Title:
Translating MOKA-Based Knowledge Model into Software Model

Authors:
Devaraja Holla Vaderahobli, devaraja_vaderahobli@infosys.com, Infosys Limited
Narayanan Chinthavalappu Chidambaran, narayanan_c@infosys.com, Infosys Limited

Keywords:
Knowledge Based Engineering (KBE), MOKA

DOI: 10.14733/cadconfP.2015.280-284

Introduction:

The idea behind Knowledge Based Engineering (KBE) is to capture the generic knowledge of the product family and their design process; embed within KBE applications (also known as "application code") which will be used in the development of similar products. This application code is often tightly coupled with commercial CAD software such as CATIA V5. The knowledge embedded in this application code is often not understood by designers and domain experts and hence its reuse is limited. The design requirements of the product are often masked inside the code resulting in lack of traceability between requirements and application code. When a requirement changes, the application code may have to be changed manually by an expert. Additionally, the current KBE systems rely heavily on the commercial CAD software for which it has been developed.

To address these issues, the underlying product and process knowledge should be represented in a CAD software-independent form and this generic model translated into application code for targeted CAD software. There have been several knowledge engineering research activities in academia and industry to represent product and its design process knowledge such as CommonKADS, Design Knowledge Acquisition and Redesign Environment (DEKLARE), and Methodologies and tools Oriented to Knowledge based engineering Applications (MOKA). But these methodologies do not readily offer a means to translate the product and process knowledge to software code. In our research, we propose a methodology to translate a MOKA knowledge model into software code.

Main Idea:

In this paper, an approach to translate MOKA based knowledge model into software model and a generic application skeleton code is presented. Various elements of knowledge model are mapped to respective elements of software model / typical application code structure so that the generic application code can be re-generated in a consistent way from knowledge model. This helps to distinguish knowledge from application code and at the same time ensures traceability between knowledge model and the application code.

As per the MOKA methodology, the first step is to create the Informal Knowledge Model. This involves structuring the knowledge into five categories – Entities, Constraints, Rules, Activities and Illustrations. Entities represent the break-down of the product. Constraints capture design boundaries. Activities represent design processes. Rules capture design calculations. Apart from these, relationships between these knowledge objects are identified and represented in the form of diagrams. Once the Informal Knowledge Model has been created, Formal Knowledge Model is built. It is created by extending the Informal Knowledge model and it comprises of Product Knowledge Model and Design Process Model. Product Knowledge Model represents various views of the product (such as structure, representation, technology, function, and behavior) and Process Knowledge Model represents the design process. This completes the definition of the product and design process. The focus of this paper is to translate this product and design process model into a software model which can then be implemented as KBE application in specific software platforms.

There have been several attempts made earlier, as reported by Emberey et. al, [1], Skarka et. al. [4] and Lohith et. al. [2] to create CAD model/design in CATIA V5 by referring to MOKA based knowledge models. Emberey et. al, [1] creates Informal Knowledge Model to capture design process knowledge and uses this for creating design application. Use of MOKA knowledge model is mostly driven from the perspective of knowledge capture and representation. Skarka et. al. [4] describes the way the Informal Knowledge Model has been used for building the generative model in CATIA V5 Knowledgeware. Lohith et. al. [2] describes the way to create generic CAD models using CATIA V5 Knowledgeware from Formal knowledge model in a traceable manner. Our focus in this paper is more on creating software model and generic KBE application code structure that covers both CAD as well as non-CAD applications.

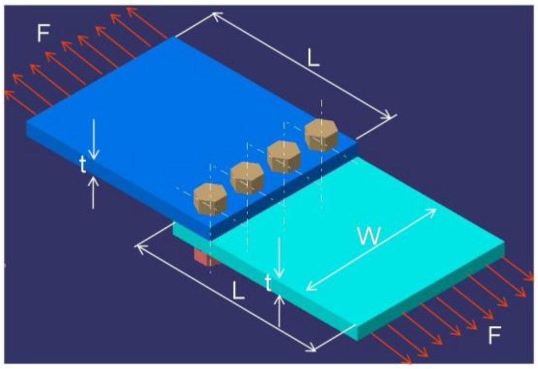
In order to realize this knowledge in a software code, a software model layer has to be created. Various elements of the knowledge model are mapped to respective elements of software model / typical application code structure so that the generic application code can be re-generated in a consistent way from knowledge model. This helps to distinguish knowledge from application code and at the same time ensures traceability between knowledge model and the application code.

