



Norwegian University of
Science and Technology

User Interface Concepts for Mechanism Modelling in the RaMMS KBE System

Thor Christian Coward

Mechanical Engineering

Submission date: June 2017

Supervisor: Bjørn Haugen, MTP

Co-supervisor: Ivar Marthinussen, MTP
Ole Ivar Sivertsen, MTP

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

ABSTRACT

By combining the principles of knowledge-based engineering (KBE) and concurrent engineering in the design process, repetitive tasks are reduced and design tasks are conducted simultaneously, enabling the engineer to explore a large design space early in the design process, when the committed production costs are low. This thesis investigates the principles of KBE, concurrent engineering, mechanisms, the mechanism design process and user interface development. *Rapid Mechanism Modelling System* (RaMMS) is described, it is a knowledge-based engineering application, whose goal is to automate tasks in the mechanism design process. The application is based on the object-oriented programming language Adaptive Modeling Language (AML), and enables for automatic modelling of mechanisms, the generated mechanism models is geometric representations with attached mechanism specific knowledge. User interface concepts for enhancing the manual optimisation process of mechanisms is presented, together with implementation details. The updated user interface allows for the application to be used in a larger part of the design process, enabling for dynamic simulation to be used in all phases, and consequently, aiding designers to explore a larger design space.

SAMMENDRAG

Ved å kombinere prinsippene for kunnskapsbasert ingeniørarbeid (KBE) og samtidig prosjektering i designprosessen, reduseres gjentatte oppgaver og designoppgaver utføres samtidig, slik at ingeniøren kan bruke mer tid på å utforske et stort designrom tidlig i designprosessen, når den forpliktete produksjonsutviklingsprisen er lav. Denne avhandlingen beskriver KBEs prinsipper, samtidig prosjektering, mekanismer, mekanismedesignprosessen og brukergrensesnittutvikling. *Rapid Mechanism Modeling System (RaMMS)*, er beskrevet, det er en kunnskapsbasert ingeniørapplikasjon, hvis mål er å automatisere oppgaver i designprosessen av mekanismer. Programmet er basert på det objektorienterte programmeringsspråket Adaptive Modeling Language, og det muliggjør automatisk modellering av mekanismer, de genererte mekanismemodellene er geometriske representasjoner med tilhørende mekanismespesifikk kunnskap. Brukergrensesnittkonsepter for å forbedre manuell optimalisering av mekanismer presenteres, og implementeringsdetaljer presenteres. Det oppdaterte brukergrensesnittet gjør at applikasjonen kan brukes i en større del av designprosessen, slik at dynamisk simulering kan brukes i alle faser, og dermed hjelpe designere til å utforske et større designrom.

PREFACE

The work of this master's thesis was carried out during the spring of 2017 for the Department of Mechanical and Industrial Engineering, at the Norwegian University of Science and Technology (NTNU), in Trondheim.

I would like to thank my supervisor Bjørn Haugen, and also, my co-supervisors Ole Ivar Sivertsen and Ivar Marthinussen for our frequent meetings and being part of the *RaMMS UI focus group*. Further, I would like to my co-student Arnt Lima for helping me out in times of difficulty.

Trondheim, June 2017

Thor Christian Coward

TABLE OF CONTENTS

Abstract	i
Preface	v
Table of Contents	viii
List of Figures	x
Abbreviations	xi
1 Introduction	1
1.1 Master assignment	2
1.2 Structure	3
2 Theoretical framework	5
2.1 Mechanisms	5
2.1.1 Links	9
2.1.2 Joints	9
2.1.3 Topology	11
2.1.4 Analysis of Mechanisms	11
2.2 Design Process	12
2.3 Knowledge Based Engineering	14
2.4 Adaptive Modeling Language	16
2.5 Graphical UI development	17
2.5.1 UI Design Process	17
2.5.2 Model-View-Controller	18
3 RaMMS	19
3.1 System Architecture	21
3.1.1 Collections	21
3.1.2 Joints	23

