*Transdisciplinary Lifecycle Analysis of Systems*
*R. Curran et al. (Eds.)*


*doi:10.3233/978-1-61499-544-9-327*

# Howtomation© Suite: A Novel Tool for Flexible Design Automation

Joel JOHANSSON[1]
*Mechanical Engineering, School of Engineering,*
*Jönköping University, Sweden*

**Abstract.** This paper shows how to achieve flexibility in design automat systems through the introduction of knowledge objects and through the adoption of an oriented view of the product structure. To demonstrate the ideas a novel tool called Howtomation© Suite (for automated know-how) is presented. The new tool handles the addressed issues and has been successfully implemented at one company. That successful implementation is described at end of the paper.

**Keywords.** Design Automation, Knowledge Based Systems, Engineer to Order, Knowledge Base, Knowledge Object, Manufacturability Analysis, Injection Molding.

## Introduction

The ability to design and manufacture individualized products increases the competitiveness of manufacturing companies and is sometimes the business case to them [1]. Three opportunities have been pointed out for Swedish industry to stay competitive on the global market: individualized products, resource-smart design and production, and a focus on customer value [2]. These opportunities can be achieved by efficient design and manufacture of customized products. However, that requires developing and integrating knowledge based systems for products and production [3]. The research presented in this paper is part of a research project aiming at these targets.

The realization of individually engineered products can be supported by the adoption of an automated engineer-to-order (ETO) [4] approach in the quotation, the development, and the production preparation processes. Automating the ETO processes allows a company to efficiently adapt their products to vast variations of customers' specifications bringing more value to the customer and profit to the company by efficient use of engineering staff, material, and manufacturing resources. The core of such a company is the exercising of a rich and diverse knowledge base about the products, their production and the required resources for design and manufacture enabling the company to quickly go from quotation to engineering the product and to production, all while maintaining the most competitive pricing. To successfully develop, implement, and maintain that core activity requires the development and implementation of com-

[1] Corresponding author, E-Mail: joel.johansson@jth.hj.se

puter systems for efficient design of product variants with associated specifications for automated manufacturing.

Since the development of a design automation system is a significant investment in time and money and since experience has shown that problems often arise when such systems are to be implemented in current operations (i.e. after proof of concept phase) it is vital to discuss some critical issues. One of few things we can infer about future is that things are going to change. That is why flexibility is the main focus in this paper. Further, since manufacturing companies deal with physical products, issues with geometry is another important aspect addressed here.

## 1. System architecture for flexibility

Knowledge based systems (KBS) which is a result of decades of research within the field of artificial intelligence has proven to be applicable to a wide range of engineering design issues [5] also forms the foundation of the architecture described in this paper. A KBS has two vital components, the knowledge base, and the inference engine. KBS is based on the strategy to formalize the knowledge to be automated and store it within the knowledge base and to let the inference engine search for consistent states of the knowledge, states where no conflicting statements exist. In practice this means that the formalized knowledge is separated from the computer routines that are applying the knowledge. The knowledge to be stored in the knowledge base can be of different kinds, and there are many ways in which the inference engine can act [6-8].

The complexity of an artifact can be measured in two dimensions including its physical realization, and the knowledge required to comprehend it. There are artifacts that cannot be made by a single person, and there are artifacts that cannot be comprehended by a single person [8]. Just as the former calls for decomposing the product into modules, the latter calls for dividing the knowledge into chunks. Consequently, two ways of achieving flexibility are identified. One is by applying an object orientated approach to the knowledge-base. The other is to apply an object orientated approach to the product structure.

### *1.1. Object Oriented Knowledge Base: Knowledge objects*

Object-oriented programming offers the possibility to develop highly flexible software. To apply object oriented programming to the knowledge base a class of objects called knowledge objects has been proposed in [9, 10]. Figure 1 illustrates one way of implementing the knowledge object class. As seen, a knowledge object contains a list of input parameters (realized as a manager object that is basically a collection), a list of output parameters, and a method (execution method) for processing input parameters to make unknown output parameters known. Other fields may be added to a knowledge object to make the system well-functioning. Proposed additional fields are listed and explained in the Table 1 (which is *not* a complete list of the fields used in the system described at end of the paper).