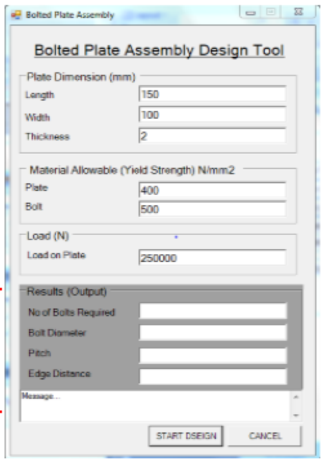
Some rules to translate knowledge model into generic application skeleton code are proposed. Entities are mapped to classes and class objects. Activities are mapped to member functions of classes. Constraints are mapped to global constants. Rules are mapped to global functions. The attributes of Entities (Product, Assembly, Part, and Feature etc.) are translated as attributes of relevant class and class objects. In addition to these elements (such as classes and methods captured from the knowledge model), there could be additional implementation-specific classes. For example, a solid model may be realized in CAD software through classes specific to that CAD software which would create geometric profiles and extrusions. These classes may be represented only in the software model and not in the knowledge model. However, these will not be discussed in the current approach.

Once the software model has been formulated, the software model elements (such as classes, constants, methods etc.) can be mapped to the appropriate code syntax of the targeted platform (such as VB .NET). In complex systems, within the software model itself, one can demarcate a pure software model which is programming language-independent and a layer which is specific to the targeted implementation language. When the targeted programming language or CAD software changes, only this adapter which connects the software model with the language / CAD platform needs to be changed. Additionally, in this system, any change in the upstream (such as requirements or product specifications) gets percolated down to the application code and they are traceable.

A typical example, “Design of Plate Assembly” is used to demonstrate the translation of Formal Model to KBE application. Figure 1 gives the picture of a plate assembly comprising of two plates connected together through multiple fasteners. Depending upon the load (F) that the plate assembly is supposed to carry, the fasteners (diameter and length) have to be designed. Figure 2 shows the UI of the tool highlighting the input and output expected from the tool. As the intent was only to illustrate the methodology, it is considered only a part of the plate assembly design i.e. only the fastener design of plate assembly. The knowledge of the design of this plate assembly is captured and represented in the form of MOKA Informal and Formal knowledge models. The knowledge models have been created in PCPACK, a knowledge modeling tool. In this illustration, we have demonstrated the translation of Formal Knowledge models to KBE application code with the traceability of various aspects of knowledge model.

Plate Assembly consists of plates joined through fasteners. Fastener assembly comprises of bolt, nut and washer. Various constraints, rules and design activities have been identified and represented as informal knowledge model. The entire model comprises of 3 entities, 18 activities, 11 rules and 9 constraints. Formal model has been created from the Informal model identifying various structural, functional, behavioral and representational entities. Also, design process model was created comprising of design activities, related rules and also linking to relevant structural entities. Since the focus of this paper is only translation of the Formal knowledge model to application code, all the details of the plate assembly knowledge model is not presented.





Bolted Plate Assembly

Bolted Plate Assembly Design Tool

Plate Dimension (mm)

Length	150
Width	100
Thickness	2

Material Allowable (Yield Strength) N/mm2

Plate	400
Bolt	500

Load (N)

Load on Plate	250000

Results (Output)

