Abstract

Knowledge is a vital component of engineering design. Computer systems enriched with logic and engineering knowledge can support engineering design by automating repetitive and time-consuming processes. This automation is best structured in the framework concept of a Design and Engineering Engine, applying Knowledge Based Engineering techniques.

The lack of recognized methodologies implies significant investments for the development and maintenance of Design and Engineering Engines.

To alleviate the required effort a Domain Specific modeling Language is developed, enabling the representation of conceptual classes of the problem domain and is considered a visual dictionary of noteworthy abstractions, domain vocabulary and knowledge content. A Domain Specific modeling Language provides an intuitive environment to model domain specific engineering knowledge and enables a tight coupling between modeled knowledge and software program code.

The Domain Specific modeling Language is used to develop a Knowledge Based Engineering application dedicated to aircraft wiring harness design processes.

1 Introduction

The increasing complexity of aircraft systems not only provides business with opportunities, it also increasingly complicates the design process. Consequently, an increase in lead-time and engineering resources is required to ensure a satisfactory design conforming to the many, often contradicting customer requirements. However, current market dynamics demand shorter lead-times and the possibility to allow for changes in the requirements in late stages of the design process. Together with the globally increasing demand for air transport [1], many engineering companies are forced to enlarge their engineering capacity.

Nonetheless, considering the western economies the trend is opposite [2] [3]. The available engineering workforce capable of performing complex engineering tasks is static and expected to decrease over the coming decades. Senior engineers are retiring, whilst fewer engineers with appropriate educational levels are graduating. Hence, increasing engineering capacity will require an increase in productivity. In order to meet the market demands, aircraft design requires a paradigm shift to exceed present design process efficiency.

A methodology that supports this paradigm shift is Knowledge Engineering (KE), providing a number of methods and tools that guides the process of acquiring, structuring and implementing engineering knowledge.

Knowledge is a vital component of engineering design and significant reductions in costs and product development time can be realized if engineering knowledge would be reused to a larger extent and more often. Where previously the geometric model took a central position, today design knowledge should have
the focus: knowledge should be managed and engineered as a key business asset [4].

Computer systems enriched with logic and engineering knowledge can support engineering by automating repetitive and time-consuming processes. This reuse of knowledge decreases the engineering resources required and relieves the engineers from non-value adding activities, making more time available to exploit their creativity and engineering skills. Knowledge Based Engineering (KBE) techniques enable this automation of repetitive and time-consuming processes. KBE applications contain a parametric product model, enabling designers to explore the design space by defining and analyzing new design topologies and configurations.

Although huge benefits can be gained, Knowledge Engineering and KBE techniques are not yet widely adopted by industry. The lack of recognized methodologies implies significant investments for the development and maintenance of KBE applications. Besides, developed design applications can be considered ‘black boxes’, since the automated procedures are poorly documented or inexpressive making the process opaque. With the current generation of KBE systems the expert knowledge embedded in the KBE applications is hard-coded using object-oriented programming languages. Modifying design options, for example changing constraints or adding design rules, is time-intensive, obscure, and requires skilful knowledge on software development. Consequently, many engineering companies currently ask for the incorporation of their best practices and domain expert knowledge in software packages supplied by engineering software companies. This can increase development cost and comes with main disadvantages. First, the engineering software companies will have their own priorities, decreasing the customer’s influence on development, maintenance and support. Second, the corporate knowledge might become available for other companies as well.

It can be concluded that there is a clear need for KBE applications with increased accessibility and maintainability. In order to reduce the development time and increase the accessibility and maintainability of modern KBE applications an intuitive model-driven design methodology is investigated, based on a Domain Specific modeling Language (DSL).

This paper investigates the contributions of a DSL to provide engineers with an expressive knowledge model providing understanding of and insight in the program software code underlying the parametric product model. First an overview of the Knowledge Based Engineering fundamentals is given in section 2, discussing the concepts of Design and Engineering Engine and Multi-Model Generator. Knowledge modeling is discussed in section 3, addressing the definition of knowledge, knowledge bases and introducing the concept of Domain Specific modeling Languages. Section 4 considers the implementation of knowledge in KBE applications. It addresses the utilization of a DSL in the light of model-driven design and provides an example of a KBE application developed to support aircraft wiring harness design. The paper is concluded with discussions in section 5.

