COMPUTER-AIDED DESIGN & APPLICATIONS, 2016
VOL. 13, NO. 4, 471–483
http://dx.doi.org/10.1080/16864360.2015.1131540

Multi-CAD approach for knowledge-based design methods

Markus Salchner[1], Severin Stadler[1], Mario Hirz[1], Johannes Mayr[2] and Jonathan Ameye[2]

[1]Graz University of Technology, Austria; [2]MAGNA STEYR Engineering AG & Co KG, Austria

ABSTRACT
The CAD-based design phase is characterized by a cooperation of manufacturer and supplier, which often use CAD software with different versions or even different vendors. In this context, the paper focusses on the development and application of knowledge-based engineering (KBE) within multi-CAD environment. Usually, knowledge-based design (KBD) methods are developed within and for specific CAD systems, respectively specific software configurations or releases. Engineering and component supplier companies are faced with the problem that car manufacturers (OEM) work with different CAD software solutions. A multi-CAD strategy is able to support the handling and organization of different CAD-related project environments, especially in view of knowledge-based and automated design applications, as well as data management. The presented approach provides a platform for the efficient development of KBD applications for multi-CAD environments.

KEYWORDS
Knowledge-Based Design; Design Automation; Multi-CAD

1. Introduction

During the past decades, the development period of a new car has been reduced from five years and more to about two years. These considerable time savings can be attributed to a continuous optimization and improvement of development processes. In this context, virtual development methods play a prominent role. Product Lifecycle Management (PLM) solutions are used for data organization, distribution and storage within different computer-aided design and simulation applications. This comprises computer-aided design (CAD), styling (CAS), manufacturing (CAM), as well as computational toolkits, e.g. multi-body simulation (MBS), computational fluid dynamics (CFD) or finite element method simulation (FEM). Furthermore, lifecycle analyses (LCA) can also be implemented in modern PLM systems. The broad field of computational design and engineering disciplines within automotive development requires a strong interaction in view of efficient software applications. Especially the CAD-based design phase is characterized by a cooperation of manufacturer and supplier, which often operate on various CAD systems with different versions or even different vendors. In this context, the paper focusses on the development and application of knowledge-based engineering (KBE) within multi-CAD environment. KBE deals with the storage and reuse of knowledge in product development processes [16]. A design-oriented specialization of KBE is known as knowledge-based design (KBD) [6], [8].

Usually, KBD methods are developed within and for specific CAD systems, respectively specific software configurations or releases [3], [12]. Especially in case of problem-oriented knowledge-based applications, the compatibility to a variety of CAD software is restricted due to a high level of programmed customized features. Engineering and component supplier companies are faced with the problem that car manufacturers (OEM) apply different design methods based on different CAD software solutions as can be seen in Figure 1. In this example, an automotive supplier has to support different CAD systems, including numerous OEM-related methodical requirements. This results in a need of high sophisticated levels of knowledge in different CAD-systems including a complex application of knowledge-based design tools. A multi-CAD strategy, as introduced in the present paper, is able to support the handling and organization of different CAD-related project environments, especially in view of knowledge-based and automated design applications, as well as data management.

2. Knowledge-based design methods

The first paragraphs give an outline on the KBE research area and the state of the art methodology. KBD methods are a specialization of KBE with focus on the design process. An overview on of KBE evolution, approaches, methodologies and future trends including the notion of KBE is treated in Reddy et al. [16], who presented

CONTACT Markus Salchner markus.salchner@tugraz.at

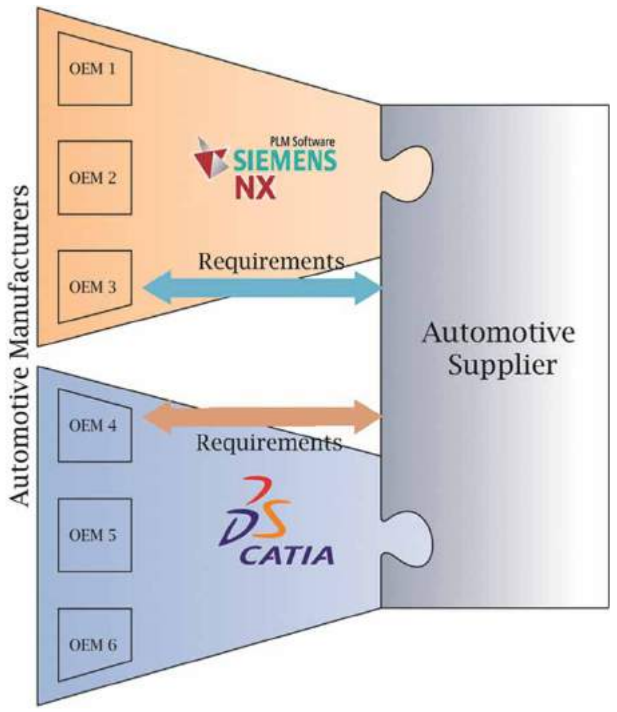


Figure 1. Automotive supplier challenge: OEMs with different CAD environments.

an approach "to carrying out the research to develop a web-oriented function based parametric modeling system using KBE" and pointed the high future potential of Case-based KBE application [9].

There is no specific definition of KBE, in general, knowledge-based methods are used to capture and reuse engineering knowledge. The slightly more detailed notion from Stokes "use of advanced software technologies to capture and reuse product and process knowledge in an integrated way" is often cited [20]. The enormous potential of KBD regarding cost and time reduction and the simultaneous quality and efficiency improvement leads to a continuous development and research in this area [16], [10], [11], [23].

Some shortcomings because of considerable effort for maintenance and management of several KBD applications within CAD environment or platform independent has been also been treaded in several literature. Sanya and Shehaba [17] identified the need "to standardize the internal knowledge representation of KBE applications and interfaces with product development solutions (i.e. PLM, CAx)". Therefore, an ontology framework for the aerospace sector was presented. The main idea was to decouple the knowledge model from the operating CAD system using standardized interfaces like Extensible Markup Language (XML), Unified Modelling Language (UML) and Ontology Web Language (OWL). The OWL does not allow the implementation of rule-based functionalities, which requires the application of another language; the Semantic Web Rule Language (SWRL). Further mentioned languages and interfaces are Common Logic Interchange Format (CLIF), Operational Conceptual Modelling Language (OCML) and Java Expert System Shell (JESS). To couple this knowledge system with suitable user interfaces and visualization tools, further APIs are required. In that approach, the transfer to an IDE Platform – Eclipse [5] – is carried out by the Jena API and OWL-API. Finally, the geometrical visualization is performed by a standardized 3D API, which is based on Java. The implementation of CAD related functionalities is not explicit mention in this work. Furthermore, the mix of different programming and modeling languages is quite challenging. Nevertheless, this paper presents a full platform independent approach [17].

A new, different approach, presented in this paper, uses a single programming – and modeling language and supports various IT solutions including CAD systems by different satellites (Figure 5). In contrast to Sanya and Shehaba [17] this proposed system is limited to Microsoft Windows platform [13], which does not constitute a significant disadvantage in the automotive sector.