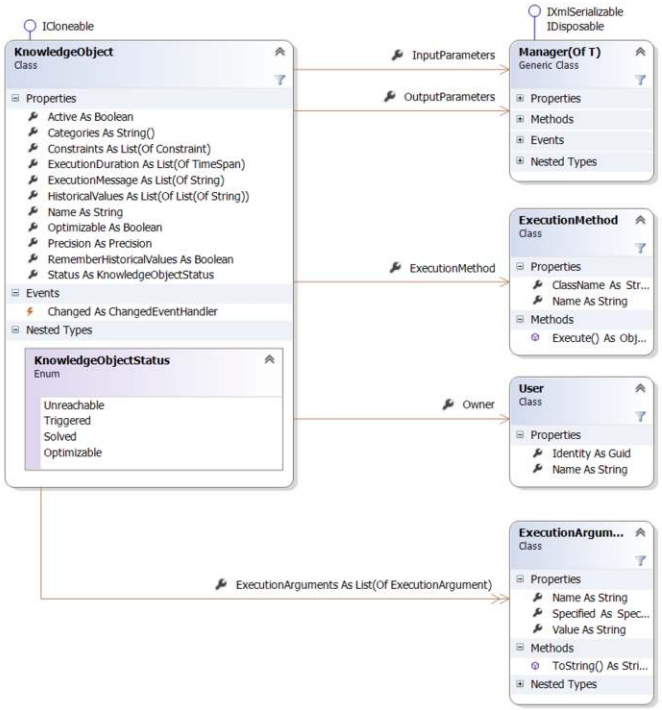


**Figure 1.** Knowledge object class definition.

**Table 1.** Attributes useful to implement for the knowledge object class.

| Field Name | Purpose |
|---|---|
| **Active** | Controls whether the knowledge object is active or not. |
| **Categories** | Categorize the knowledge automated by the knowledge object. |
| **Constraints** | Specifies when the knowledge object is applicable. |
| **ExecutionArguments** | Serves as sockets to the computer routine specified as execution method. |
| **ExecutionDuration** | Keeps track of how long it takes to execute the knowledge object. |
| **Execution Message** | Stores the resulting messages of executing the method. |
| **ExecutionMethod** | The method to run when executing the knowledge object. |
| **HistoricalValues** | Stores result values. |
| **InputParameters** | List of the input parameters for the knowledge object. |
| **Name** | The name of the knowledge object. |
| **Optimizable** | Specifies whether the knowledge object can be put into an optimization loop. |
| **OutputParameters** | List of the input parameters for the knowledge object. |
| **Owner** | Specifies the user responsible for the accuracy of the knowledge automated by the knowledge object. |
| **Precision** | Specifies the precision of the knowledge represented by the knowledge object. |
| **RemeberHistoricalValues** | Specifies whether to store input and output values. |
| **Status** | Indicates the current status of the knowledge object. |

When developing the knowledge objects, they should be defined in a way that makes them autonomous. Methods used to process the parameters should preferably be automated external software applications so that the knowledge representation is put outside of the knowledge handling system. The external applications should be selected so that the resulting design automation system contains user readable and understandable knowledge, and is easy to use. The benefits of developing knowledge objects that are

autonomous using common and wide-spread applications as methods are two-folded: 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.

### *1.2. Object Oriented Product Structure*

The second way of achieving a high degree of flexibility in a design automation system is by adopting object orientation to the product structure. When taking such a perspective on the product structure all components are viewed as objects with dimensions or other features as attributes. These objectified components can subsequently be wrapped as knowledge object where the attributes serve as input or output parameters. This can be achieved using any parametric CAD-system.