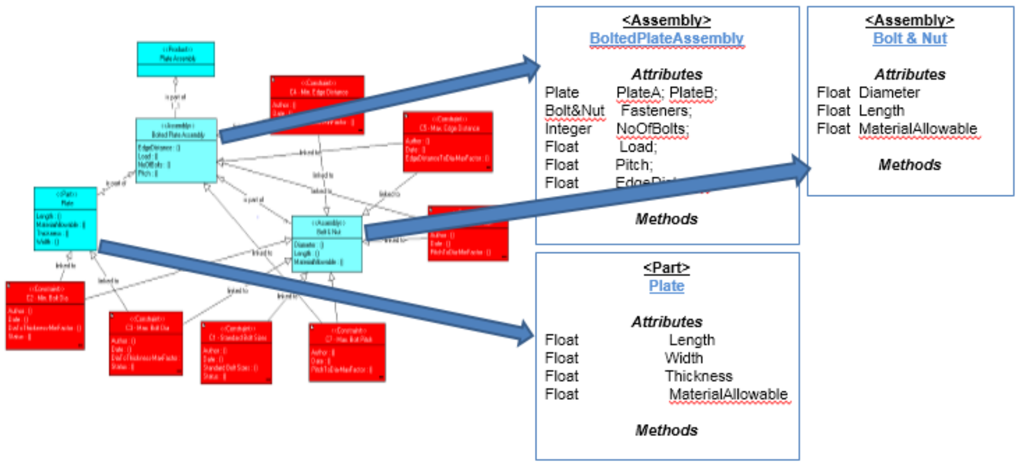
No of Bolts Required	
Bolt Diameter	
Pitch	
Edge Distance	

Message ...

Fig. 1: Plate assembly example.

Fig. 2: Plate assembly example UI.

The following illustrates the translation of the Informal and Formal models into software code.

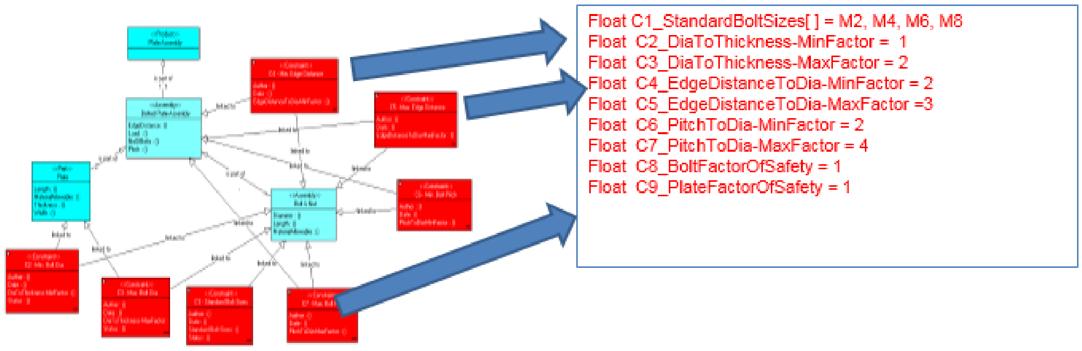


```
<Assembly>
BoltedPlateAssembly

Attributes
Plate       PlateA; PlateB;
Bolt&Nut    Fasteners;
Integer     NoOfBolts;
Float       Load;
Float       Pitch;
Float       EdgeDi

Methods
```

```
<Assembly>
Bolt & Nut

Attributes
Float  Diameter
Float  Length
Float  MaterialAllowable

Methods
```

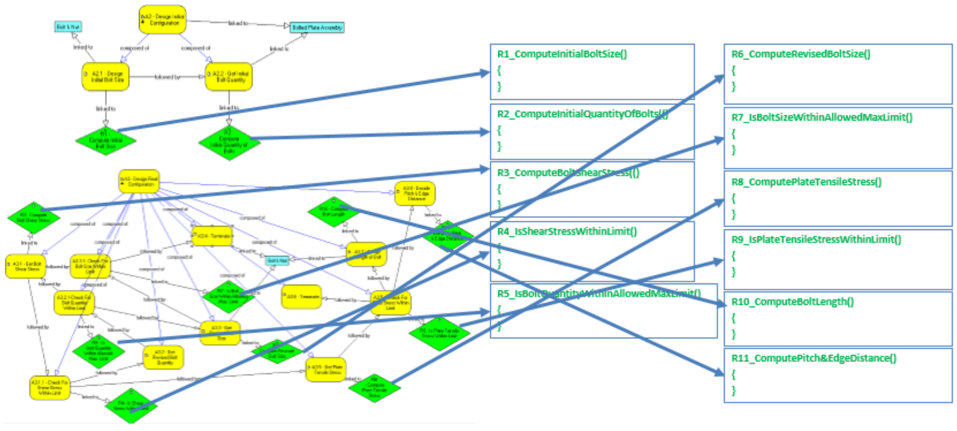
```
<Part>
Plate

Attributes
Float       Length
Float       Width
Float       Thickness
Float       MaterialAllowable

Methods
```

Fig. 3: Translation of product model – structure to class/object definitions.