Figure 2 illustrates a principle chart of the different KBD levels. The implementation and development of design methods can only be efficient and successful, if group-wide directives are clearly defined. This comprises a project independent overall process, design and automation directive. The workflow process including the release, change and quality management is in common regulated and handled by a product data management system, like Teamcenter [18]. The definition of part, respectively assembly design within a CAD system is different due to numerous design possibilities and working methods of the engineers. In addition, the inescapable verification is quite challenging. The definition and handling of required global directives for one single CAD system is state of the art and mostly supported by different integrated design-related checkers, like Q-Checker [22] for CATIA [4] or Check Mate in NX [18]. A list of necessary automation guidelines for the management of KBD is mentioned in Figure 2. This includes for example the definition of the development environment, programming language, structure, rollout method and further framework conditions.

The nethermost KBD methods, shown on the left side in Figure 2, are non-parametric models. This implies, that the engineer creates for example a specific CAD model with given dimensions, whereby a geometrical variation is not possible afterwards. In such a case, the complete model has to be built up all over again.

The next level is based on parametric-associative design methods, which correspond to the current state of

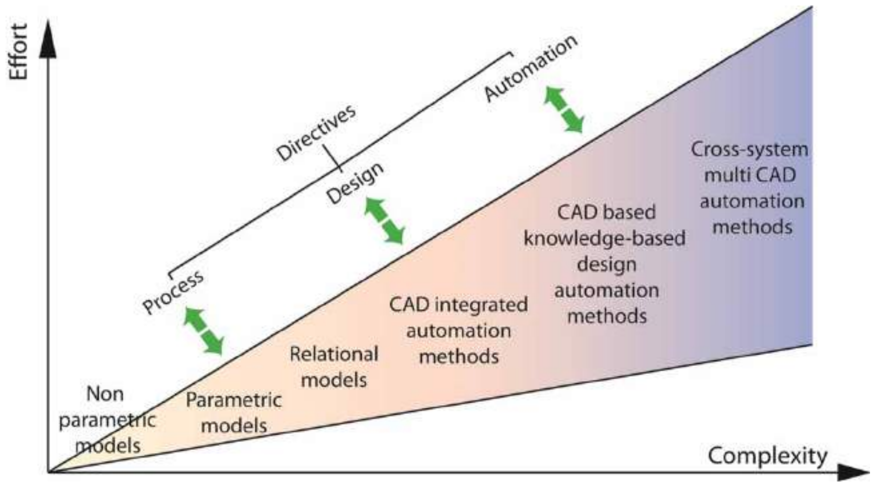

Effort
Complexity
Process
Directives
Design
Automation
Non parametric models
Parametric models
Relational models
CAD integrated automation methods
CAD based knowledge-based design automation methods
Cross-system multi CAD automation methods

Figure 2. Knowledge-based design levels, according to [8].

the art in automotive engineering processes. In this case, the exemplary CAD model is designed with variable values. These values can either be directly integrated in the geometry definition, like a length constraint in a sketch, or they can be stored in a feature definition itself. Another possibility is the definition of separate parameters, decoupled from the geometry, which are stored in the desired CAD component. This allows an efficient design adaption by changing the parameter values in the CAD system or within an external file. Parameters can also be used for the definition of characteristics such as materials, textures, tolerances and other relevant information [8].

Relational models define the third level in Figure 2, which are based on parametric models and include further functionalities. In common, an interaction between CAD elements, like sketches, features, bodies or even parts and assemblies, is carried out by use of rules, checks or loops. This method corresponds to the next higher level, the CAD integrated automation methods. In CATIA, relational design can be carried out without the use of any internal scripting or programming language. In NX, a definition of feature based rules requires the use of an integrated knowledge application, called Knowledge Fusion (KF), which requires basic scripting knowhow. The different levels and functionalities of integrated methods are explained in the following subsections.

CAD based automation methods mainly differ from integrated methods due to the use of external "integrated development environment" (IDE), like Visual Studio [14] or Eclipse [5]. This allows the development of external automation tools with

- customized graphical user interfaces (GUI),
- use or implementation of external applications (e.g. Excel, Word, etc.),
- development of programming library based on the object oriented programming (OOP) principle,
- CAD independent scripting language,
- consistent development environment and
- simultaneous application development.

Furthermore, development-related know-how is no longer stored in the CAD system itself, but in specific applications or even coupled databases and therefore improved accessible for other engineering disciplines and employees.

Commercial KBE software packages like KF [18], CATIA Knowledgeware [4], CADEC Works [1], KBEWorks [9] or TactonWorks Engineer [21] can be assigned to the category of CAD integrated automation methods. Therefore, these tools does not meet the requirements, and are not explained in detail.

The highest level of KBD methods is represented in cross-over methods and tools, which support different CAD solutions within the company. Process related know-how is completely decoupled from the CAD system and made for anyone to disposal. The initial development effort for such automation tools is quite high. Especially the definition of a comprehensive multi-CAD environment requires a detailed consideration of different aspects. One main problem to face is the data transfer via inconsistent application programming interfaces of the different CAD systems. It is a big challenge to serve all systems concerning design, knowledge-based methods and automation strategies. In this context, the present paper introduces a new approach for the efficient development of a multi-CAD KBD environment under consideration of the prior problem statements.

The presented approach is viewed from three different developer perspectives. The topmost developer, called KBD-Expert, is responsible for the design, maintenance and enhancement of the abstract multi-CAD-framework. This working range considers the mentioned overall directives and the principle layout of the KBD environment – see section 3. The next development level concentrates on the KBD-Engineer, who is responsible for the development of selected applications supporting design. The implemented functionalities strongly depend on the specific task and do not consequently require a multi-CAD realization in the early phase. Nevertheless, the KBD-Engineer is assisted by a superordinate framework – see section 4. The last but not least level, the CAD-User is supported by the different KBD tools in his work to raise productivity and quality of the entire design process. The KBD-User simultaneously is the person which consolidates tasks of daily design work for potentially new KBD-applications to be developed.

Figure 3 shows the different knowledge-based automation levels regarding complexity and functionality. The required experience for the common development of KBD tools is schematically illustrated, by line number one. Due to the implementation of the multi-CAD approach, the required programming effort of the KBD-Engineer can be dropped to a lower level – line two. Another advantage of the presented high level KBD approach can be seen in line three. Due to the use of application programming interfaces, process or application related know-how is no longer stored in the CAD system, but rather in the application or a centralized database itself. Thereby, a crossover knowledge distribution and management can be reached.

The group on the left side in Figure 3, called basic KBD tools, uses internal tools and functionalities of a CAD system. This includes parameters, formulas, spreadsheets, rules, checks, loops, templates and user defined features. Other tools of this group are CAD based recorders and integrated programming languages. The provided functionalities of these internal tools strongly depend on the CAD solution and the available license package. The next higher level uses the CAD based and integrated development environment. This group uses the CAD API and allows a partially decoupled KBD development – depending on the CAD system. Therefore, an effective multi-CAD strategy cannot be reached out with the mentioned tools. The development of an efficient overarching strategy can only be achieved by the highest level – a CAD independent development environment.

3. Multi-CAD approach

In contrast to the general understanding of a multi-CAD environment, the present strategy represents an unrestricted method. Functionalities in commercial available CAD systems are restricted to the provision of neutral geometry data interfaces for the exchange of CAD models, e.g. STEP, IGES, STL. These converters allow an exchange of geometry between different CAD formats, but there is no possibility to handle intelligent knowledge-based software modules, e.g. templates for automated geometry creation. The presented approach provides a smart development and implementation of multi-CAD capable knowledge-based design methods [4], [15], [18].

