executed for specific customer specifications in a sequence specified either by a predefined order or resolved by an inference mechanism in run-time. Commonly, the development of a design automation system is an iterative process alternating between a top-down and a bottom-up approaches. An overall strategy is a necessity for successful system development, however, to successfully define the tasks, retrace all the necessary knowledge and to close gaps in both the tasks and the knowledge definitions require a complete and detailed understanding of the specific domain. In this paper, the concept of Knowledge Object is described together with examples of its use in both the development and system realization of design automation systems enabling product customization. The concept has shown to be useful for modelling of design processes, tasks, and engineering knowledge as well as in system development and realization. It also supports traceability and understanding by relations to other concepts describing associated requirements and design rational.

Keywords. Customized Products, Knowledge Object, Design Modelling

Introduction

The research in the field of design automation has mainly adopted an artefact oriented approach supported by the evolution in CAD software, i.e. the rules have been defined and organized in accordance to a product structure. This has been further supported by the different commercial KBE tools available today for modelling of design knowledge (e.g. CATIA KWA and Siemens PLM NX Knowledge Fusion). The process approach, on the other hand, has gained more success in the area of computing, where engineering tasks defined in different applications are connected for the purpose of simulation and optimization (e.g. ModeFrontier and Simulia Isigth). Two specific areas that have been subject for research are the development process of systems for customization of products and the modelling of product related information and knowledge supporting system realization. Hvam et al [1] describes a complete and detailed methodology for constructing configuration systems in industrial and service oriented companies. An iterative process is suggested including: analysis of product portfolio, object-oriented modelling, object-oriented design and programming. Every activity results in a description of the problem domain with different levels of abstraction and formalization. Two strategies are proposed for system documentation, either by using a product variant master and associated CRC (Class Relationship

[1] Corresponding Author.Fredrik Elgh, School of Engineering, Jönköping University, P.O.Box 1026, 551 11 Jönköping, Sweden; e-mail: Fredrik.elgh@jth.hj.se.

Collaboration) cards or by using the class diagram of a formal model and associated CRC-cards. The original content and structure of the CRC-cards have been further developed by Haug and Hvam [2]. Haug et al [3] have developed a prototype system for the documentation of the CRC-cards, the product variant master and the class diagrams. A procedure for development of design automation systems has been outlined by Rask [4] where issues about documentation and maintenance are addressed by emphasizing the need and importance of routines regarding versioning, verification and traceability. A possible means to support the updating of the knowledge-base proposed by Rask et al [5] is to strive for a design automation system implementation that allows the revision and the documentation to be executed at system runtime. Stokes [6] describes a methodology for the development of knowledge based engineering applications, MOKA, Methodology and software tools Oriented to Knowledge Based Engineering Applications. Two central parts of the methodology are the Informal and Formal models. The Informal model is used to document and structure knowledge elicited from experts, handbooks, protocols, literature etc. The Formal model is derived from the Informal model with the purpose to support system specification and programming.

It can be concluded that there is a lack of a comprehensive and detailed methodology for design automation development and realization supporting a process approach. The need of a methodology has been identified in projects executed in close collaboration with industry and the objective of this work is to bring together, structure and further expand actions and experiences in that direction with the purpose of building a methodology. The starting-point in industrial problems follows problem-based research as described by Blessing's research methodology for the development of design support [7]. The system development method [8] has been deployed as research methodology for the purpose to explore the research issue including the introduction, evaluation, and refinement of new concepts which, in turn, are perceived as prescriptive models in accordance with the design modelling approach [9].

1. Knowledge Object

The concept of Knowledge Object is at the core of the methodology described in this work. The concept was initially introduced by Elgh and Cederfeldt [10,11], Fig 1.

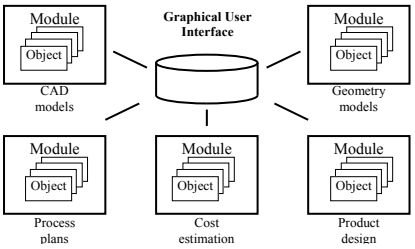

Figure 1. A system architecture based on Knowledge Objects introduced by Elgh and Cederfeldt [10,11].