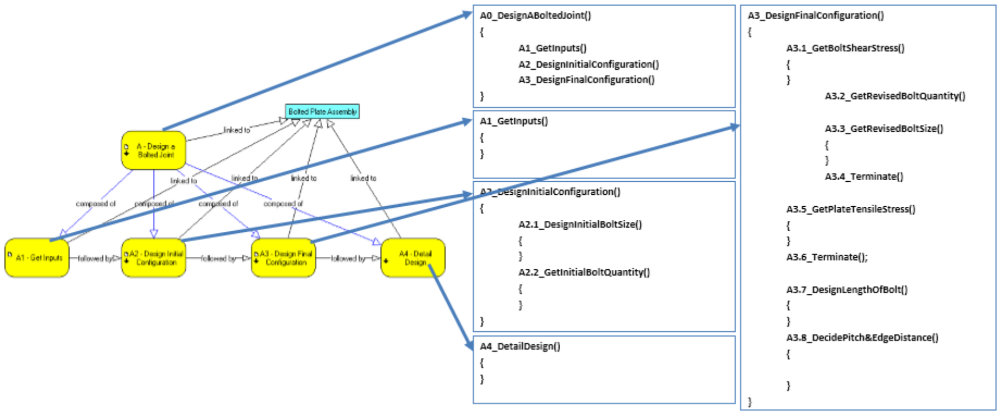
```
Float C1_StandardBoltSizes[ ] = M2, M4, M6, M8
Float  C2_DiaToThickness-MinFactor =  1
Float  C3_DiaToThickness-MaxFactor = 2
Float  C4_EdgeDistanceToDia-MinFactor = 2
Float  C5_EdgeDistanceToDia-MaxFactor =3
Float  C6_PitchToDia-MinFactor = 2
Float  C7_PitchToDia-MaxFactor = 4
Float  C8_BoltFactorOfSafety = 1
Float  C9_PlateFactorOfSafety = 1
```

Fig. 4: Translation of product model constraints into global constants.



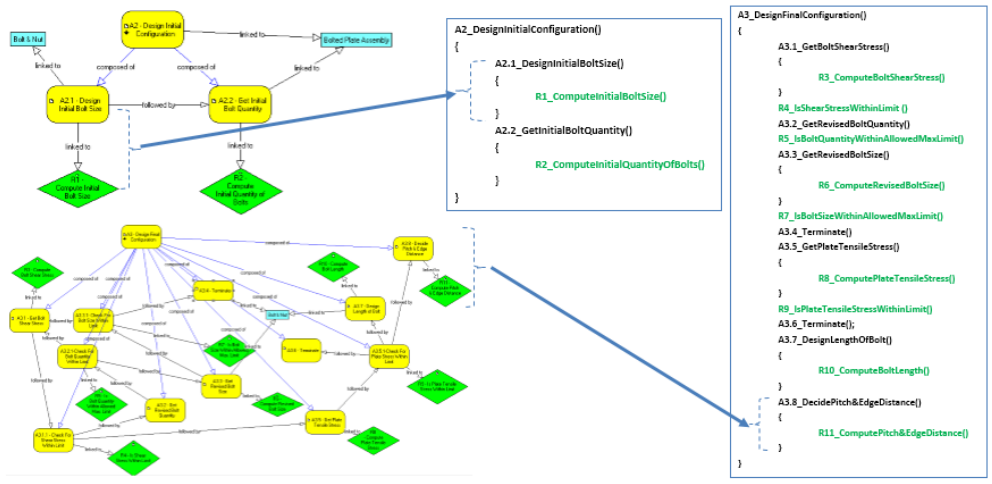
```
R1_ComputeInitialBoltSize()
{
}

R2_ComputeInitialQuantityOfBolts()
{
}

R3_ComputeBoltShearStress()
{
}

R4_IsShearStressWithinLimit()
{
}

R5_IsBoltQuantityWithinAllowedMaxLimit()
{
}

R6_ComputeRevisedBoltSize()
{
}

R7_IsBoltSizeWithinAllowedMaxLimit()
{
}

R8_ComputePlateTensileStress()
{
}

R9_IsPlateTensileStressWithinLimit()
{
}

R10_ComputeBoltLength()
{
}

R11_ComputePitch&EdgeDistance()
{
}
```

Fig. 5: Translation of design process model rules into global functions.