3.1. Development process

Figure 4 illustrates the developed process model for the layout of the novel strategy. The process is divided into three main layers. The iterative process covers both the horizontal – layer internal – , and vertical – across layers – interactions.

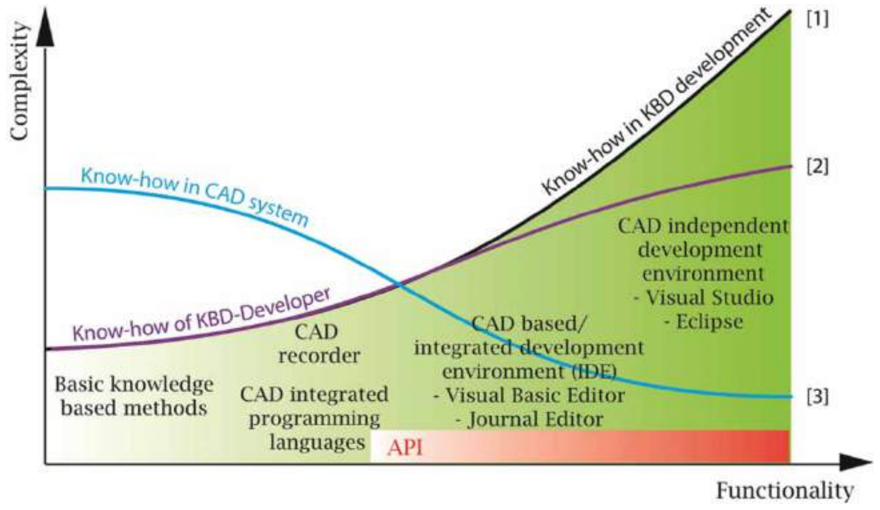

Complexity
Functionality
Know-how in KBD development
Know-how in CAD system
Know-how of KBD-Developer
[1]
[2]
[3]
CAD independent development environment
- Visual Studio
- Eclipse
CAD based/ integrated development environment (IDE)
- Visual Basic Editor
- Journal Editor
CAD recorder
CAD integrated programming languages
Basic knowledge based methods
API

Figure 3. Levels of KBD automation.

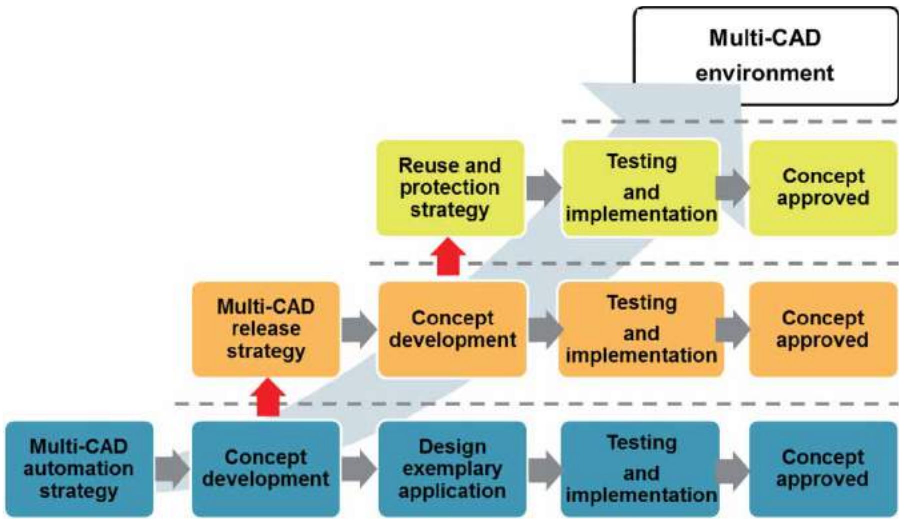

Multi-CAD environment
Reuse and protection strategy
Testing and implementation
Concept approved
Multi-CAD release strategy
Concept development
Testing and implementation
Concept approved
Multi-CAD automation strategy
Concept development
Design exemplary application
Testing and implementation
Concept approved

Figure 4. Development process of multi-CAD environment.

The first step includes the conceptual layout of a multi-CAD automation strategy, illustrated at the bottom line in Figure 4. By an exemplary application, the strategy has been used to integrate the CAD software packages CATIA V5 and Siemens NX; but the general approach may be suitable for other CAD software, too. This application states the basis for the subsequent development, implementation and testing procedure. Therefore, the exemplary application is designed to cover a wide range of automation methods, taking into account various namespaces, classes and functions of the provided programming interfaces. In case of Siemens NX, there are four main libraries available, whereby the CATIA API is structured more widespread (more than eighty libraries). After the validation of this concept, the next development step focuses on a multi-CAD release strategy – layer two. This treats the application compatibility for one CAD system including its chronological releases. After the testing and implementation cycle of this layer, both, the multi-CAD - and the multi-Release concept are transferred to the final development layer. This development level considers a strategy for the reuse and protection of the overall method. This entire development process results in a final multi-CAD environment, as illustrated in the top layer.

3.2. Overview of the Multi-CAD-framework

The iterative development process, explained in the previous section, consequently leads to a multi-CAD-Framework. This subsection explains the developed framework and the programming libraries in view of a KBD-Expert. The framework is based on the IDE of Visual Studio and uses the programming language Visual Basic [14]. The decision criteria for this development environment was mentioned in the previous section and the criteria for the language-related decision can be found in section 4.

Figure 5 illustrates the environment of the present approach including the internal main modules and their docked satellites. The abstract methods, which cover universal functions, are stored inside the framework. This includes “CAD base”, “Utilities”, “Infrastructure” and “Settings” classes. These base classes are managed by the KBD-Expert and define fundamental functionalities of the docked satellites. The development and implementation of these objects are not explained in detail in this paper, except the “CAD base” class. This superordinate functional definition allows docking of different applications in an efficient way and grants the consequent enhancement of the framework. The KBD-Engineer implements the framework in the development process of a new KBD-Tool. The developer can access – on require – all implemented methods via the novel framework. The final KBD-Tool has to be analyzed by the KBD-Expert regarding their implemented functionalities. In the next step, the KBD-Expert performs an abstraction of the functionalities and defines the adequate implementation of new base functions. Thus, the base functionality rises with each single application and expands the predefined functions for future projects.

The CAD related assemblies are illustrated at the bottom of Figure 5, by Siemens’ NX, Dassault’s CATIA V5 and any additional CAD system – like PTC Creo Elements [15]. Furthermore, third party applications like Excel, Acrobat or Matlab are also supported by the interface. Different PDM systems, databases and finally single specialized KBD tools are optional satellites, which can be implemented in the multi-CAD-Framework. The predefined overall structure grants the integrity and the