When taking an object-oriented view on the components it has proven to be good praxis to communicate dimension values and suppress states of features through user defined parameters. The parameters should then be put at an appropriate level in the product structure, which is in the tree leafs if possible. In a case where a parameter is affecting several components it is put at the lowest possible level in the product structure above the components it is controlling, see Figure 2. Any rule or calculation are put at the same level as its dependent parameters. Parameters in assemblies are inherited by descendant subassemblies and components and are repeated in them. In practice this means that the introduction of a parameter in a subassembly controls any such parameter in its descending components. This behavior can be achieved by functionality in the most common CAD-systems (the functions have different names for example publications, external link, reference), but can also be achieved by macro programming. Consequently, when replacing a descendant component it will automatically be updated to parameters in the ascendant assembly.

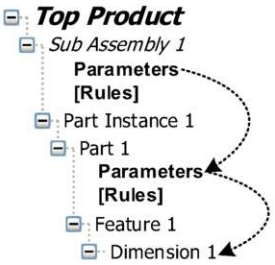


**Figure 2.** Parameters and rules should be put in leaf nodes as far as possible. Parts and components inherit parameters from ascending assembly. The brackets surrounding the rule-nodes indicates that, if any, only rules intuitively connected to the geometry should be put in the CAD-models.

### *1.3. Inference engine*

The inference engine is used to automate the formalized knowledge stored in the knowledge-base. The inference engine arranges the knowledge in the knowledge-base in an executable order. Two main types of search-based inference engines exist: forward and backward-chaining [11]. A forward-chaining (also called data-driven) mechanism uses the information initially presented to fire all applicable rules. The method has two steps. In the first step, triggered rules are listed. In the second step, an appropriate rule from the triggered ones is selected and fired. After firing the selected rule,

all triggered rules are listed again and so on, until no triggered rules are found. If knowledge objects are used to build the knowledge-base, the inference engine searches for knowledge objects with all input parameters known. It then selects one of the found knowledge objects to execute the method defined in that knowledge object to calculate output parameters using the input parameters. When the method has run, the stock of known parameters is updated, and a new search for executable knowledge objects is initiated (depth first) or executes the next knowledge object in the found executable knowledge objects (width first).

A backward-chaining mechanism (also called goal-driven) is fed with goal states. The mechanism then searches backward to see how to end up at that state. When knowledge objects are used, the knowledge-base is searched for knowledge objects to fire to find the queried parameters. The user is later asked to put in required information. The backward-chaining mechanism is more effective at runtime than the forward-chaining one. This is because executions of unnecessary methods are avoided.

Event handling is available in modern operating systems. Here, it is proposed that the inference engine should make use of these functions in the operating systems. That gives an event-based, forward-chaining search mechanism that works as follows. When a parameter is changed, an event is raised in the system notifying that a change has occurred. This triggers an update of the conflict set. If there still are knowledge objects left in the conflict set, one of them is selected to be executed, in accordance with implemented rules for selection. When the object is executed, its output parameters are changed, and the conflict set is updated, and so on. When implementing the inference engine using event handling, a significant amount of loop algorithms are avoided and when running the system, the inference engine is triggered automatically on change.

## 2. Dealing with geometry

What makes computer systems for automated engineering design outstandingly hard to develop is that geometry is a big share of the problem domain. It has proven that geometrical problems are hard to automate, some examples are found in [12]. Two main strategies exist to deal with this situation, one is to create template CAD-models that are parametrically and/or topologically modified, and the other is to generate the geometry programmatically. The former strategy is here referred to as the template based systems and the later one is referred to as generative systems, it is of course possible to combine the two strategies into hybrid systems.

The advantage of the template based approach is that it is easy to predict the outcome of the system, what you see is what you get. In the CAD-system the engineers can define parameters and add rules for updating dimensions and suppressing/activating features to make the components turn into shapes corresponding to given parameters. The drawback is that the template based approach is not scalable so that over time when adding more and more features the CAD-models are hard to maintain and hard to instantiate. This is because the template models in fact contains the complete set of the design space of the component so when instantiating the models, the entire design space is instantiated again. The system gets fragile when starting to instantiate such CAD-models into assemblies.