Bolted Plate Assembly
A - Design a Bolted Joint
linked to
composed of
A1 - Get Inputs
followed by
A2 - Design Initial Configuration
A3 - Design Final Configuration
A4 - Detail Design

```
A0_DesignABoltedJoint()
{
        A1_GetInputs()
        A2_DesignInitialConfiguration()
        A3_DesignFinalConfiguration()
}

A1_GetInputs()
{
}

A2_DesignInitialConfiguration()
{
        A2.1_DesignInitialBoltSize()
        {
        }
        A2.2_GetInitialBoltQuantity()
        {
        }
}

A4_DetailDesign()
{
}

A3_DesignFinalConfiguration()
{
        A3.1_GetBoltShearStress()
        {
        }
                A3.2_GetRevisedBoltQuantity()

                A3.3_GetRevisedBoltSize()
                {
                }
                A3.4_Terminate()

        A3.5_GetPlateTensileStress()
        {
        }
        A3.6_Terminate();

        A3.7_DesignLengthOfBolt()
        {
        }
        A3.8_DecidePitch&EdgeDistance()
        {

        }
}
```

Fig. 6: Translation of design process model activities into methods.


Bolt & Nut
A2 - Design Initial Configuration
Bolted Plate Assembly
linked to
composed of
A2.1 - Design Initial Bolt Size
followed by
A2.2 - Get Initial Bolt Quantity
R1 - Compute Initial Bolt Size
R2 - Compute Initial Quantity of Bolts

```
A2_DesignInitialConfiguration()
{
        A2.1_DesignInitialBoltSize()
        {
                R1_ComputeInitialBoltSize()
        }
        A2.2_GetInitialBoltQuantity()
        {
                R2_ComputeInitialQuantityOfBolts()
        }
}

A3_DesignFinalConfiguration()
{
        A3.1_GetBoltShearStress()
        {
                R3_ComputeBoltShearStress()
        }
        R4_IsShearStressWithinLimit ()
        A3.2_GetRevisedBoltQuantity()
        R5_IsBoltQuantityWithinAllowedMaxLimit()
        A3.3_GetRevisedBoltSize()
        {
                R6_ComputeRevisedBoltSize()
        }
        R7_IsBoltSizeWithinAllowedMaxLimit()
        A3.4_Terminate()
        A3.5_GetPlateTensileStress()
        {
                R8_ComputePlateTensileStress()
        }
        R9_IsPlateTensileStressWithinLimit()
        A3.6_Terminate();
        A3.7_DesignLengthOfBolt()
        {
                R10_ComputeBoltLength()
        }
        A3.8_DecidePitch&EdgeDistance()
        {
                R11_ComputePitch&EdgeDistance()
        }
}
```

Fig. 7: Translation of design process model – connecting methods and rule functions.