2 Knowledge Based Engineering

According to Milton [5] KBE concerns the development of software systems to support engineers, usually design engineers, to increase their productivity. KBE systems are considered a subset of Knowledge Based Systems empowered with Computer Aided Design (CAD) capabilities. KBE systems are able to capture and reuse product and process engineering knowledge to solve engineering problems involving the manipulation of geometry, product configuration and analyses and computations. The main objective of KBE is reducing time and cost of product development by means of the following:

- Automation of repetitive and non-creative engineering tasks
- Support of multidisciplinary integration during the conceptual phase of the design process

The KBE cornerstones are rule-based design, object-oriented programming, and parametric CAD [6]. KBE focuses on capturing
and reusing engineering knowledge to automate repetitive, non-creative human processes. This automation is best structured through the framework concept of a Design and Engineering Engine.

### 2.1 Design and Engineering Engine

A Design and Engineering Engine (DEE) is defined [5] as an advanced design environment that supports and accelerates the design process of complex products through the automation of non-creative and repetitive design activities. Figure 1 shows the DEE concept.

The main components of the DEE are the Initiator, the Multi-Model Generator, the Analysis tools and the Converger and Evaluator.

The Initiator is responsible for providing feasible initial values for the instantiation of the generative parametric product model.

The Multi-Model Generator (MMG) is responsible for instantiation of the product model and extracting different views on the model in the form of report files to facilitate the discipline specialist tools. The MMG forms the heart of the DEE.

The Analysis (discipline specialist) tools are responsible for evaluating one or several aspects of the design in their domain or discipline (e.g. structural response, aerodynamic performance or manufacturability).

The Converger & Evaluator is responsible for checking convergence of the design solution and compliance of the product’s properties with the design requirements and generation of a new design vector. These elements are addressed iteratively in order to define a feasible design solution satisfying the requirements definition.

### 2.2 The Multi-Model Generator

Designers think of an aircraft as a collection of objects providing conceptual solutions to fulfill functional requirements such as generating lift and providing controllability (Figure 2) [7].

The MMG is able to support designers in the conceptual and preliminary phase of aircraft design by providing a suite of parametric building blocks, the HLPs. They represent classes of objects encompassing sets of design rules that determine parameter values to instantiate objects composing the aircraft. The collection of HLPs or object classes is called the product model and provides a parametric view of the aircraft. The HLPs can be individually tailored and assembled to compose a large number of different aircraft configurations and topologies. Capability modules (CMs) are applied to extract specific aspects of the product, e.g. an aerodynamic mesh or a Finite Element model, to facilitate the analysis tools representing the various engineering disciplines.

The concept of the MMG is drawn in Figure 3.

The product models constructed using the current KBE systems like the ICAD system or Genworks’ GDL are object-oriented and based on general-purpose programming languages. The object-oriented programming approach allows complex products to be modeled as modular systems composed of sub-systems thus
supports the engineers in their object-oriented perception of the world.

Consequently, the concept of the MMG supports a modular approach to engineering design problems and offers the designers a more effective approach to visualize their ideas, compared to the approach offered by conventional CAD systems. The HLPs are considered primitives storing engineering knowledge. Instead of geometric primitives incorporated by CAD systems, the MMG is oriented to knowledge primitives.

3 Modeling Knowledge

Knowledge is defined as (i) the information, understanding and skills that you gain through education or experience; (ii) the state of knowing about a particular fact or situation [8]. Besides multiple definitions, there are also various classifications of knowledge. One important dimension defines procedural knowledge versus conceptual knowledge.

Procedural knowledge concerns processes, tasks and activities. It describes the conditions, under which specific tasks are performed, the order in which tasks are performed and the resources required to perform tasks.

Conceptual knowledge concerns the description of concepts and their relation to other concepts. Hence, it addresses the ways in which objects are related to one another and their properties. An important form of conceptual knowledge concerns taxonomies, i.e. classes and class membership. Another type of conceptual knowledge addresses attributes of concepts.