The advantage of generating the CAD-models programmatically is that the resulting models are lightweight compared to the template approach. It is also possible to make the generated models much more general so that the resulting models might look

completely different. The drawback is that it is hard to predict the outcome, and that it is hard for the engineers to modify the models, as they are represented by computer programming code. Real systems are of course hybrids.

### *2.1. Where to put the knowledge repository*

It is possible to implement the knowledge-base in stand-alone automated engineering design systems or into CAD-integrated KBE-systems (KBE is the acronym for Knowledge Based Engineering).

When using a CAD-integrated KBE-system to implement the knowledge-base, the rules will be listed in the model-tree among the different features. This can be valuable since it is easy to see what geometries the rules are connected to. It also makes the user feel familiar with the user interface. But the knowledge-base in such a system can be cumbersome to understand when the knowledge-base contains a vast number of rules compared to the number of geometry features. This is especially true if many of the rules do not deal with the geometry. In such cases a stand-alone automated engineering system should be used.

Another issue to consider is that when using a CAD-integrated KBE-system, the knowledge is bound to the CAD-system. This means the knowledge-base will be difficult to translate to other CAD-systems. In stand-alone systems, knowledge is automated outside the CAD-models and design proposals can be generated in native or neutral CAD-formats. Another benefit of putting the knowledge in a system outside the CAD-system is the distinct interface between CAD and the knowledge, which is usually realized by a set of parameters helping clarifying which parameters are the governing ones of the design. These parameters are the attributes of the objects when taking object oriented perspective of the product structure. One drawback with the stand-alone approach is that it can be hard to implement a knowledge-base containing mostly geometric relation-ships into a stand-alone KBE-system.

## 3. Constraints, constraints, and constraints

Product and production development involves the identification and propagation of active constraints. In product development these constraints often are referred to as the dimensioning parameters, and in production development they are often referred to as the production window. These constraints originates from laws of physics, legacy, economics, or customer and affects the physical realization of the product. These constraints can be explicitly defined using parameters implemented in the knowledge base becoming input parameters, or implicitly defined introducing new parameters that becomes output parameters.

More abstract, we also have constraints on the knowledge used to derive the product indicating the valid range of it. Take for instance the commonly used slender rod assumption. It is said to be valid if the length of the rod is much greater than the cross section of it (factor 10 is widely used), which is a constraint on the knowledge itself.

Finally if introducing optimization algorithms to search for optimum solutions mathematically modeled constraints have to be induced from the above mentioned constraints. Hence, it is important for design automation systems to be able to process constraints. Theoretical foundation for that is for example found in [13, 14].

## 4. Howtomation Suite

To verify the concepts presented in the previous sections a novel tool was developed and applied to a real life example. The Howtomation suite is based on the Microsoft .net platform and consist of five parts: core definitions for parameters, inference engine, knowledge objects, graphical user interface components, and a constraint solver.

The company where the Howtomation Suite was applied develops and manufactures heated runner systems for injection molding of plastic materials.

### *4.1. Hot runners for injection molding*

One reason for making the automation at the company was that the product is an ETO-product. Every produced hot runner system is unique. The runner systems differ in layout, see for instance Figure 3 where an examples of the X-shaped layout is shown, there are also H-shaped layouts, circular layouts, and custom layouts. The runners are connected to the tooling cavity through in-gates. The number of in-gates is up to 48 for a single system, see Figure 3. There are 5 series of in gates that all can have two different types of bushings, be of length ranging up to 600 mm, and have 9 different types of end caps.