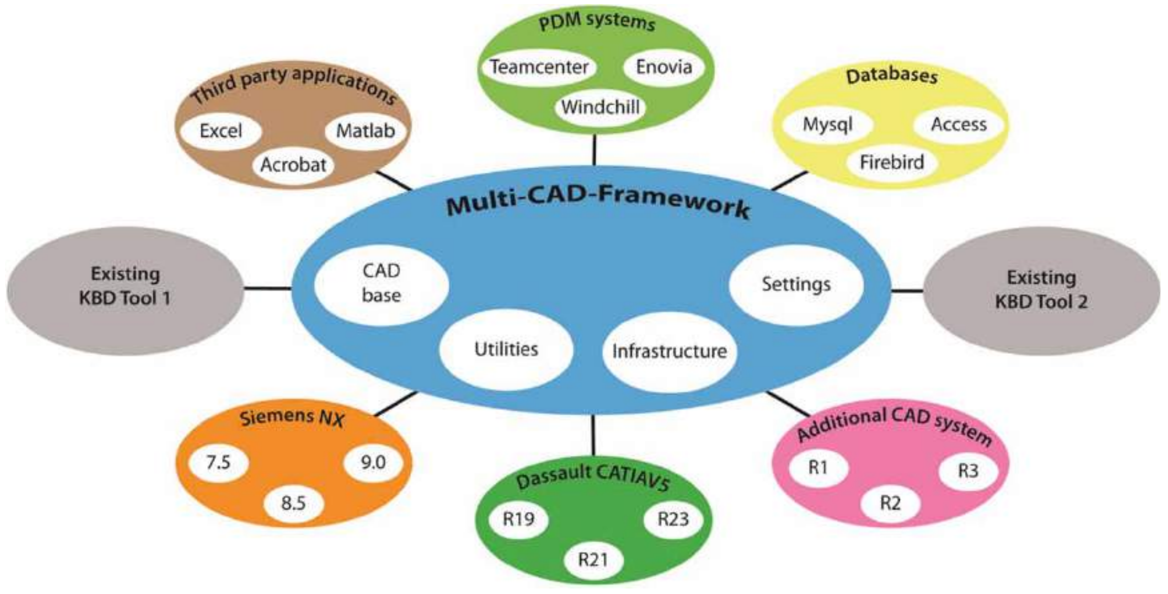


Figure 5. Multi-CAD-Framework.

intuitive working principle for all KBD-Engineers. Furthermore, the environment includes a defined strategy for serviceability, reuse, as well as protection and licensing aspects.

The database interface gives the developer the ability to use different database functionalities. This includes the ease creation of user defined local or server databases and allows quick storage and reuse of application related information. Furthermore, database-applications provide useful functionalities for listing, sorting, or searching of data and enable a dynamic application layout.

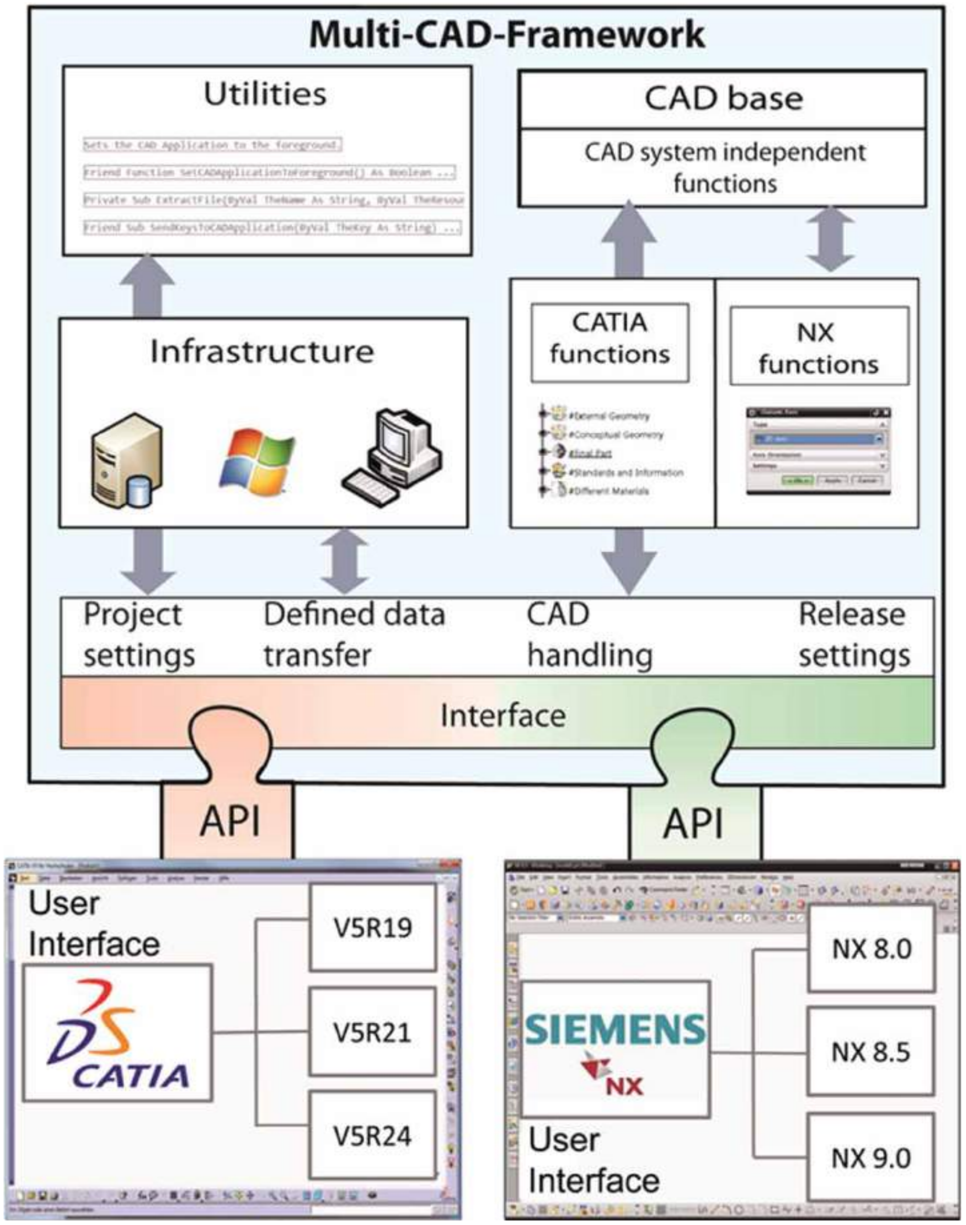
The overall implementation expands the function scope enormously and enables an efficient interaction between CAD and third party tools. Exemplary, interdisciplinary KBD tools are geometry-based analysis and optimization routines, e.g. specific suspension optimization. For example, this enables a transfer of main suspension points, which are defined and stored in the CAD system, to a Matlab optimization routine. The routine performs an optimization with regard to occurring forces, the maximum possible steering angle or dive angle. The results are displayed in the CAD system and the engineer can adapt the design model. This application is exemplary illustrated as KBD-Tool 1, whereby the KBD-Tool 2 represents a separate exemplary application, shown in Figure 5. Each implemented application expands the framework and enables a continuous and efficient enhancement.

3.3. Realization of Multi-CAD-framework

Figure 6 gives a more detailed overview of the developed environment, concentrating on the multi-CAD environment. In contrast to Figure 5, this representation contains no global assignment, but allows a more detailed explanation of the developed environment.

At the bottom of Figure 6, CAD systems like CATIA [4] and NX [18] are exemplarily illustrated with different releases. The developed multi-CAD interface implements the inconsistent application programming interfaces of the diversified CAD systems, see sub section 4.3. The interface definition is stored in the multi-CAD-Framework and contains abstract methods for the implementation of CAD systems, called “CAD base”. Therefore, each CAD system and release fulfills the same basic functionalities like create part, create picture, get properties or close CAD application. The interface allows a simultaneous, sequential or single execution of the developed KBD tool within the different CAD systems. In addition, different project and release settings are supported.