3.1 Knowledge Base

In order to support the reuse of engineering knowledge by KBE techniques, knowledge is captured and stored in a knowledge base, a repository containing a formal description of knowledge representing the expertise of a particular domain [5].

To represent the domain expertise efficiently, the structure of a knowledge base is identical to the structures that underlie human expertise or psychological registration. Psychologists have found that this is based on four main components; objects, attributes, values and relations [5]. According to the Methodology and tools Oriented to KBE Applications (MOKA) a knowledge base contains two main types of diagrams to visualize the knowledge components [9]; a process model and a product model.

The first diagram is a process flow chart, defined as the process model. The process model focuses on the activities performed by actors and is oriented to the ‘input-behavior-output’ perspective. It mainly contains procedural knowledge.

The second diagram is a product composition tree, called the product model. The product model is product centric and object-oriented and provides a hierarchical decomposition of the system into subsystems. It is oriented to the ‘object-relation-object’ (triple) perspective and mainly contains conceptual knowledge.

Besides the two main types of diagrams, the knowledge base is structured into two distinct segments, the informal model and the formal model. Both models incorporate the two types of diagrams.

The informal model is used to capture and validate the expert knowledge in close corporation with the domain experts. The engineering knowledge is represented using natural language, terminology from the domain under consideration and pre-defined forms. The incorporated process model focuses on the activities performed by the engineers. The
product model provides a methodic decomposition of the system into conceptual classes or objects representing subsystems.

The formal model provides a more formal view of the problem. It provides different features of the objects of the informal model, for example structure, behavior and functions. The process model, also called the design process model, is considered a blueprint for the process of the DEE. The formal model suits the developers of KBE applications, so-called knowledge engineers and the developers of software in general. To that purpose it uses a more formal representation. Within the MOKA methodology the MOKA Modeling Language (MML) is used. The formal model takes the informal model as fundamental input and takes into account aspects from the employed KBE system. The structure of the both the informal and the formal model is displayed in Figure 4.

Using a two-fold structure for the knowledge base, the informal model acts as an analysis model, developed to obtain understanding of the domain. The concepts in the domain of interest are organized without any consideration of their role in the software application. It is completely decoupled from the design process model, representing the process of the KBE design application.

It can be stated that the two-fold structure increases the development time and decreases the accessibility and expressiveness of the software code, since there is no direct link between the object-oriented code structure and the product models. The application of one single model based on a Domain Specific modeling Language (DSL) can combine the intuitive interpretation of the informal model with the software design of the formal model.

3.2 Domain Specific Modeling Language

While traditional modeling languages like the Unified Modeling Language (UML) aim to be as generic as possible to serve a broad range of domains, a Domain Specific modeling Language (DSL) is carefully defined to allow modeling of systems within a particular problem domain. A DSL enables a representation of conceptual classes of the physical world and is considered a visual dictionary of noteworthy abstractions, domain vocabulary and knowledge content of the domain under consideration [10].

In addition to an ontology defining the types of components that exist in a particular domain hence the knowledge base (solution space), a DSL populates the components with domain specific knowledge (design options) and can be considered an instantiation of the ontology. The representation is provided by a domain model, a graphical and object-oriented model intended to convey aspects and attributes in order to represent a selective and organized abstraction of knowledge relevant to the problem domain [11]. The domain model can be considered a joint effort between domain experts and the knowledge engineers, providing a means of communication to visualize, structure and validate their conceptual ideas.

As opposed to the MOKA methodology, a DSL provides one single domain model to represent the relevant engineering knowledge. The incorporated product composition tree or product model incorporates characteristics of the employed KBE system, while expressing the objects or primitives in domain vocabulary. This
product model indeed is the parametric product model encompassing the HLPs. The HLPs are applied to represent the conceptual objects or subsystems individually or mutually and integrate software design methodologies, for example addressing the reuse of code. An ‘airfoil HLP’ can be used to define objects such as wings, tail planes and control surfaces. Considering the HLPs, it can be stated that a DSL forms the medium between the mental model and the program code.

The next chapter focuses on the support offered by a DSL to the implementation of conceptual knowledge in the software model.

4 Implementing Knowledge

Captured and validated domain expert knowledge can be implemented in the concept of the DEE to enable the automation of repetitive non-creative engineering tasks.