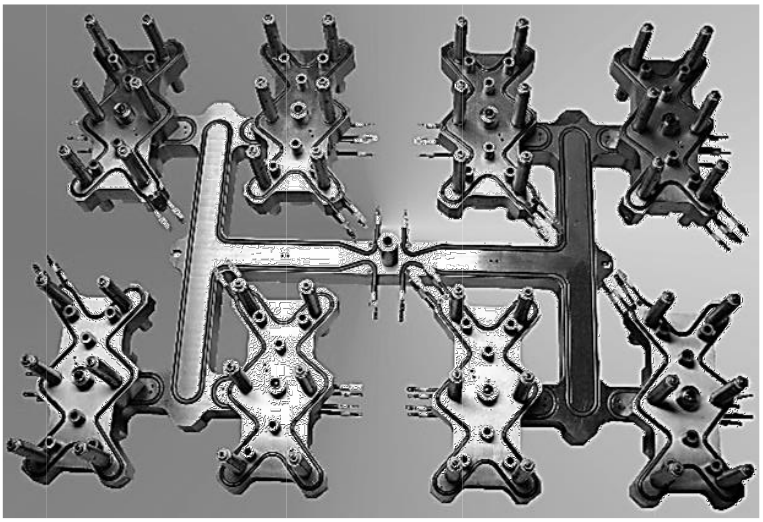

**Figure 3.** Example of hot runner system with 48 gates in X-layout.

### *4.2. Knowledge Objects*

The design of a runner system starts with planning the layout, which is done manually and results in a CAD-model containing a sketch including lines schematically illustrating the runner system. The subsequent steps are automated and is visualized by the Howtomation Suite as shown in Figure 4 and Figure 5, where executed knowledge objects are green, triggered knowledge objects are yellow and unreachable knowledge objects are red. Also, known parameters are green while unknown parameters are red. The visualization facility is used during the automation phase (design mode) but is not visible to the engineers making use of the system.

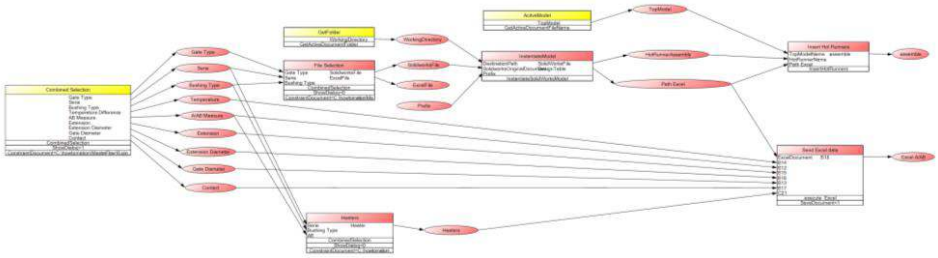

**Figure 4.** Prior to run there exist three triggered knowledge objects in the knowledge base (the picture should be viewed in color).

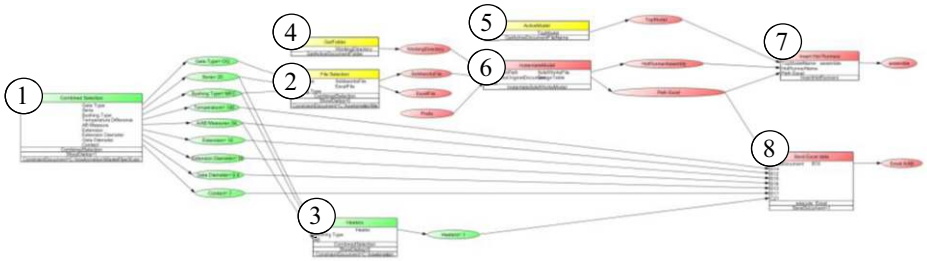


**Figure 5.** Knowledge objects are executed by the inference engine to turn unknown parameters into know. The figure show two executed objects (1 and 3). Numbers added correspond to table 2 (the picture should be viewed in color).

The process automated at the company includes 8 knowledge objects of 5 different kinds. The first three objects performs combined selections (many times referred to as configuration) applying the constraint solver included in the Howtomation Suite and that was developed based on the theories in [13, 14]. The first combined selection is based on 11 parameters and realize the company's product catalog. In that selection the constraint solver has to deal with 17 574 796 800 possible and impossible combinations of in gate parameters. The validity check of combinations are done through 7 constraints. The other two selections include selecting appropriate template CAD-models for instantiation and to select number of heating elements. Other five knowledge objects connects to the CAD-system to retrieve information about active CAD-models, instantiating template CAD-models, updating family tables, and inserting components into assemblies. See Table 2 for details about the hot runner knowledge base.