The “Infrastructure” area covers basic functions and methods of the operating system Microsoft Windows [13]. Exemplary elements are folder and data management. Furthermore, it offers functionalities that allow the handling of all Windows based applications. This includes maximize, minimize or hide functions, etc., and an integrated docking class. This method allows the implementation / docking of any application into the developed graphical user interface. The user interface covers all required information, and the KBD-User is not forced to switch between different windows. The standardized layout of the GUI and the mentioned functionality is also implemented in this library and supports the KBD-User and KBD-Engineer equally.



```
Sets the CAD Application to the foreground.
Friend Function SetCADApplicationToForeground() As Boolean ...
Private Sub ExtractFile(ByVal TheName As String, ByVal TheResou
Friend Sub SendKeysToCADApplication(ByVal TheKey As String) ...
```

Figure 6. Multi-CAD automation environment.

The "CAD base" class initializes all CAD related developed functions and methods. This library does not implement the functionality itself, but it defines them. Each single implemented CAD system is a derived class and is forced to implement the defined functions. Relatively simple CAD basic functions, like the creation of a point, a line or a feature, are not implemented in this level, because these CAD related functions are supported by the API and it is not useful to reproduce all of them. In contrast, it includes a collection of commonly used and know-how based functions. On basis of an object oriented programming technique, it is possible to force the developer to implement the designed function for all required CAD solutions. One example of application is a function for capturing of images. The code for the image creation is not challenging, but to adjust a whole environment to the defined settings can be very complex. The implemented function performs the following actions in addition to the passed parameters: Perspective, position, zoom, visibility of constraints, annotations, treeview, components, change the color of parts, surfaces or even the entire assembly.

The explained framework can be implemented and used as references by each KBD-Engineer in the development environment of Visual Studio. The depicted satellites in Figure 5 are standalone libraries, whereby an individual implementation is enabled. Some applications do not need a database - or the CATIA interface, and therefore an implementation of these libraries is not effective. The current framework comprises the following main libraries:

- Multi-CAD-Framework
- Dassault CATIAV5
- Siemens NX
- Third party application – e.g. Excel
- Firebird databases [7]

Additional functionalities with numerous overloading methods are stored in a separate snippet library. They provide a large amount of different input parameters regarding to their use. An exemplary CAD related function is the "CreatePoint" method. The design of a point can be done in various ways - by coordinates, on a line, a surface, a body or as a center point of a circle. Twenty overloading methods in NX are available, whereby each of them uses different NX related objects, which in turn are overloaded. Other exemplary methods are folder selection, file selection – or save methods, which strongly depend on the current requirements. The snippets are implemented in the graphical interface of Visual Studio and can be easily accessed. The snippets are based on XML and copy the stored source code elements into the current project. The KBD-Engineer can customize the scripting lines to the current requirements. The developed snippet library uses the same structure as the framework library and grants an intuitive use.

4. Multi-CAD developer

The previous section explained the entire framework of the multi-CAD approach. With the target of an effective use, the framework design was made with focus on an intuitive usage and nomenclature. To ensure stability and quality of the framework, some programming rules have to be considered. Furthermore, the implementation effort for the KBD-Expert, mentioned in sub section 3.2, can be reduced to a minimum. This section treats the defined development steps for the design of new multi-CAD-Framework applications.

4.1. Programming directives

This subsection focuses on programming fundamentals, which are required for the understanding and implementation of the multi-CAD approach. Common KBD automation tools are developed for one unique task by one single developer [3], [10–12], [16]. To keep it simple, these tools are often coded in the so-called "Spaghetti code". This means, that the code is unstructured and the application of user defined objects, classes or instances is limited. Furthermore, access modifiers, e.g. public, private, protected are not used consequently and the code flow cannot be traced - it is like a bowl of spaghetti. This way of coding seems to be the easiest way of programming, but with increasing complexity and multiple developers, it causes a lot of problems. Consequently, an implementation in the explained framework is impossible, which requires the application of the so-called object oriented programming (OOP) approach [14].

The principle of OOP bases on the definition of user defined objects and a structured code flow. The objects contain different fields, methods and properties, as shown in Figure 7. The field group contains user defined variables, which should be only used internal. The properties provide external functionalities, which can be accessed from outside. The mentioned access modifiers are indicated as small symbols beside the illustrated elements. These modifiers are important for a structured code flow and allow the protection of selected attributes.

This means, that specific attributes can only be changed under certain conditions in contrast to public attributes. The main principle structure is explained by a CAD based example. The base class – "CADApplication" has two derived classes, "CADNX" and "CADCATIAV5", as shown in Figure 7. The base class defines all common functionalities, where the program code does not differ. The modifier "MustInherit" defines, that the creation of a direct instance from this class is restricted. The illustrated "CADApplication" class contains two fields, two properties and five methods.

The "SetCADApplicationToForeground" and the constructor sub "New" are base methods, which are not related to a specific CAD system. The source code of these functions is implemented directly in the "CADApplication" class. The two remaining methods "CADApplicationReady" and "OpenPart" are CAD system related functions and separately coded in the derived classes. They bear the modifier "MustOverride", which means that all derived classes have to implement them. Of course, derived classes can contain additional properties and methods, like the "CADNX" class.

The "CADInfos" class is a user defined class with CAD related attributes like "Version", "StartUpPath" or "Type", depicted on the right in Figure 7. The attributes are declared as public elements; there is no sufficient protection and secure access management. Under consideration that each "CADApplication" instance comprises the private variable "m_CADInfos", which represents an instance of the "CADInfos" class, the related information cannot be accessed. The "CADInfos" property is declared as "ReadOnly", and so the relevant "CADInfos" can only