3.1.3	Links	23
3.1.4	Loads, Springs and Dampers	23
3.1.5	Mesh generation	23
3.1.6	Simulation- and Geometry Export	24
3.2	Current UI	24
4	Method	25
4.1	User Interface Development	25
4.1.1	Theoretical Study	25
4.1.2	Rapid prototyping	25
4.1.3	Implementation	26
5	UI Concepts	29
5.1	UI Solutions	29
5.1.1	Global Coordinate Input and Editing	30
5.1.2	Joint Modelling	32
5.1.3	Mechanism Link Modelling	35
5.1.4	Load-, Spring- and Damper- Definitions	38
5.1.5	FE Mesh Generation	38
5.2	UI Discussion	40
6	UI Implementation	41
6.1	Model-view-controller in RaMMS	41
6.2	Implementing the <i>Shapes Editor</i>	42
6.2.1	NURBS module	42
6.2.2	The <i>Shapes Editor</i>	43
6.3	Implementing the Table Widgets	45
6.4	<i>Mechanism Library</i> Control	47
7	Examples of Mechanism Design	49
7.1	Case 1: <i>Hoeken' linkage</i>	49
7.2	Case 2: Double Wishbone Suspension	55
8	Discussion	63
8.1	RaMMS in the Design Process	63
8.2	CAD v. RaMMS	64
8.3	Usability	65
9	Conclusion	67
	Bibliography	69
A	Appendix	73
A.1	AML UI Classes	74
A.2	NURBS Module Source Code	76
A.3	Shapes Editor Source code	80
A.4	Risk analysis	88

LIST OF FIGURES

2.1	Watt's straight line mechanism	6
2.2	Watt's rotating steam engine	6
2.3	Schematic of Hoeken's linkage	7
2.4	Schematic of knee joint using Hoeken's linkage	8
2.5	Robot knee joint using Hoeken's Linkage on robot	8
2.6	Link types	9
2.7	The six lower pairs.	10
2.8	Hoeken's linkage with link and constraint names	11
2.9	Technological islands	13
2.10	Life-cycle cost, design knowledge and freedom versus the product development timescale	13
2.11	Bridging technological islands.	14
2.12	KBE's influence on the life-cycle cost, product knowledge and freedom in the design process.	14
2.13	Generative KBE application	15
2.14	Box model widget	17
2.15	The organisation of Model-View-Controller	18
3.1	Input and output of the RaMMS KBE application	19
3.2	Class-object diagram of the main-mechanism-class and collections	22
4.1	Iterative user interface design process	26
5.1	Keypoint widget mock-up.	30
5.2	Keypoint widget mock-up with symbols	31
5.3	FEDEM menu for editing a joint	32
5.4	Joint symbols in FEDEM	32
5.5	Early mock-up of a joint widget.	33
5.6	Final constraint widget mock-up.	34
5.7	Constraint widget mock-up with symbols	34
5.8	Creating and manipulating a spline in <i>Siemens NX</i>	35

5.9	Early <i>shapes editor</i> mock-up.	36
5.10	Final shape editor mock-up	37
5.11	Load editor widget mock-up	38
5.12	Advanced load editor widget mock-ups	39
5.13	Final spring and damper table widget mock-up	39
6.1	Model-view-controller in RaMMS	42
6.2	The NURBS module	43
6.3	Annotated figure of the <i>Shapes Editor</i>	44
6.4	The keypoint table widget	45
6.5	[Annotated code for the <i>table-widget-class</i>	46
6.6	Code for the function instantiating the Keypoint Widget	46
7.1	Defining keypoints for the <i>Hoeken's linkage</i> using the keypoint table widget.	50
7.2	Defining constraints and links for the <i>Hoeken's linkage</i> using the constraints table widget.	50
7.3	Defining a load for the <i>Hoeken's linkage</i> with the load table widget.	51
7.4	Defining one spring and one damper for the <i>Hoeken's linkage</i> , using the spring- and damper table widget.	51
7.5	Mesh of the <i>Hoeken's linkage</i>	52
7.6	The <i>Hoeken's linkage</i> exported to FEDEM	52
7.7	Analysis of the <i>Hoeken's linkage</i> in FEDEM	53
7.8	Modifying keypoints using the keypoint widget	53
7.9	Analysis of the modified <i>Hoeken's linkage</i> in FEDEM	54
7.10	Defining keypoint coordinates for the double wishbone suspension, using the keypoints table widget.	55
7.11	Defining constraints for the double wishbone suspension, using the constraints table widget	56
7.12	RaMMS sweep failure	56
7.13	AML sweep error message	57
7.14	The upper link of the double wishbone suspension in the shape editor, prior to editing.	57
7.15	Modified NURBS curve of the upper link of the double wishbone suspension	58
7.16	Sweep of a modified upper link of a double wishbone suspension.	58
7.17	Modified NURBS curve of the middle link of the double wishbone suspension	59
7.18	Wheel mount imitation of a double wishbone suspension	59
7.19	The geometric model of the double wishbone suspension after modifications in the shapes editor.	60
7.20	The double wishbone suspension is meshed in AML.	60
7.21	The double wishbone suspension in FEDEM.	61