**Table 2.** Attributes useful to implement for the knowledge object class. Numbers refer to Figure 5.

| Name | Purpose |
|---|---|
| **1. Combined Selection** | Displays a dialog where the engineer can configure the gates based on customer enquiries, see Figure 6. |
| **2. File Selection** | Combined selection to pick appropriate template CAD-model. |
| **3. Heaters** | Combined selection to identify how many heating elements should be used. |
| **4. Get Folder** | Get the directory folder of the active SolidWorks model. |
| **5. Active Model** | Gets the file path of the active SolidWorks model. |
| **6. Instantiate Model** | Creates copies of the template CAD-models in to the folder of the active CAD-model. |
| **7. Insert Hot Runners** | Assembles in-gate instances into the CAD-model based on selected points or sketches (sometimes up to 48 times). |
| **8. Update Family Table** | Updates the family table to match current parameter values. |

When ready, the knowledge base is executed in release mode which means that the graphical user interface as shown in Figure 4 and Figure 5 is not visible. To make the knowledge easily accessible to the engineers a button was added to the SolidWorks user interface to execute the knowledge base. During run-time knowledge objects might enquiry the engineer for inputs, for instance when executing the first knowledge object a selection dialog box shows up where user requirements are filled in and at end of the processes he/she is asked to indicate where in the CAD-model the gates are to be inserted.

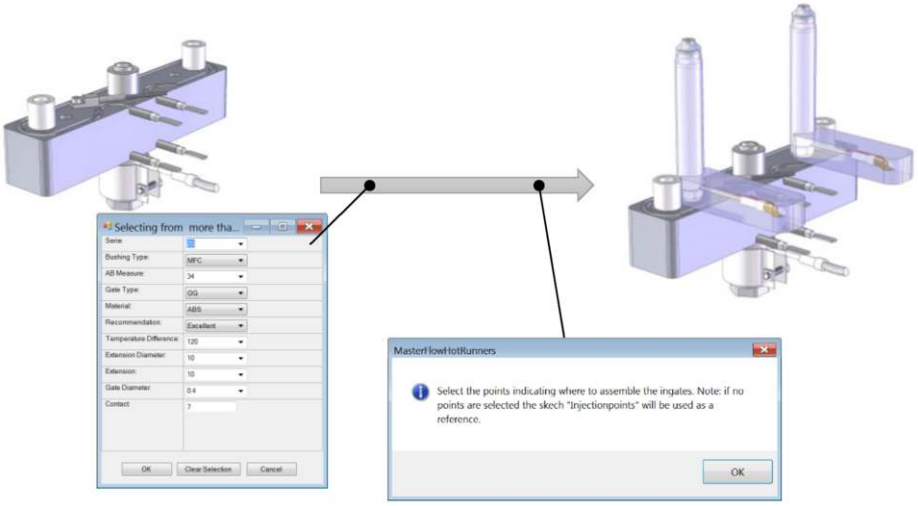


**Figure 6.** At run-time the knowledge objects are not visible to the engineer but may be interactive, here two dialog boxes show up during execution of the knowledge base. The execution is started from the CAD-system.

### *4.3. Object orientation applied to the hot runner product structure*

When formalizing the knowledge and the product structure a previously developed (huge) design table containing the complete design space ($17.5 \cdot 10^9$ combinations) was sliced down to a set of design tables with no rules connected to template CAD-models. The rules were put in the Howtomation Suite instead. Previously the complete design space of the gates were instantiated together with each instance of the gates (48 times in the Figure 3) which made the CAD-system break down after up to an hour of crunching. Now the gate instance are light weight bakeries resulting from the automated process.