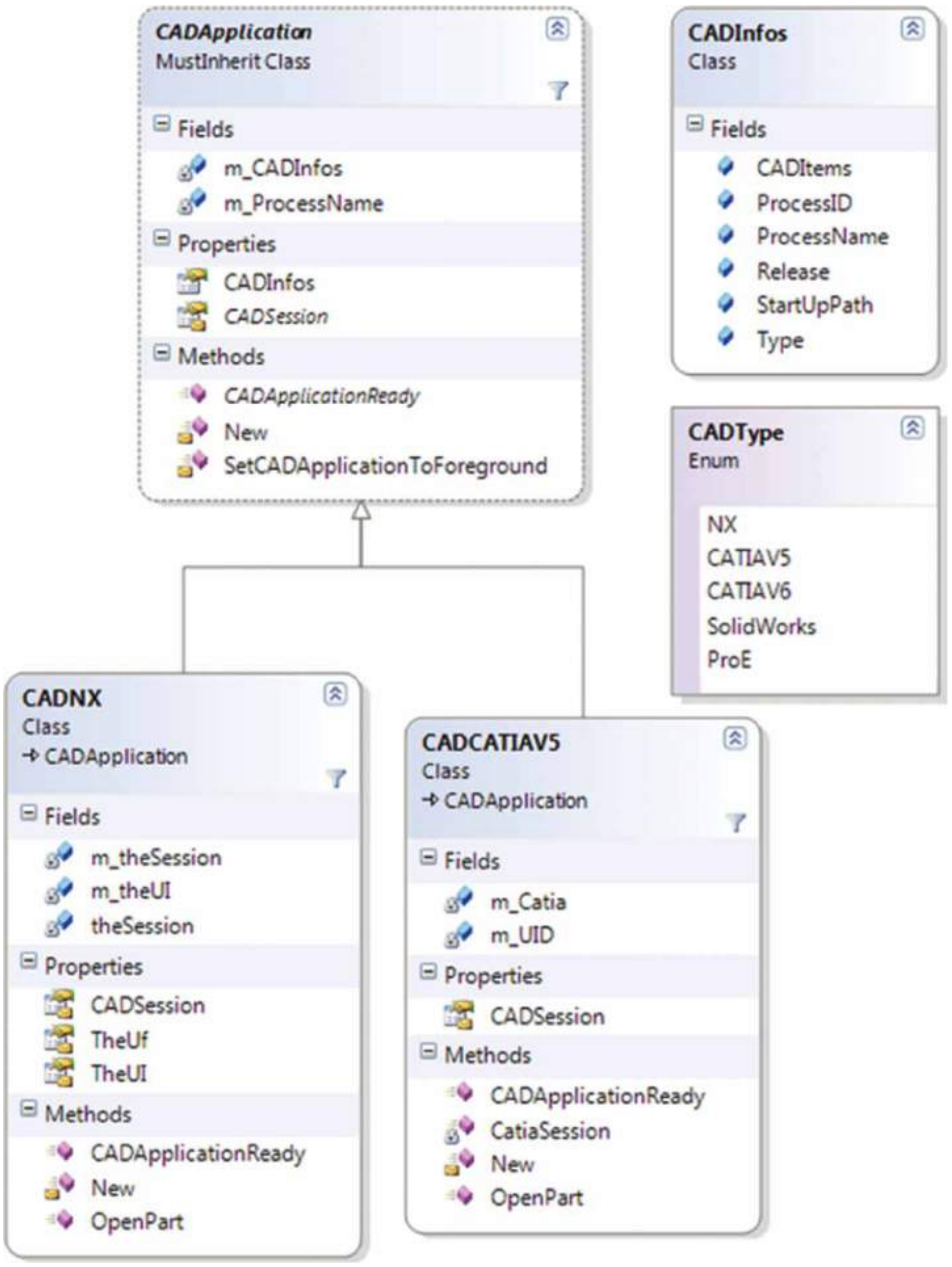


Figure 7. Exemplary structure of the "CADApplication" class.

be adapted inside the "CADApplication" class. The initial definition of these variables is done by the initialization process of a new "CADCATIAV5" or "CADNX" object.

Considering the multi-CAD-Framework structure, explained in section 3, the "CADApplication" class is stored in the framework itself, whereby the derived classes are located in the selected CAD satellites. To grant, that each KBD-Engineer receives the required information, XML comments for the documentation of each single property, function or sub, as shown in the Figure 8 are mandatory.

```
''' <summary>
''' Check if the CAD application is ready.
''' </summary
''' <returns>True or False</returns>
''' <remarks>None</remarks>
Public MustOverride Function CADApplicationReady() As Boolean
```

Figure 8. Exemplary XML comment.

4.2. Structured programming

Based on the mentioned programming fundamentals, this section concentrates on the developed structural programming guideline, which is an essential part of the multi-CAD strategy. Each multi-CAD automation application should follow the main rules to grant:

- Comprehensibility for other developers
- Simplified adaption and enhancement
- Release handling
- Enable the multi-CAD approach
- Handle API changes

Figure 9 illustrates the principle structure of a multi-CAD application. The application consists of at least two different projects, the Main - and the Application Project, whereby the Application Project is referenced to the Main Project. The latter can access the defined methods and properties of the Application Project. Additional feasible projects could be databases or third party projects. The assembly class in the Main Project is responsible for all GUI related calls, events and also for the separation of different CAD systems.

This distinction is carried out by the instantiation of different application assemblies. For example to support CATIA and NX within this application there has to be one "NXAssembly" and one "CATIAAssembly" instance. These derived classes inherit the base class "ApplicationAssembly" and follow the "CADApplication" - principle, shown in the previous section. The required multi-CAD initialization functions are called from this class. The graphical user interfaces often change due to the development process or in case of different customers. Therefore, a codding in GUI elements should be avoided. It is much more efficient to utilize user defined events and handle them in the assembly class. Last but not least, the Main Project handles the exception events, the custom help file and optional license functions. This structure allows an eased adaption and enhancement of main functionalities of the program by any KBD-Engineer. Different application releases for various customers with different functionalities can be compiled in an efficient way.

The main development rules for a multi-CAD application are:

(1) At least two separate projects – Main and Application
(2) No CAD or know-how related source code in the Main Project

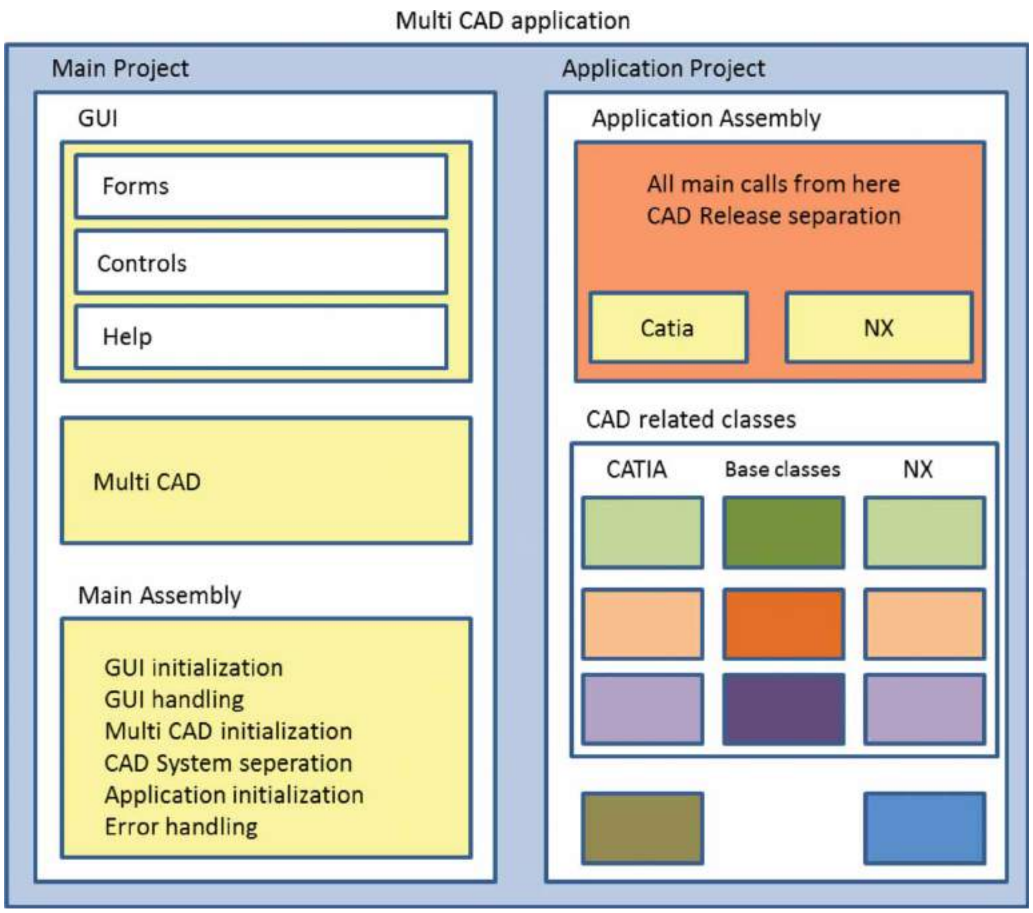


Figure 9. Structured programming layout for multi-CAD applications.