They described a system for automated design, process planning and cost estimation of bulkhead in a submarine escape section. The knowledge enabling the automation of these activities was captured in Knowledge Objects grouped in modules. The concept was later adopted by Johansson [12] in the development of a system for automated design of toolsets for the rotary draw bending of aluminium tubes, Fig 2.

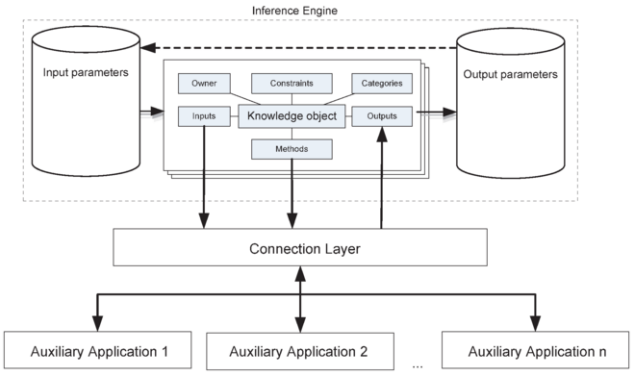

Figure 2. The use of the Knowledge Object concept in the work presented by Johansson [12].

In its simplest form, a Knowledge Object transforms input to output and contains a list of input parameters, a list of output parameters, and a method for processing input parameters to output parameters (Fig. 3). Other fields may be added such as constraints, owner, categories, precision, and comments. Owner is used to trace who is responsible for the Knowledge Object and its method (the task it performs). The field categories can be used to sort Knowledge Objects into groups. Comments are used to add information usable for explanation extractions and debugging facilities. Finally, the list of constraints and the precision value is used to allow the knowledge-bases to contain alternative Knowledge Objects.

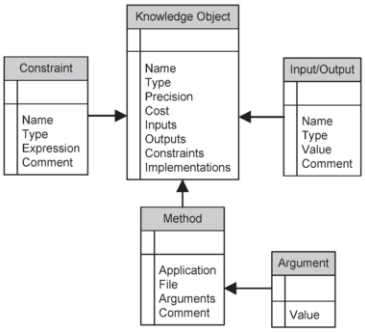

Figure 3. Knowledge Object model [13].

Knowledge Objects use external applications as methods. To pass information to the external applications, meta-data is needed (not to be confused with meta-knowledge). That meta-data is locally stored within the Knowledge Objects. The calculated parameter values are stored in a global list of parameters. When implementing the Knowledge Objects, they should be defined in a way that makes them autonomous. Since the methods used to process information preferably are external software applications, the applications should be selected keeping in mind the list of requirements imposed on the design automation system. They are the following: low effort of developing, user readable and understandable knowledge, longevity, and ease of use [10]. The benefits of developing Knowledge Objects that are autonomous using common wide-spread applications as methods are two-fold: the knowledge can be used manually without the design automation system, and it is easy to find people skilled enough to use the very same knowledge the design automation system does - it makes the knowledge more human-readable

2. Design Modelling with Knowledge Objects

The main activities for the development of an automated design system are: system output definition, customer parameters definition, product items (parts and assemblies) definition, definition of variables associated with the product items, company parameters definition, process modelling, acquisition of knowledge used in the design process, analysis of relationships at different levels, identification of problems and knowledge gaps, resolving problems and filling knowledge gaps, definition of design tasks that will constitute system embedded knowledge, system realization, and system test and evaluation. These activates are commonly performed iteratively in the pursuit of a complete solution. The work includes the definition of design algorithms, rules, and relations that transform stakeholder's parameters to product variables (e.g. properties and specifications) which results in a process structure with associated knowledge. Process formalization can be achieved by means of process modelling of the involved tasks and their relations using Dependency Structure Matrices. The tasks can be conditional dependent, or can be executed in parallel whilst others require methods to resolve mutual dependencies. The definition of Knowledge Objects cannot be based on identified tasks exclusively as they in turn can contain, or conceal, dependencies that aggravate system execution. To unwind, or at least to reveal, dependencies requires a detailed analysis on parameter level. This analysis is then followed by an elimination of all recursive dependencies, which in turn can affect the grouping of tasks and, consequently, the definition of Knowledge Objects. This alternation between domains, for revealing and, hopefully, unwinding dependencies, supporting the formation of Knowledge Objects, is depicted in Fig. 4.