The parametric product model of the MMG embedded in the DEE is developed using an object-oriented approach and a general-purpose programming language. Although many software developers benefit from applying object-oriented capabilities to program code, the main advantage is obtained when the code expresses the concepts of the related domain model [11]. The object-oriented programming approach allows the software model to resemble the abstractions of the domain model. A DSL makes the program code more interpretative and expressive thus alleviating the effort required to implement engineering knowledge into KBE design applications.

4.1 Model-Driven Design

The combination of OOP and a DSL tightly couples the implementation of engineering knowledge into program code to the domain model of the knowledge base. Model driven design is a methodology that supports the development of software code based on expressive models and guides the implementation of conceptual knowledge in the software classes representing the parametric product model. A DSL raises the level of abstraction of the domain model, since the modeling language represents the problem domain with domain specific concepts in natural language definitions [12]. This in turn reduces the amount of effort related to conceptual mapping. The HLPs from the problem domain are mapped one on one into the software domain to form the software classes: the program code can be designed based on the domain model and hence becomes an expression of the model (Figure 5): the domain model will provide insight in the composition of the software model and adds meaning to the MMG program code [10] [11]. This insight will increase the success rate of KBE applications, since lack of user involvement in software development is one of the main reasons for project failure [13].

To ensure the software model resembles the domain model, the HLPs are designed applying an iterative and agile development process. Designing software programs in general implies that during several iterations of the development lifecycle, additional implicit knowledge might be discovered and made explicit. The consequential modifications to the design of the MMG should result in changes in the associated domain model and vice versa in order for the domain model to sustain understanding of and insight in the structure of the program code. Not just the program code, also the domain model is developed iteratively.

In the next section, the domain driven
design methodology will be applied to develop the HLP software classes of a DEE in order to support the design process of aircraft wiring harnesses.

4.2 Aircraft Wiring Harness Design Application

Aircraft electric wiring harnesses can be comprised of hundreds of cables and ten thousands of wires, providing electrical connectivity between all the mission and vehicle systems ensuring sufficient redundancy and reliability. Aircraft wiring design has a repetitive, time consuming and rule-based nature making it a well-suited domain to develop KBE design applications [14]. The development of the application is performed in close corporation with a main, international player on the aircraft electric wiring market, regarding both design and manufacturing.

For the wiring harness design process, one of the key opportunities is the assignment of signals at production breaks, where connectors connect the different wiring harnesses (Figure 6). Each wiring harness connector can include up to 150 slots, called pins, to accommodate a signal. The pins can vary in size, as do the wires transferring the signals to be assigned. This process of pin assignment is repetitive and time-consuming due to the vast quantity of signals to be assigned and rework caused by changing input, e.g. governed by design iterations for the aircraft structural design. Furthermore separation of signals across multiple wiring harness segments is enforced by numerous opposing design rules and regulations, e.g. redundancy of flight controls and electromagnetic compatibility.

4.3 Wiring Harness Domain Specific Modeling Language

A DSL for wiring harness design is developed in order to support the design of a DEE. The knowledge base and the associated DSL are developed using the Knowledge Management (KM) software suite PCPACK5 [15], using the Extensible Mark-up Language XML to store the knowledge. The ontology used to structure the

Fig. 6 Connectors applied at a wiring harness production break

Fig. 7 Part of knowledge base taxonomy encompassing conceptual classes and software classes
knowledge base is derived from the MOKA ontology, oriented to general engineering.

Besides relationships and attributes, there are four components constituting the knowledge base: objects, constraints, activities and rules. Concerning the pin assignment process, four main conceptual classes of objects or primitives are identified: signal, production-break, connector and pin. Next to the conceptual classes, software classes are defined. This contains the definition of objects according to the KBE system. The KBE system utilized for the development of the MMG is Genworks’ GDL [16]. Therefore the sole software class of the knowledge base embodies GDL objects. Both the conceptual classes and the software classes can be considered super classes of objects, as displayed by the taxonomy in Figure 7.