ABBREVIATIONS

AML	=	Adaptive Modeling Language
CAD	=	Computer aided design
CAE	=	Computer aided engineering
FEDEM	=	Finite Element Dynamics in Elastic Mechanisms
KBE	=	Knowledge based engineering
MVC	=	Model view controller
NURBS	=	Non-uniform rational basis spline
RaMMS	=	Rapid Mechanism Modelling System
UI	=	User interface

INTRODUCTION

Imagine you are a mechanical engineer eager to design the best double wishbone suspension ever built. First, you have to gather the best suspension engineers. Have a bunch of meetings, make numerous design concepts. Commit to some design concept. Call in a CAD expert, have him draw 3D models of chosen design. Bring the 3D models to a CAE person for analysis. Evaluate results. Do modifications, and start the whole process again. That is, of course, if you have a generous boss with an open minded economist.

Or, you can use a *Knowledge Based Engineering* application, exploring more double wishbone suspension designs, in a fraction of the conventional design time. And, if that application is linked to a multidisciplinary simulation tool, you can conduct simulations, simultaneously while exploring designs. All before committing to design concepts.

KBE introduces automation in engineering design, by storing and reusing domain specific knowledge in computer applications, and is reported to give massive time savings when applied in design processes [1, 2].

RaMMS is an acronym for *Rapid Mechanism Modelling System*, and it is a KBE application with built in mechanism knowledge, able to generate mechanism models with attached mechanism specific knowledge. These models, together with their knowledge, is exported for analysis in the dynamic simulation tool FEDEM.

The RaMMS application, was first created by Ole Ivar Sivertsen, further developed by Rasmus Skaare [3] and Anders Kristiansen together with Eivind Kristoffersen [4]. Earlier work [5] has demonstrated that RaMMS is a working KBE application for mechanism modelling, however, the lack of a good user interface (UI) is restricting users of utilising the full potential of RaMMS being in used in rapid manual optimisation of mechanism design.

This thesis presents concepts for an UI, whose objective is to improve the efficiency in manual optimisation of mechanism design in RaMMS. These concepts are discussed, and implementation details presented for those concepts found most interesting.

RaMMS with a fully working and efficient user interface will aid the mechanism designer in rapidly creating and testing new designs, freeing up time and allowing the designer to spend time on *creating* instead of *repeating*.

1.1 Master assignment

The Rapid Mechanism Modeling System (RaMMS) is the latest version of the mechanism modeling tool that the candidate studied and improved during his project assignment. The focus of this master assignment is to develop a user interface for the RaMMS program using graphical input when found practical. Techniques used in the FEDEM user interface should be considered before making the final selection of option for implementation.