(3) No code elements in the GUI
(4) CAD system separation in the Main Project
(5) All calls from the project related assembly class
(6) CAD release separation in Application Assembly class

The Application Project contains all required functions for the developed application itself. This includes universal base classes and derived CAD related classes, as shown in Figure 9. Of course, this project can also include CAD independent classes for general functionalities, like the creation of folders, files or other third party applications.

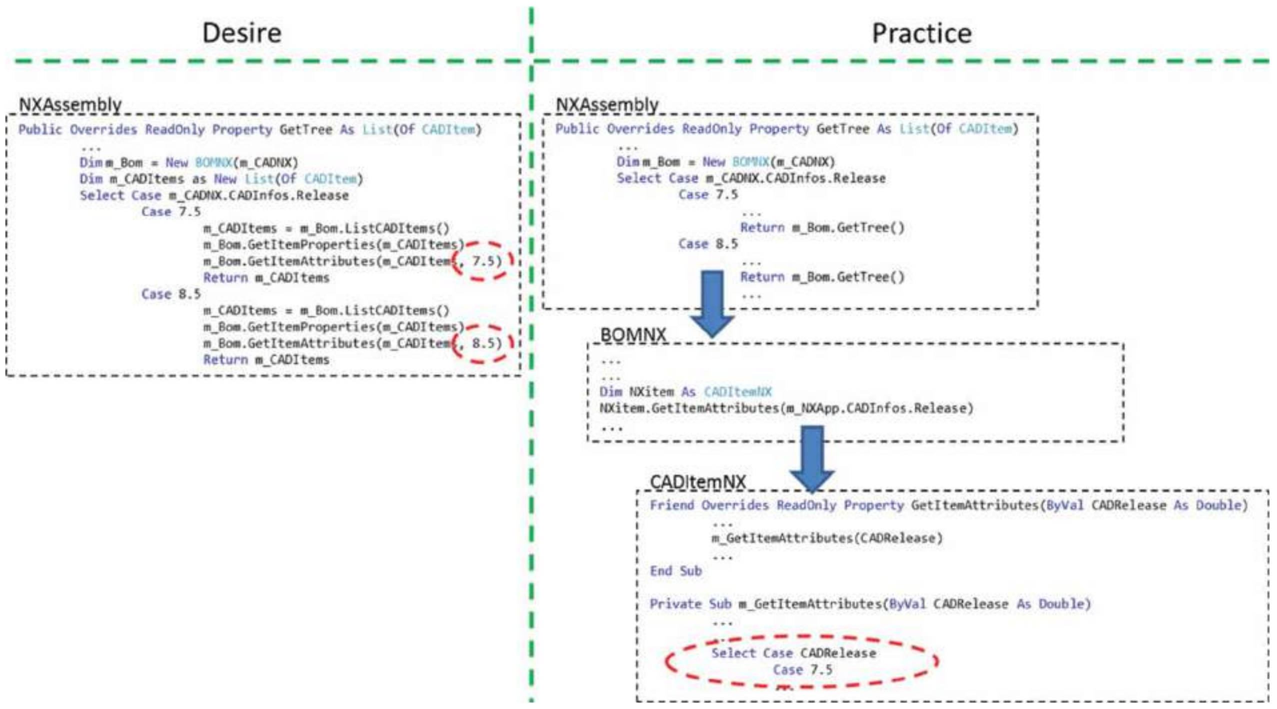
It is up to the developer to decide when a separate project (e.g. third party project) is created or not. In general, it is advisable to create a new project as soon as an additional API is used (e.g. Excel). The code separation for different CAD releases, caused by API changes, should be handled in the concerning CAD assembly, as illustrated in 10.

The Figure 10 shows the three different classes –“NXAssembly”, “BOMNX” and “CADItemNX”- including some general code elements. The “GetTree” property of the “NXAssembly” is the starting point, where a separation of NX releases is carried out – NX 7.5 and NX 8.5. The optimum source code on the left side would call the release dependent methods from here, and also release independent functions, like “ListCADItems”. Unfortunately, this represents a major effort in the practical development. The application is in common developed for one single release, and therefore the code structure is defined in a logical way – as shown on the right side. The implementation of further releases is added as needed later. From this stage of development, the KBD-Engineer is confronted with possible API differences within the CAD releases. In the example in Figure 10, the code differs in a deep layer of the program structure. The release separation is required in the “m_GetItemAttributes” sub within the “CADItemNX” class – at the bottom. The developer has to change the entire structure and methods to perform a release distinction in the “NXAssembly”. Therefore, identical basic code elements have to be duplicated for small differences within some code lines. This is not purposeful, produces unnecessary code and the effort increases in maintenance, modifications and handling of programming errors. Experienced developers can estimate the differences already in the run and thus implement an appropriate structure.

To avoid the mentioned structural changes in case of small API differences, but to allow later code tracking, a keyword, e.g. “CADRelease”, is defined. This keyword is used as function or property argument for each single method, which requires a separation of CAD releases.

4.3. CAD Automation interfaces

APIs are based on core algorithms of the CAD system and allow the definition and development of KBD automation



```
Public Overrides ReadOnly Property GetTree As List(Of CADItem)
        ...
        Dim m_Bom = New BOMNX(m_CADNX)
        Dim m_CADItems as New List(Of CADItem)
        Select Case m_CADNX.CADInfos.Release
                Case 7.5
                        m_CADItems = m_Bom.ListCADItems()
                        m_Bom.GetItemProperties(m_CADItems)
                        m_Bom.GetItemAttributes(m_CADItem, 7.5)
                        Return m_CADItems
                Case 8.5
                        m_CADItems = m_Bom.ListCADItems()
                        m_Bom.GetItemProperties(m_CADItems)
                        m_Bom.GetItemAttributes(m_CADItem, 8.5)
                        Return m_CADItems
```

```
Public Overrides ReadOnly Property GetTree As List(Of CADItem)
        ...
        Dim m_Bom = New BOMNX(m_CADNX)
        Select Case m_CADNX.CADInfos.Release
                Case 7.5
                        ...
                        Return m_Bom.GetTree()
                Case 8.5
                        ...
                        Return m_Bom.GetTree()
                        ...
```

```
...
...
Dim NXitem As CADItemNX
NXitem.GetItemAttributes(m_NXApp.CADInfos.Release)
...
```

```
Friend Overrides ReadOnly Property GetItemAttributes(ByVal CADRelease As Double)
        ...
        m_GetItemAttributes(CADRelease)
        ...
End Sub

Private Sub m_GetItemAttributes(ByVal CADRelease As Double)
        ...
        Select Case CADRelease
                Case 7.5
```

Figure 10. API handling.

routines. The functional scope of the automation routines is only limited by the provided CAD-API, whereby this functionality is often coupled with the used programming language and available license packages. The common denominator of different CAD systems regarding functionality, costs and development within the presented approach is stated by the .NET framework. It is a software framework developed by Microsoft since 2002, which supports different coding languages like Visual Basic (VB), C# or J#. Due to the fact, that many CAD solutions deliver integrated VB programming editors and simultaneously provide VB.NET, the multi-CAD strategy is developed based on this programming language. The main disadvantage, compared to programming languages like C, is a lack of code protection. Nevertheless, there are some options to protect the code in an efficient and sufficient way. Considering that today's most KBD tools in automotive industry are specific in-house applications, the protection of knowledge plays an important role, [4], [14], [15], [18].