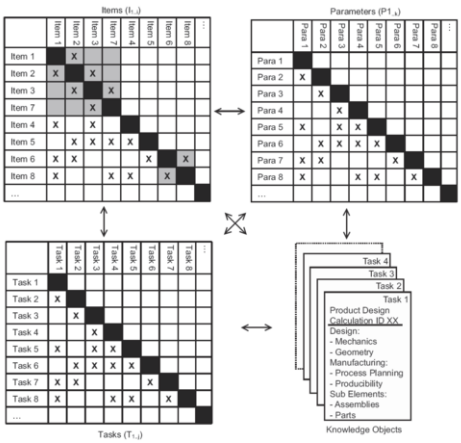

Figure 4. Definition of Knowledge Objects by means of Dependency Structure Matrices [14].

2.1. Supporting top-down and bottom-up modelling

The previously described system development activities can be performed in an arbitrary order within an iterative process. Adopting a top-down approach would imply that a complete system is defined using Items, Knowledge Objects, and Parameters without considering what really happens within the different Knowledge Objects. The Knowledge Objects would be treated as black boxes, implicitly connected by Parameters for input and output that constitutes a network for the dataflow. When the

overall structure is complete, the work with creating the realization of individual Knowledge Objects takes place. On the other hand, a bottom-up approach would start with the creation of the individual Knowledge Objects' realization (e.g. files for computing product dimensions and properties). When these realizations have been created, they are incorporated into the overall system and the relations for the dataflow are created by assigning Parameters as input and output. A mapping table supports the use of different names in the realization of a Knowledge Object to the ones used in the system. This is a functionality that supports a bottom-up approach allowing for the use of existing files without modifying it according to a pre-defined naming convention.

2.2. Modelling overlapping knowledge

It is often possible to calculate a single variable in different ways. Sometimes a heuristic rule can be used, or rules analytically derived from the fundamental laws of physics. But it is also possible to do FEM-calculations or experiments to evaluate a design variable. In addition to these four types of knowledge, an engineer needs to have the capability to decide when to use what knowledge; this is called meta-knowledge, or knowledge about knowledge. When more than one type of knowledge source is available, the question of when to use which source arises. In one state, the system may be executed in order to make a quotation calculation with only a small set of available input parameters. In the next step, detailed design is the purpose of running the system, with high accuracy as the main focus and with a larger set of available input parameters. Different kinds of knowledge are used in these different contexts, and implementing meta-knowledge would allow for flexible use. A list of constraints and a precision value need to be added to the knowledge object class in order to make it possible to have multiple knowledge objects pointing to the same parameter in the knowledge-base. The constraints dictate when the knowledge object is applicable, and the precision value tells how good the outputs are. When using constraints together with precision values, the system will run in the following sequence.

Do until the conflict set is empty:

1. List all triggered objects not violating any constraints, exclude solved objects. Sort the list by precision.
2. Execute the knowledge object with highest precision ("first come, first served" if several objects with the same precision exist).
3. Clear all output parameters in knowledge objects dependent on the outputs from the fired knowledge object. (This can cause rules to fire more than once, but is done to make sure that the output with the highest precision is the final result.)

Two different situations may occur when allowing the knowledge-base to contain multiple Knowledge Objects for a single phenomenon. Let us say that there are four triggered Knowledge Objects at one stage in the conflict set. Three of the Knowledge Objects contain knowledge about phenomenon P1, and one Knowledge Object deals with phenomenon P2. Knowledge Object number one and two have been assigned a high precision value, Knowledge Object number three a medium precision, and number four a low precision value. The design parameters A-D and I are known, but J-N are unknown. The selected Knowledge Object to run in this situation will be Knowledge Object number one. This is because it has the highest precision value and was added to the system before Knowledge Object number two. (Since Knowledge Objects one and two have the same precision value, the "first come, first served" rule is applied.)

Situation one: The parameter calculated by a knowledge object is assigned a precision value equal to or higher than the precision value of knowledge object number

one. Since knowledge object number one is selected, it will be executed using the pre-defined method. Values for parameters I-K will be calculated. However, since a knowledge object with a precision value equal to or higher than the precision value of the current knowledge object (knowledge object number one) set the I parameter, the value of parameter I will not be overwritten. Knowledge object number one will set only the parameters J and K in this situation. When updating the conflict set, knowledge objects 1, 3 and 4 will be considered solved since parameters I to K are known. The only Knowledge Object left in the conflict set is number two.