The super classes enable the integration of relevant attributes from respectively the wiring harness domain and the software domain into the class definition of objects. Each object type has a specific annotation template, used to define characteristic properties (attributes) of the class. By specifying the values of the properties, specific instances of classes are defined. An example annotation template is given in Figure 8.

Instances of the four object classes constitute the hierarchical composition of the product model and are related by the - has part - relationship. The geometry of the connector is based on the child ‘back-shell’, which is an instance of the class ‘circle’: although the back-shell is a concept in the philosophical domain, it is not considered a primitive in the problem domain. By defining new classes, for example rectangular connectors, additional design options can be defined and new configurations can be generated. In the example of rectangular connectors, the class ‘rectangular connector’ is added to the knowledge base, while the associated back-shell becomes an instance of the class ‘box’, another primitive defined by the GDL software classes. The pin primitive is reused to provide the rectangular connector with a set of contacts.

The DSL enables the domain model to resemble the structure of the software code underlying the product model in an intuitive diagrammatic style. The conceptual classes of the domain model and the software classes of the software model have become each others domain equivalent, describing knowledge on different levels of abstraction. The domain model alleviates the effort required to implement knowledge into the software code embodying the HLPs. Hence it decreases the time required for the development of the application. Furthermore, engineers are not only capable of operating the design application; they will also gain better understanding of the process that is applied in order to obtain the results.

To empower automatic analysis of the pin assignment problem a multi-agent task environment is used. The environment integrates the components of the DEE into a framework and provides communication between the components through software agents [17]. Next to the MMG developed using GDL, the functions of the Converger and Evaluator are fulfilled by ILOG’s CPLEX optimizer [18].

Using the resulting KBE application, the lead-time for the pin assignment process is reduced by approximately 80%.

<table>
<thead>
<tr>
<th>connector-instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Parent</td>
</tr>
<tr>
<td>Children</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Attribute</td>
</tr>
<tr>
<td>connector</td>
</tr>
<tr>
<td>production-break</td>
</tr>
<tr>
<td>pins, back-shell</td>
</tr>
<tr>
<td>Connecting element between two wiring harness segments</td>
</tr>
<tr>
<td>connector-signal-data</td>
</tr>
<tr>
<td>data-directory</td>
</tr>
<tr>
<td>description</td>
</tr>
<tr>
<td>input-data-file</td>
</tr>
<tr>
<td>sequence</td>
</tr>
<tr>
<td>serial-number</td>
</tr>
<tr>
<td>x-coord-connector</td>
</tr>
<tr>
<td>y-coord-connector</td>
</tr>
<tr>
<td>z-coord-connector</td>
</tr>
</tbody>
</table>

Fig. 8 Annotation template to specify properties for object instances
Discussion

A DSL alleviates the effort required to create new design options and configurations in the knowledge base using an intuitive interface to the domain specific engineering knowledge. It fully supports the reuse of existing models.

However, the functional decomposition and the conceptual mappings from function to object and object to HLP remain of vital importance for the success of any design or KBE application. The opted compositions resulting in the structure of the domain model should be considered engineering strategy.

The tight coupling between the domain model and the software code creates the opportunity to automatically generate code from the domain model, providing the structure for the software classes of the HLPs. An interface between the knowledge base and the employed KBE system has been designed, based on the intermediate data format of XML.

The domain model provides the definition for the design options (object classes) as well as the configuration (relations). Using Extensible Stylesheet Language XSL technology, the knowledge relevant to the KBE system is extracted from the knowledge base. Using a standard XML parser the data is transformed into s-expression format. The s-expressions are then translated into GDL code using a custom GDL application: the conceptual definitions of the HLPs are transformed into code. Initial tests focus on regenerating the code from the domain model to match the original program code. The automatic generation of program code underlying the MMG will be topic of further research.

Acknowledgement

The author would like to express his gratitude to Dave Cooper, Genworks and Clive Emberey, Epistemics for their support and contributions.

References


Copyright Statement

The authors confirm that they, and/or their company or institution, hold copyright on all of the original material included in their paper. They also confirm they have obtained permission, from the copyright holder of any
third party material included in their paper, to publish it as part of their paper. The authors grant full permission for the publication and distribution of their paper as part of the ICAS2008 proceedings or as individual off-prints from the proceedings.