The assignment includes:

1. *Specify functionality for*
 - (a) *Global coordinate input and editing*
 - (b) *Joint modeling including joint springs and dampers*
 - (c) *Mechanism link modeling including*
 - i. *Sweep curves for members*
 - ii. *Member cross section variation*
 - iii. *Surface connecting members*
 - iv. *Rounding of sharp edges (Blending)*
 - (d) *Axial spring and dampers between links and forces on triads*
 - (e) *Control system modeling*
 - (f) *Controlling FE mesh generation for links including RBE2 connections*
 - (g) *Controlling manual and automatic optimization*
 - (h) *Controlling mechanism library functions*
2. *As far as time allows, implement the functionality specified in point 1 and improve and correct the RaMMS functionality where required.*

Assignment discussion

After reflecting on the assignment points, and discussing them with the supervisors, some points were found more interesting than others. The simulation tool FEDEM provides a good UI for control system modelling, and there is another master thesis parallel to this investigating automatic optimisation using RaMMS, thus at this stage, it would have been redundant to work on adding such functionality to RaMMS, therefore point (d) *Control system modeling* and second half of point (g) (...) *automatic optimisation*, have not been given much focus when working with this thesis.

Rather has the focus been, in particular, on the first half of point (g) *manual optimisation*. Thus, the objective has been to achieve better manual optimisation, and an effort has been made to answer the rest of the points by creating UI concepts favouring this.

1.2 Structure

The remaining part of this thesis is structured as follows:

Chapter 2 gives a theoretical framework of mechanisms, the mechanism design process, KBE, Adaptive Modeling Language (AML) and UI development.

Chapter 3 presents the RaMMS KBE system, its input, system architecture and current user interface.

Chapter 4 elaborates the methods used when working with this thesis, focusing on how work has been conducted when developing and implementing UI concepts.

Chapter 5 presents UI concepts in the form of *mock-up* sketches, explains the reasoning behind the chosen solutions, and finally, discusses the various concepts.

Chapter 6 describes implementation details of the most interesting concepts of chapter 5.

Chapter 7 presents two cases of demonstrating mechanism design and iteration using the new UI.

Chapter 8 discusses how RaMMS with the updated UI can be used in the design process, compares RaMMS to conventional CAD tools, how it combines KBE and CAD, and finally, the workflow and overall usability of the updated application.

Chapter 9 draws conclusions from the discussion in chapter 8.

Chapter 10 presents proposed further work.

THEORETICAL FRAMEWORK

RaMMS is a mechanism modelling system, generating models both based on, and, with attached mechanism specific knowledge, thus, to create a good UI for such a system, this knowledge has to be well known. This chapter presents an introduction to mechanisms, both, from a historical perspective, and, how we think about, and design, mechanisms in the present-day. Further, is the design principles that RaMMS is founded upon described, namely concurrent and knowledge based engineering. Finally, the programming language of RaMMS, AML, and UI development principles are explained.

2.1 Mechanisms

The field of mechanics consist of two disciplines, *statics* and *dynamics*. Statics concerning the analysis of stationary structures, and dynamics concerning systems changing in time. With the dynamics split into two under disciplines, *Kinetics* and *Kinematics*. Kinetics dealing with forces and torques producing motion, and kinematics dealing with the geometrical study of motion.

The German mechanical engineer Franz Reuleaux, often described as the father of kinematics, defines in his book from 1876, *Kinematics of Machinery* [6], a mechanism as:

an assemblage of resistant bodies, connected by movable joints, to form a closed kinematic chain with one link fixed and having the purpose of transforming motion.

Thus, a mechanism can be described as an arrangement of interconnected resistant (rigid) bodies transferring motion. The bodies, or *links*, of a mechanism is connected by joints

constraining relative motion between the links, in such a way that the mechanism is able to transmit motion from an input link to an output link.

Mechanisms together with energy-transmission functions make machines, for example Watt's rotating steam engine (fig. 2.2), joining a steam piston together with Watt's straight line mechanism (fig. 2.1) to drive a rotating shaft.

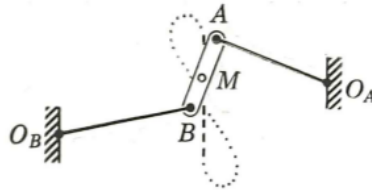


Figure 2.1: Watt's straight line mechanism [7].