Situation two: The parameter calculated by a knowledge object is assigned a precision value smaller than the precision value of knowledge object number one. Since knowledge object number one is selected, it will be executed using the pre-defined method. Values for parameters I-K will be calculated. Since a knowledge object with a precision value smaller than the precision value of the current knowledge object (knowledge object number one) set the I parameter, the value of parameter I will be overwritten. All the parameters I-K will be set by knowledge object number one in this situation. Since parameter I is changed, all dependent parameters must be cleared. When searching the knowledge-base, it is found that knowledge objects 5 and 6 have parameter I as input. This will invalidate parameters O-T, since they were calculated using the value of the I parameter with a smaller precision. When invalidating parameters O-T, knowledge objects number 5 and 6 will be triggered and put to the conflict set. Knowledge objects 1, 3 and 4 are considered solved. At this stage, there are still four knowledge objects in the conflict set due to that the value of the I parameter has a higher precision. Firing knowledge objects 5 and 6 using this better value will (probably) increase the precision of the values of parameters O-T.

2.3. Managing Knowledge Objects

Knowledge Objects implements computations, actions, consequences and relations but they do not encapsulate the argumentation for their existence or the reason behind their design. The definitions of rules are based upon insights, decisions or facts derived from prerequisites, trial and error, experience, calculations, simulations, experiments, filed tests, literature etc. which constitutes another kind of knowledge that can provide a deeper understanding of the Knowledge Objects. A deeper understanding can be supported by the access to answers for questions, such as Why, When, Scope, Valid ranges of input/output, Origin, Supporting theories, Simplifications, Assumptions etc. The answers to these questions constitute knowledge about knowledge i.e. Meta-Knowledge or, as commonly referred to, design rationale defined as the set of reasons behind the decisions made during the design of an artefact. Two different approaches to represent design rational are Argumentation-based and Template-based [15]. In addition, traceability, defined as "…the ability to describe and follow the life of a conceptual or physical artifact." [16], across domains is also essential. Product Variant Master (PVM) [1], MOKA [6], Systems Modelling Language (SysML) [17] and CommonKADS [18] are methodologies for system development with some support for managing design rationale and traceability. Specific applications, with more or less functionality for managing design rationale and traceability, are PCPACK [19], Design Rationale Editor (DRed) [20] and Product Model Manager (PMM) [21].

Three different tasks have been identified as essential to support and these are reuse, expansion and maintenance. Reuse is the use of existing Knowledge Objects in a new context (e.g. a new product family or system foundation). Expansion implies

increasing the design space or functionality (e.g. scale the parameters' ranges or extend the topology). Maintenance concerns modifying existing Knowledge Objects according to new circumstances (e.g. changes in manufacturing constraints, material properties, manufacturing processes, legislations, standards etc.). In addition to the domains and the tasks identified above, three general enablers for successful task execution are the structuring, the validation and the adaptation of model elements. Structuring is required for the purpose of enabling searching and finding candidate Knowledge Objects for reuse, expansion or maintenance. Validation is required to ensure the applicability of candidate Knowledge Objects. Adaptation is necessary when changes are required to make the selected Knowledge Objects applicable in a new context.

The structuring of design rationale concerning Knowledge Objects is based upon the information model in Fig. 5. The main principle is to sub-divide the process into different tasks, i.e. Knowledge Objects, on a level that supports both a contextual meaning and the access to detailed descriptions. The Rational class can be used to group objects and to describe why a Knowledge Object exists, what it operates upon, what it is affected by, its relation to other objects, or in detail describe the set of Input, the set of Output, and the transformation associated with a specific Knowledge Object. The SupportingObject enables traceability to reports, protocols, guide-lines, standards, legalizations etc. for more detailed descriptions.

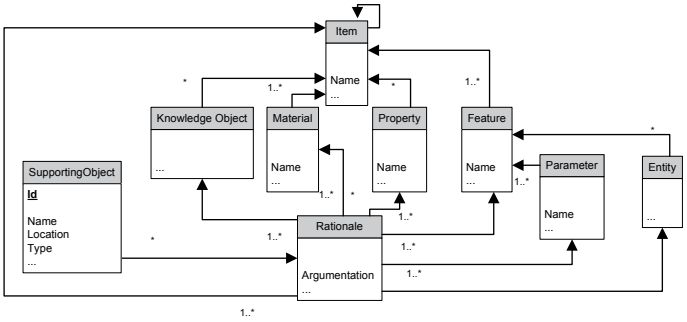

Figure 5. Information model for structuring design rational of Knowledge Objects, adapted from [22].