CAD related programming libraries can be provided in two species. The first one includes dynamic link libraries (dlls), which can be added as stand-alone references to the desired programming environment – NX. The second possibility is to use registered ActiveX elements, which are based on the Component Object Model (COM) standard - CATIA. In this case, the references are not linked in a static way, but based on windows registry entries. The required libraries are loaded at runtime by default, either from the startup path (dlls) or by use of the current registered COM element. This faces a problematic situation in case of different CAD releases. The code libraries of different CAD releases or versions have the same nomenclature, and therefore a specific linking method was developed.

The mentioned problem can be handled in different ways. During the iterative development process, explained in section 3.1, a large amount of different concepts has been worked out. The result, which fits all mentioned requirements under consideration of the entire strategy is explained in the next lines.

The solution bypasses the windows registry entries and allows a much more efficient realization of multi-release capability. At startup of the initialization process, the current running CAD process is linked. Based on the current CAD system the required libraries are automatically created. In the final step, the libraries are loaded at runtime. This method fulfills all requirements considering the multi-CAD approach.

The virtual connection between the KBD tool and the CAD system is carried out by a connection principle, which is also defined by the API of the CAD system. CATIA for example, is based on the Component Object Model (COM) of Microsoft, and therefore the connection to the actual active CATIA window is reached out due to the registered application name – "CATIA.Application." Thus, different CATIA releases cannot be differentiated, and therefore multiple window applications are not allowed. The API of NX on the other hand is not based on the COM strategy. In this case, the connection is performed due to the Transmission Control Protocol / Internet Protocol (TCP/IP) by use of a server and a client application, whereby both can be applied on one single computer [4], [14], [18].

The mentioned CAD initialization process and the different connection methods are implemented within the multi-CAD-Framework accessible for KDB-Developer.

5. Conclusions

The presented approach provides a platform for the efficient development of KBD applications for multi-CAD environments. This includes a multi-CAD interface and different software modules that manage inconsistent CAD APIs, capture knowledge, provide reusable functions and libraries, and have an enhanced service and error handling. In contrast to common approaches, where one single KBD tool has to be released separately for each single CAD-system and release, this novel strategy allows a standalone-releasing of different KBD tools. This simplifies company-internal roll out procedure, update management and administration. Of course this strategy is part of the framework and uses the mentioned Firebird database as application storage, user management and administration. Furthermore, it plays an important role regarding the protection of the application and consequently of the knowledge. On the other hand, the change or update management in view of the KBD-Engineer is therefore optimized, because of a centralized project management. In this way, the KBD-User is not forced to execute different application files and the standardized consistent GUI of all KBD tools minimizes the training period significantly. Referring to the starting statement concerning future trends of web-based and case-based applications, the presented framework allows the creation of four different solutions:

- Windows Form Application [14]
- Windows Console Application [14]
- Windows Service Application [14]
- Web Apps with ASP.NET [14]

In the next months the approach is applied in a broad way by implementation of the new API of CATIA V6 [4] and additional open source CAD viewing tools. Furthermore, different open source geometrical libraries will be investigated and the open xml software development kit [14] will be implemented. In this way, the functionality will grow and the platform - and vendor dependency will decrease step by step.

Considering multiply effects because of a large number of KBD tools used in automotive industry, a significant time reduction in the application of complex CAD-environment can be achieved.

ORCID

Markus Salchner http://orcid.org/0000-0001-6379-0562
Severin Stadler http://orcid.org/0000-0001-9867-0552
Mario Hirz http://orcid.org/0000-0002-4502-4255

References

[1] CADEC Works, http://www.cadec.in, access date: 2015-04-08.
[2] Catic, A.; Malmqvist, J.: Towards integration of KBE and PLM, International Conference on Engineering Design, 2007.
[3] Chapman, C.B.; Pinfold, M.: The application of a knowledge based engineering approach to the rapid design and analysis of an automotive structure, Advances in Engineering Software, 32, 2001, 903–912. http://dx.doi.org/10.1016/S0965-9978(01)00041-2.
[4] Dassault Systemes, http://www.3ds.com, access date: 2015-01-27.
[5] Eclipse, https://eclipse.org/, access date: 2015-01-27.
[6] Eigner, M.; Stelzer, R.: Product Lifecycle Management: Ein Leitfaden für Product Development und Life Cycle Management, Springer, 2009, ISBN: 9783540443735, http://dx.doi.org/10.1007/b93672.
[7] Firebird, http://www.firebirdsql.org/, access date: 2015-04-13.
[8] Hirz, M.; Dietrich, W.; Gfrerrer, A.; Lang, J.: Integrated computer-aided design in automotive development: development processes, geometric fundamentals, methods of CAD, knowledge-based engineering data management, Springer, 2013, ISBN: 9783642119392, http://dx.doi.org/10.1007/978-3-642-11940-8.
[9] KBEWorks, http://www.visionkbe.com, access date: 2015-04-08.
[10] Lin, B.-T.; Chan, C.-K.; Wang, J.-C.: A knowledge-based parametric design system for drawing dies, International Journal of Advanced Manufacturing Technology, 36, 2008, 671–680, http://dx.doi.org/10.1007/s00170-006-0882-y.
[11] Lin, B.-T.; Chang, M.-R.; Huang, H.-L.; Liu, C.-Y.: Computer-aided structural design of drawing dies for stamping processes based on functional features, International Journal of Advanced Manufacturing Technology, 42, 2009, 1140–1152. http://dx.doi.org/10.1007/s00170-008-1670-7.
[12] Ma, Q.C.; Liu, X.W.: Review of Knowledge Based Engineering with PLM, Applied Mechanics and Materials, 10–12, 2007, 127–130. http://dx.doi.org/10.4028/www.scientific.net/AMM.10-12.127.
[13] Microsoft Developer Network, http://msdn.microsoft.com/de-at/dn308572.aspx.
[14] Microsoft, http://microsoft.com, access date: 2015-01-27.
[15] PTC, http://www.ptc.com, access date: 2015-01-27.
[16] Reddy, E. J.; Sridhar, C. N. V.; Rangadu, V. P.: Knowledge Based Engineering: Notion, Approaches and Future Trends, American Journal of Intelligent Systems, 5(1), 2015, 1–17. http://dx.doi.org/10.5923/j.ajis.20150501.01
[17] Siemens PLM, http://www.plm.automation.siemens.com, access date: 2015-01-27
[18] Sanya, I.O.; Shehab, E.M.: An ontology framework for developing platform-independent knowledge-based engineering systems in the aerospace industry, International Journal of Production Research, 52, 2014, 6192–6215. http://dx.doi.org/10.1080/00207543.2014.919422.
[19] Skarka, W.: 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.
[20] Stokes, M.: Managing Engineering Knowledge – MOKA: Methodology for Knowledge Based Engineering Applications, Professional Engineering Publishing Limited, ASME Press, 2001
[21] Tacton, http://www.tacton.com, access date: 2015-04-08.
[22] Transcat-PLM, https://www.transcat-plm.com/, access date: 2015-01-27.
[23] Van Der Laan, A.H.; Van Tooren, M.J.L.: Parametric modeling of movables for structural analysis, Journal of Aircraft, 42, 2005, 1606–1614, http://dx.doi.org/10.2514/1.9764.