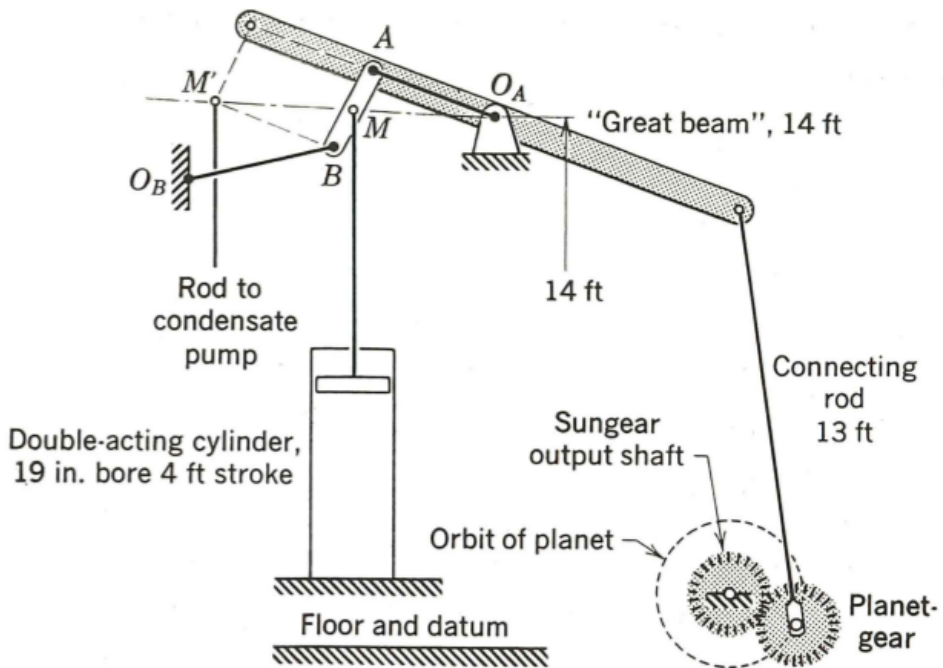


Figure 2.2: Watt's rotating steam engine [7].

Hoeken's linkage

Throughout this report, the *Hoeken's linkage* will be used as an example, it is a four bar mechanism, transferring motion from a rotating motion into two characteristic motions, first a straight line, then a quick *draw-back*. Figure 2.3 presents a schematic view of the mechanism with input motion at O_2 and output motion at P , Δx represents the straight line motion and the dotted line the *draw-back*.

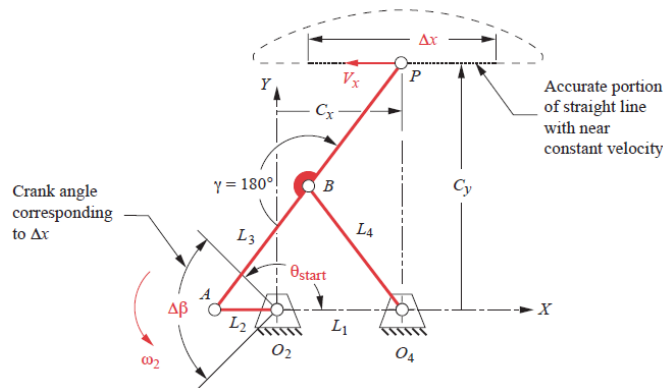


Figure 2.3: Schematic of Hoeken's linkage [8].

Although the Hoeken's linkage bear the name of the German engineer Karl Hoeken, it was first presented by the famous Russian mathematician Pafnuty L. Chebyshev [9] in 1869, and applied in his *Plantigrade machine*¹, a walking machine displayed at the 1878 World Exhibition in Paris. Most probably the name *Hoeken's linkage*, origins from an 1926 article [10] where Karl Hoeken presents the same mechanism for use in a gearbox, and his name has been stuck with the mechanism in literature [8, 11] ever since².

In the modern age the *Hoeken's linkage* has been applied to various complex systems[12, 13, 14, 15]. Knabe, Lee & Hong[15], has applied the mechanism to create robotic joints, transferring motion from an straight line actuator to a rotating motion lifting a robot's leg, a schematic representation of the *Hoeken's linkage* on top of their knee joint is presented in figure 2.4, as well as the knee joint at various output rotation angles in figure 2.5.

¹Plantigrade Machine: <http://en.tcheb.ru/1>

²It should be noted that there exists numerous online articles crediting Chebyshev by referring to the mechanism as the *Chebyshev's Lamda Mechanism*, however in contemporary literature it is referred to as the *Hoeken's linkage*.