3. Debugging and execution of Knowledge Objects

Debugging a collection of Knowledge Objects implies, on a meta-level, that all the necessary data and their relations constituting a domain of application have been entered. An iterative process, based on the alternation between top-down and bottom-up approaches, conducted under a period of time, together with the deployment of different domain knowledge and the involvement of different persons in the system development, is not easy to manage and support is required for assessing system completeness. Means to reach a total system understanding and tools for detailed examination of data and relations are two important functions supporting system development. These functions can be provided by DSM-views on the system defined data. System defined Items, KnowledgeObjects and Parameters can be retraced and their internal relations analysed by the generation of DSM-views for the different concepts, Fig.6. Further, the concepts' interrelationships can be view and analysed by the construction of DSM-views visualising the mapping between the concepts.

However, even though all the required Items, KnowledgeObjects and Parameters have been entered into the database, the system might fail to execute. To ensure the system functionality, the database has to be checked for recursive dependencies, undefined parameters (variables), the existence of multiple providers for a parameter (variable), and the existence of knowledge objects not providing any output. These basic data checks can be performed and their status communicated to the developers continuously during system development. If any problem exists, the incorporation of the different DSM-views, both original and partitioned, can be used as an aid to examine the problem in detail (Fig. 7).

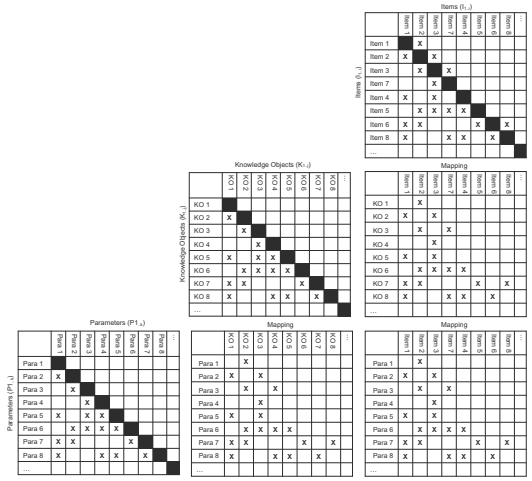

Figure 6. Principle DSM-views of concepts and the mappings between them [23].

3.1. Execution of Knowledge Objects

A collection of Knowledge Objects can executed by a predefined static flow, a run-time generated static flow or with a dynamic flow. A predefined static flow is stored or manually entered prior to execution which requires an understanding of how Knowledge Objects and Parameters are related, however, this can require extensive work and has to be redone if there has been any changes. A run-time generated static flow can be achieved by using an inference mechanism. The relations between Parameters can be modelled as directed graphs and the execution order resolved using algorithms for creating and operating on a reachability matrix. The order is then mapped to the Knowledge Objects. Since the execution sequence of the system is not fixed, the execution order can change whenever new knowledge is introduced. A dynamic flow can be achieved by forward-changing. A global list of parameters is defined and watched by all Knowledge Objects. Whenever a Knowledge Object has enough information to solve its task, it will ask the inference engine to be allowed to calculate some of the unknowns in the global list of parameters. Based on meta-knowledge about the Knowledge Object (e.g. low precision), it might be rejected. If approved to perform the task, it will write the new information in the global list of parameters, which might cause other Knowledge Object to be invoked. The execution will continue until no more Knowledge Object is to be executed.

The execution can be as a single-run based upon a specification with values for a set of initial Parameters with the purpose the find out which Parameters are affected

and their resulting values. Multiple-runs, preferably supported by Design of Experiments (DoE), enable executions based on different combinations of Parameters and their values which will support the pursue for a valid or "best" solution. Multiple runs can also be used to create response surfaces or trade-off curves that can be distributed for easy use if access to the system is limited or execution includes substantial simulations. If the Knowledge Objects can be executed without interruption, the search for a best solution can be supported by introducing optimization algorithms.