### *4.4. Geometrical problems to handle*

There was one geometrical problem to overcome during the automation process and that was encapsulated into the knowledge object that inserts the gates into the main assembly (nr 7 in Figure 5 and Table 2). When defining the layout of the hot running system the gate locations are defined by a 2d sketch. The lines in the sketch defines the channels of the runner system and all the end points of the sketch defines the locations of the gates. A routine had to be developed that looped through all the curves of the sketch to identify the endpoints.

## 5. Conclusion

Products and their underlying knowledge change over time and an automated engineering design system needs to be flexible so that product components and pieces of knowledge can easily be added, updated, or deleted without disrupting the operation of the system. Adopting an object oriented perspective on the product structure and the introduction of knowledge objects to define autonomous chunks of knowledge have proven successful to achieve such flexibility when developing design automation systems. In this paper a platform for automating engineering activities by implementing these ideas was presented together with an in production application, hot runner systems for injection molding of plastics. The knowledge objects introduced in that system are supporting selection of gates for the hot runners using constraint processing algorithms. Subsequently template CAD-models are selected, instantiated, updated and assembled. The execution of the knowledge objects is controlled by an inference engine and to make the system easy to maintain a graphical user interface was developed.

## Acknowledgements

The work has been carried out within the project IMPACT, funded by the Knowledge Foundation (KK-stiftelsen), Sweden.

The Howtomation© suite was tested at the company MasterFlow and the author is grateful for their enthusiasm and willingness to adapt the system.

## References

[1] L. Hvam, N.H. Mortensen, Riis, *Product customization*. 2008; Available from: http://public.eblib.com/choice/publicfullrecord.aspx?p=336869.

[2] Vinnova, *Challenge-driven innovation - Vinnova's new strategy for strengthening Swedish innovation capacity*. Vinnova information vi 2011:07. 2011, Stockholm, Sweden: Vinnova.

[3] N. N., *Factories of the future : multi-annual roadmap for the contractual PPP under Horizon 2020*. 2013.

[4] J. Gosling, M.M. Naim, Engineer-to-order supply chain management: A literature review and research agenda, *International Journal of Production Economics,* 122(2), pp. 741-754, 2009.

[5] A.A. Hopgood, *Intelligent systems for engineers and scientists*, CRC Press, Boca Raton, 2001.

[6] J.-W. Choi, Architecture of a knowledge based engineering system for weight and cost estimation for a composite airplane structures, *Expert Systems with Applications*, 36(8), pp. 10828-10836, 2009.

[7] J. Wang, A cost-reducing question-selection algorithm for propositional knowledge-based systems, *Annals of Mathematics and Artificial Intelligence*, 44(1-2), pp. 35-60, 2005.

[8] C.Y. Baldwin, *Design rules / the power of modularity*, MIT Press, Cambridge, 2000.

[9] F. Elgh, J. Johansson, Knowledge Object - a Concept for Task Modelling Supporting Design Automation, in J. *Cha et al. (eds.): 21th ISPE International Conference on Concurrent Engineering, 8-11 September, Beijing, China*, IOS Press: Amsterdam, pp. 192-203, 2014.

[10] J. Johansson, *A flexible design automation system for toolsets for the rotary draw bending of aluminium tubes*, in *2007 ASME IDECT (DFMLC)*. 2007.

[11] G.F. Luger, *Artificial intelligence : structures and strategies for complex problem solving*, Addison-Wesley, Harlow, New York, 2005.

[12] J. Johansson, F. Elgh, How to successfully implement automated engineering design systems: Reviewing four case studies. in: *C. Bil (eds.): Proceedings of 20th ISPE International Conference on Concurrent Engineering (CE2013), Sep, 2 - 5 2013*, Melbourne, Australia, IOS Press, Amsterdam, pp. 173- 182, 2013.

[13] R. Dechter, *Constraint processing*, Morgan Kaufmann, San Francisco, 2003.

[14] T. Frühwirth, S. Abdennadher, *Essentials of constraint programming*, Springer, Berlin New York, 2003.