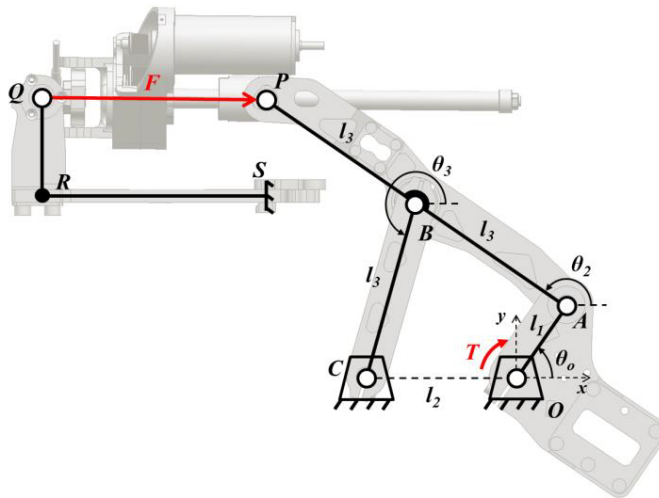


Figure 2.4: Schematic of knee joint using Hoeken's linkage, from Knabe, Lee & Hong[15].

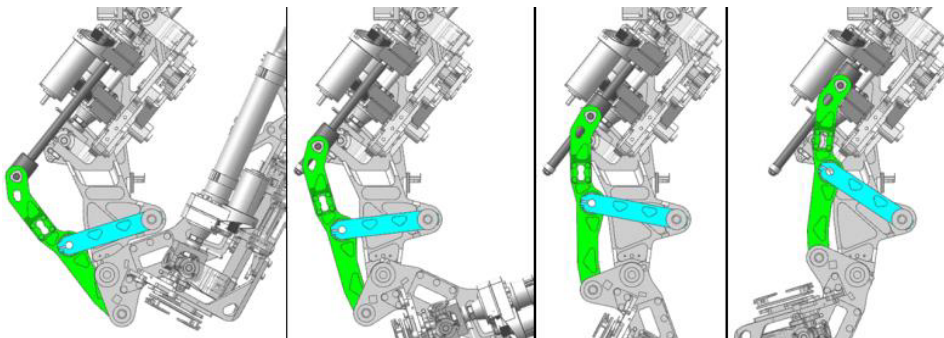


Figure 2.5: Robot knee joint using Hoeken's Linkage, at 135°, 90°, 45° and 0° from Knabe, Lee & Hong[15].

2.1.1 Links

Dealing with the kinematics of mechanisms, links are assumed to be rigid, and thus a link is defined as *the rigid connection between two or more elements of different kinematic pairs* [16]. Links can be in a wide range of shapes and forms, but, the connection points are always relatively fixed to each other, this rigid geometry between connection points, is called the link's substructure, and substructures consist of one or more rigidly fastened smaller parts, called *members*.

The links are connected to each other by kinematic pairs, also called joints, constraining relative motion. And, links are classified by their number of connections, referred to as the link's *degree*. If a link has only two connections it is said to be a binary link (fig. 2.6a), three connection points is a ternary link (fig. 2.6b), four is a quaternary (fig. 2.6c) and so on.

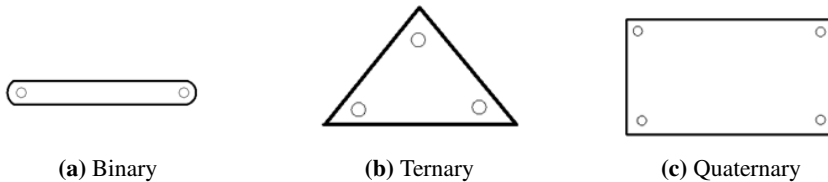


Figure 2.6: Link types

2.1.2 Joints

A joint share two mating surfaces, one for each link, therefore they are said to form *kinematic pairs*. Reuleaux[6] divided the pairs into *lower* and *higher* pairs. Lower pairs sharing surface contact, while higher pairs only sharing a line or point contact. There are only six pairs classified as lower pairs, *revolute*, *prismatic*, *helical*, *cylindric*, *sphere* and *flat*, the lower pairs are illustrated in figure 2.7 and further presented in table 2.1 with their pair symbols (as proposed by Hartenberg and Denavit [7, p. 64]), pair variable and degrees of freedom. While there are only six lower pairs, there could be an infinite number of higher pairs, e.g. a ball rolling on flat surface, mating gear teeth or a belt and pulley.