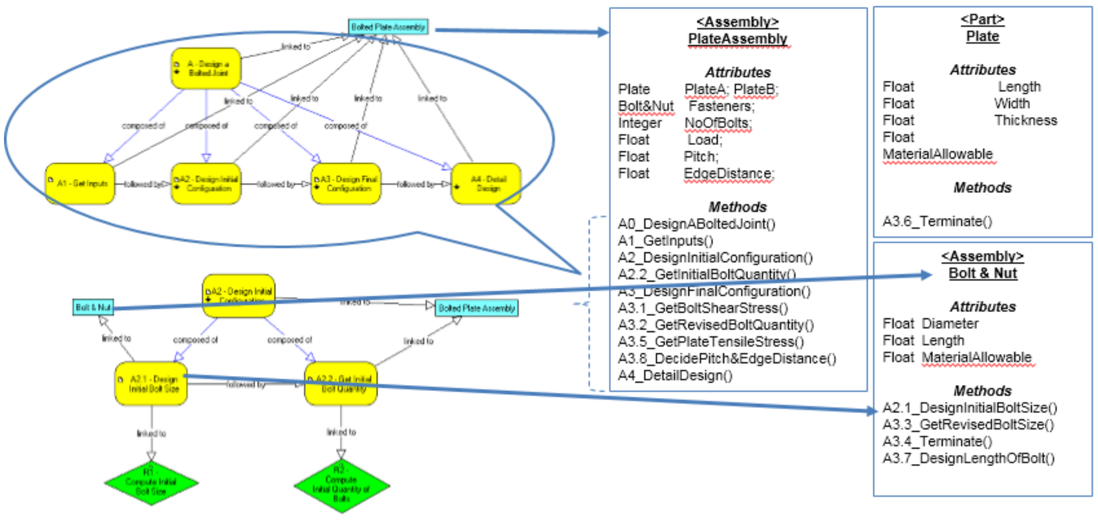

Bolted Plate Assembly
A - Design a Bolted Joint
A1 - Get Inputs
A2 - Design Initial Configuration
A3 - Design Final Configuration
A4 - Detail Design
linked to
composed of
followed by
Bolt & Nut
A2.1 - Design Initial Bolt Size
A2.2 - Get Initial Bolt Quantity
R1 - Compute Initial Bolt Size
R2 - Compute Initial Quantity of Bolts
<Assembly>
PlateAssembly
Attributes
Plate PlateA; PlateB;
Bolt&Nut Fasteners;
Integer NoOfBolts;
Float Load;
Float Pitch;
Float EdgeDistance;
Methods
A0_DesignABoltedJoint()
A1_GetInputs()
A2_DesignInitialConfiguration()
A2.2_GetInitialBoltQuantity()
A3_DesignFinalConfiguration()
A3.1_GetBoltShearStress()
A3.2_GetRevisedBoltQuantity()
A3.5_GetPlateTensileStress()
A3.8_DecidePitch&EdgeDistance()
A4_DetailDesign()
<Part>
Plate
Attributes
Float Length
Float Width
Float Thickness
Float MaterialAllowable
Methods
A3.6_Terminate()
<Assembly>
Bolt & Nut
Attributes
Float Diameter
Float Length
Float MaterialAllowable
Methods
A2.1_DesignInitialBoltSize()
A3.3_GetRevisedBoltSize()
A3.4_Terminate()
A3.7_DesignLengthOfBolt()

Fig. 8: Translation of design process model – methods to classes.

Conclusions:

In this paper, we proposed a methodology to translate MOKA based knowledge model into a software model and generic application code structure. The proposed methodology provided one-to-one mapping between requirements and application code thereby ensuring traceability between them. The methodology has been illustrated with an example of design of a bolted plate assembly model.

Acknowledgements:

The authors would like to acknowledge Dr. Ravi Kumar G. V. V. for the review, support and guidance provided in carrying out this work. We also thank Al Sameer Nujumuddin for his support in this work to create the application code for the illustrated example. We also would like to thank E.Krishna Kumar and Virendra Raghavendra Wadekar for their review and input on this paper.

References:

[1] Emberey, C.L.; Milton, N.R.: Application of Knowledge Engineering Methodologies to Support Engineering Design Application development in Aerospace, American Institute of Aeronautics and Astronautics, 2007, http://dx.doi.org/10.2514/6.2007-7708

[2] Lohith, M.L.; Prasanna, L.; Holla, D. V.: Translating MOKA Based Knowledge Models into a Generative CAD Model in CATIA V5 Using Knowledge ware, July 23, 2013, International Conference on Modeling, Simulation and Visualization Methods (MSV 13), Las Vegas, Nevada, USA.

[3] Brimble, R.; Sellini, F.: The MOKA Modeling Language, Knowledge Engineering and Knowledge Management Methods, Models, and Tools, Lecture Notes in Computer Science Volume 1937, 2000, 49-56.

[4] Wojciech, S.: Application of MOKA methodology in generative model creation using CATIA, Engineering Applications of Artificial Intelligence, 20, 2007, 677–690. http://dx.doi.org/10.1016/j.engappai.2006.11.019

[5] Stokes, M.: Managing Engineering Knowledge, MOKA: Methodology and Tools Oriented to Knowledge Based Engineering Applications, Professional Engineering Publishing Ltd, London, United Kingdom, 2001.