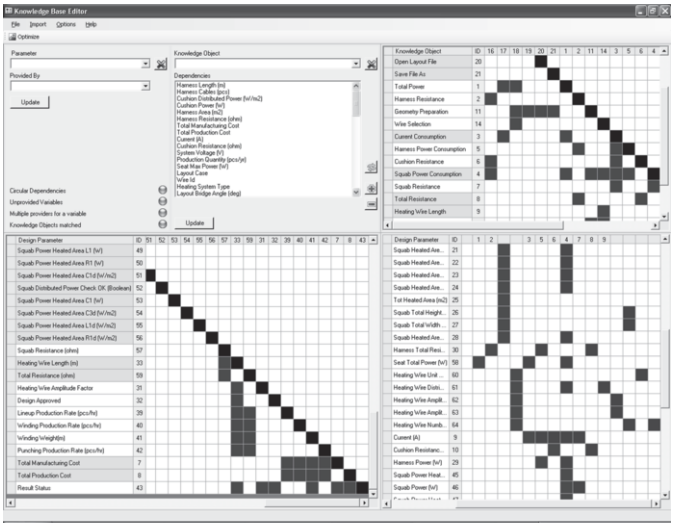

Figure 7. A graphical user interface for ensuring system completeness and functionality [23].

4. System realization based on Knowledge Objects

One example of an information model incorporating the concept is illustrated in Fig. 8 [24]. The main concepts used for knowledge modelling and representation are: KnowledgeObject, Variable and KnowledgeObj_Parameter. The different attributes of the KnowledgeObject concept are: Id, Name, Parameter (commonly a path to a file to be executed), KnowledgeObjectTypeFK (pointing at the concept KnowledgeObjType which identifies the software application for execution), KnowledgeBaseId (pointing at a concept defining the superior domain of application) and KnowledgeObjectType (used for classification). One special class is Specification comprising KnowledgeObjects not requiring any input. Variable is a central concept for the proposed model. In general terms, a Variable is a property defined by a task, i.e. a KnowledgeObject. Variable can represent different types of properties related to, by example, geometry, material, product structure, manufacturing operations, cost levels etc. The different attributes of the Variable concept are: Id, FriendlyName (can easily be interpreted), DefinedBy (pointing at a KnowledgeObject), NameInDefinedBy (supports the use of a different name in the realization of a KnowledgeObject) and Dimensions (for type declaration). The execution of a KnowledgeObject commonly requires input (parameters). Parameters are commonly defined by other KnowledgeObjects. The required input for a KnowledgeObject is defined by the concept KnowledgeObj_Parameter which includes the attributes Id,

KnowledgeObjectFK (defines the KnowledgeObject), VariableFK (defines the Variable) and ForeignParamName (supports the usage of a different name in the realization of a KnowledgeObject). The above implies that no explicit concept for parameters is needed.

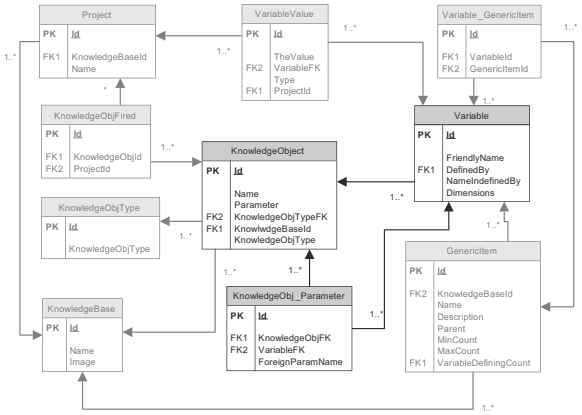

Figure 8. First example of information model for system realization [24].