Table 2.1: The Lower Pairs, presented with name, symbolic notation, mathematical variable, degrees of freedom and relative motion.

Pair	Symbol	Pair Variable	DOF	Relative Motion
Revolute	R	$\Delta\theta$	1	Circular
Prismatic	P	Δs	1	Rectilinear
Screw	S	$\Delta\theta$ or Δs	1	Helical
Cylinder	C	$\Delta\theta$ and Δs	2	Cylindric
Sphere	G	$\Delta u, \Delta v, \Delta w$	3	Spheric
Flat	F	$\Delta\theta, \Delta x, \Delta y$	3	Planar

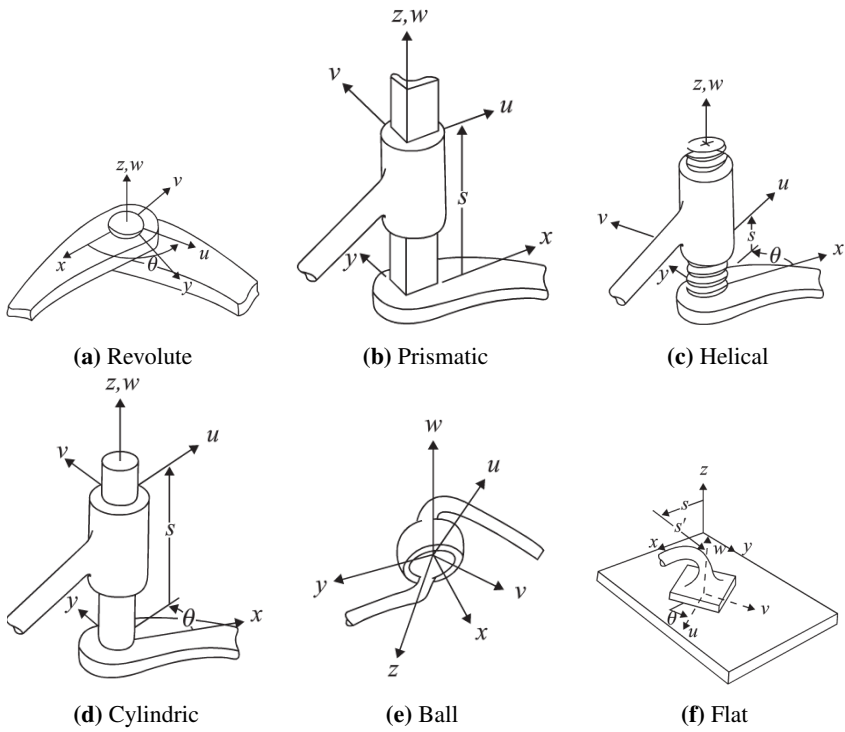


Figure 2.7: The six lower pairs.

2.1.3 Topology

The topology of a mechanism can be presented by an incidence matrix, an incidence matrix of the *Hoeken's* mechanism is presented in table 2.2, together with the link names in figure 2.8, it gives a useful presentation of male and female joints and incident links of the mechanism.

Table 2.2: Incidence matrix of a Hoekens Linkage.

Constraint	From link (male)	To link (female)
A	0	1
B	2	1
C	0	3
D	2	3

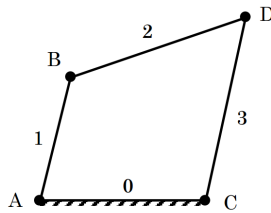


Figure 2.8: Hoeken's linkage with link and constraint names (link 0 is ground).