A second example of an information model incorporating the concept is illustrated in Fig. 9. The main classes in that model is the KnowledgeObject, KnowlegeParameter, KnowledgeConstraint and Connection. Also, two classes CollectionOfKnowledgeConstraints and CollectionOfKnowledgeParameters were introduced to make it possible to trap and consume events raised when KnowledgeParameter values and KnowledgeObject statuses changed. A Knowledge Object in this implementation has 11 attributes, 4 methods, and 2 events. The Active attribute is used to indicate that the Knowledge Object is active or suppressed, Categories is used to group knowledge objects, Comments is used to add descriptions of the knowledge automated by the current Knowledge Object, ConstraintsSatisfied is a read only attribute indicating whether the Knowledge Object is valid based on given constraints, Cost indicates how much time it takes to apply the underlying method, Folder is used to group the Knowledge Objects, Name is used as identifier of the Knowledge Object, Owner is used to indicate whom is responsible for the automated knowledge and its applicability, Precision is used to indicate the quality of the automated knowledge, Triggered is a read only attribute indicating whether the knowledge object is ready to execute, Type is used to categorize the Knowledge Object. The ClearOutputs method is used to set any value of the indicated output parameters for the Knowledge Object to null, Clone is used to make an in-depth copy of the Knowledge Object, Execute is used to execute the Knowledge Object if active, triggered and constraints are satisfied, OnObjectChanged is used to invoke update of the knowledge base. The Executing event is used to flag that the Knowledge Object is currently executing its method, and the ObjectChanged is used to indicate that the knowledge base has to be updated. A KnowledgeParameter has 9 attributes, of which 4 are used the same way as for the Knowledge Objects. The Groups attribute is used to group the knowledge parameter, Locked is used to lock the value of the knowledge parameter in case of an optimization algorithm, ProductOf is used to indicate what Knowledge Object set the value of the knowledge parameter, Unit is used to identify the metric unit of the current value of the knowledge parameter and Value carries the

value object of the knowledge parameter. A KnowledgeConstraint has 6 attribute of which 3 are used in a similar way as for the Knowledge Objects and knowledge parameters. The Expression attribute is used to carry a string-based formula expressing the constraint, Parameters is used to indicate which parameters are involved in the expression (these parameters will also be marked as input to any knowledge object the constraint is added to) and Satisfied is a read only attribute indicating whether the constraint is true. A Connection has 5 attributes, ClassName is used to identify the class the knowledge object use for execution, ConnectionFile indicates where the class is stored, and Method indicates what method to run within the indicated class.

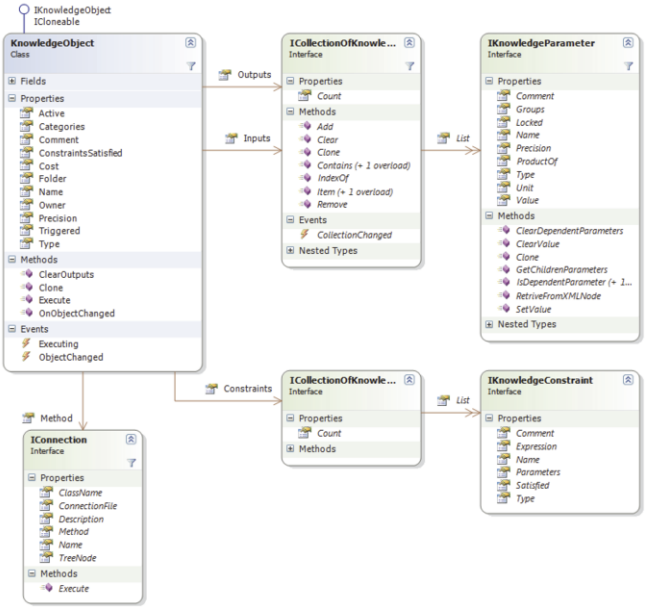

Figure 9. Second example of information model for system realization.

5. Conclusion

The concept of Knowledge Object has been described together with examples of its use in both development and system realization of design automation systems. The concept is useful for modelling and execution of design processes, tasks, and engineering knowledge. Support for traceability and understanding by mapping to other concepts describing requirements and design rational is also included for management purpose. To further improve and validate the concept, development of additional systems targeting other domains are required - this will be subject for future work.

Acknowledgement

This work was conducted within the project *Efficient Implementation and Management of Systems for Design and Manufacture of Custom Engineered Products – Impact,* and financial support from the Knowledge Foundation, Sweden, is gratefully acknowledged.

References

[1] L. Hvam, N.H. Mortensen, J. Riis, *Product Customization*, Springer Verlag, Berlin, Germany, 2008.

[2] A. Haug, L. Hvam, CRC-cards for the development and maintenance of product configuration systems, in: *Joint Conference IMCM06 and PETO06*, GITO-Verlag, Berlin (2006) 369-381.

[3] A. Haug, A. Degn, B. Poulsen, L. Hvam, Creating a documentation system to support the development and maintenance of product configuration systems, in: *WSEAS International Conference on Computer Engineering and Applications*, Gold Coast (2007) 122-131.

[4] I. Rask, *Rule-based product development - Report 1*, Industrial Research and Development Corporation, Mölndal, 1998.

[5] I. Rask, S. Sunnersjö, R. Amen, *Knowledge based IT-systems for product realization*, Industrial research and development corporation, Mölndal, 2000.

[6] M. Stokes, *Managing Engineering Knowledge – MOKA*, Prof Eng Publications ltd, London, UK, 2001.

[7] L. Blessing, *A Process-based approach to computer-supported engineering design*, PhD thesis, University of Twente, Twente, 1994.

[8] A. Duffy, M.M. Andreasen, Enhancing the evolution of design science, *Proceedings of Conference on Engineering Design* **1** (1995), 29-35.

[9] F. Burstein, System development in information systems research, in: K. Williamson (Ed.), *Research methods for students, academics and professionals – information management and systems*, Centre for Information Studies, Wagga Wagga, (2002) 147-158.

[10] F. Elgh, M. Cederfeldt, A design automation system supporting design for cost – underlying method, system applicability and user experiences, in: M.W. Sobolewski, P. Ghodous (Eds.), *Next Generation Concurrent Engineering - Smart and Concurrent Integration of Product Data, Services, and Control Strategies*, Springer Verlag, Berlin, (2005) 619-627.

[11] F. Elgh, M. Cederfeldt, (2007), Concurrent cost estimation as a tool for enhanced producibility – system development and applicability for producibility studies, *Journal of Production Economics* **109** (1-2) (2007), 12-26.

[12] J. Johansson, A flexible design automation system for toolsets for the rotary draw bending of aluminium tubes, *Proceedings of IDETC/CIE 2007*, ASME, New York, (2007).

[13] J. Johansson, *Automated computer systems for manufacturability analyses and tooling design: applied to the rotary draw bending process*, PhD thesis at Chalmers University of Technology, Gothenburg, 2011.

[14] F. Elgh, Decision support in the quotation process of engineered-to-order products. *Advanced Engineering Informatics* **26**(1) (2012), 66-79.

[15] A. Tan, Y. Jin, J. Han, A Rational-based Architecture Model for Design Traceability and Reasoning, *The Journal of Systems and Software* **80** (2007), 918-934.

[16] K. Moham, B. Ramesh, Traceability-based Knowledge Integration in Group Decision and Negotiation Activities*, Decision Support Systems* **43** (2007), 968-989.

[17] S. Friedenthal, A. Moore, A. Steiner, *A Practical Guide to SysML: the Systems Modeling Language*, Morgan Kaufmann, San Francisco, US, 2008.

[18] G. Schreiber, H. Akkermans, A. Anjewierden, R. Hoog, N. Shadbolt, W. Velde, *Knowledge Engineering and Management: The CommonKADS Methodology*, The MIT Press, Cambridge, US, 2000.

[19] Epistemics, PCPACK, http://www.epistemics.co.uk/Notes/55-0-0.htm (Acc. 24 June 2014), 2008.

[20] R. Bracewell, K. Wallace, M. Moss, D. Knott, Capturing Design Rationale, *Computer Aided Design* **41**(3), (2009), 173-186.

[21] A. Haug, L. Hvam, N.H. Mortensen, Implementation of Conceptual Product Models into Configurators: From Months to Minutes, *Proceedings of MCPC 2009* (2009).

[22] F. Elgh, Modeling and management of product knowledge in an engineer-to-order business model, in: S.J. Culley, B.J. Hicks, T.C. McAloone, T.J. Howard, P. Badke-Schaub, (Eds.), *Proceedings of the 18th International Conference on Engineering Design (ICED 11)*, The Design Society, Somerset, (2011), 86-95.

[23] F. Elgh, Knowledge modelling and analysis in design automation systems for product configuration, in: A. Dagman, R. Söderberg (Eds.), *Proceedings of Norddesign 2010*, Chalmers University of Technology, Gothenburg, (2010) 257-266.

[24] F. Elgh, A tool for automated design supporting management and analysis of quotations and product variants – information model and system principles, in: J. Pokojski, S. Fukuda, J. Salwiński (Eds.), *New World Situation: New Directions in Concurrent Engineering*, Springer, London, (2010), 361-368.