2.1.4 Analysis of Mechanisms

Traditionally, when designing mechanisms only the mechanism's kinematics was considered, and a kinematic analysis is done by considering the rigid body dynamics, taking no consideration of masses, forces or link shapes, solely the orientation and position of joints. Which was a fair enough approach for making the machines of Watt's era, but, the machines of the modern age require higher precision and smarter use of materials, thus, in the modern design of mechanisms a flexible body analysis is also done. A flexible body analysis considers the kinetics of the mechanism, where links are not assumed rigid, hence geometry and material properties are of importance.

Links in flexible body analyses are represented by *finite element* models. The nodes of these models are split into external and internal nodes. External nodes, also called super nodes, are point of interest, such as joint, spring and damper connections, or points of external force or mass. In the solver of the dynamic simulation tool FEDEM, the degrees of freedom for the external nodes are retained throughout the simulation, while the degrees of freedom from the internal nodes are replaced by a limited amount of vibration modes, called *modal degrees of freedom*. This reduction is called *Component Mode Synthesis* [17], and reduces the degrees of freedom of the model to only those of the external nodes and the modal degrees of freedom.

Joints are represented by constraining super nodes of different links together, with one super node being the master and the rest being slave nodes, following the masters motion in the constrained directions, e.g. in a revolute joint (fig. 2.7a) connecting two links, the master node has six degrees of freedom, while the slave node only has one rotational degree of freedom, giving the two links only one relative rotational motion. Note that connections between super nodes can also be modelled as springs or dampers, where the constraints is not rigidly locked, but has a given stiffness.

2.2 Design Process

The product development process of mechanisms is complex and involves multidisciplinary tasks, combining various engineering fields, such as CAD, kinematic analysis, and FE modelling. The process is often divided into three main activities [18, p. 6]:

1. **Conceptual design.** Based on a design specification, design parameters is established, and mechanism synthesis along with kinematic analysis is conducted, to obtain a working concept and topology, i.e. deciding number of joints, joint types, link types, lengths and orientations.
2. **Preliminary design.** The selected concept is transformed into a product, by choosing components and materials, identifying boundary conditions and creating 3D models.
3. **Detailed design.** The product's overall behaviour when introduced to forces and torques is evaluated by the use of dynamic simulation, and the product design is updated iteratively to obtain an optimum design.

This process requires a team of engineers from different fields cooperating in a sequential manner, figure 2.9 presents a scenario where different technological islands is cooperating in a mechanism design process. Communication between these islands is often manually controlled data conversions between independent software tools. Consequently, simulation and design optimisation is conducted late in the design process, when, because of commitment to technology, the cost of changing design is higher [19, 20]. Figure 2.10 presents a plot of the designers ease of change, commitment and incurred costs versus the product development process timescale. Early in the design process, the designers has a lot of design freedom, and they can influence the total product outcome the most, and as the modern age requires high precision machines and smart use of materials, the earlier dynamic simulation is done in the process, the better.

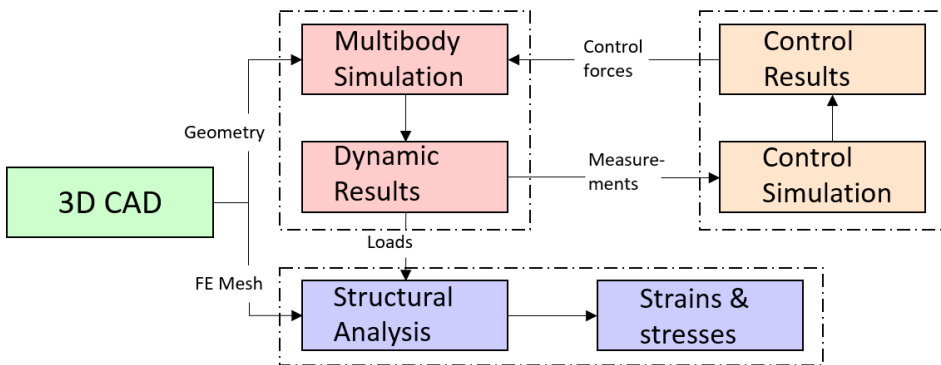


Figure 2.9: Technological islands of mechanism design [18, p. 8].

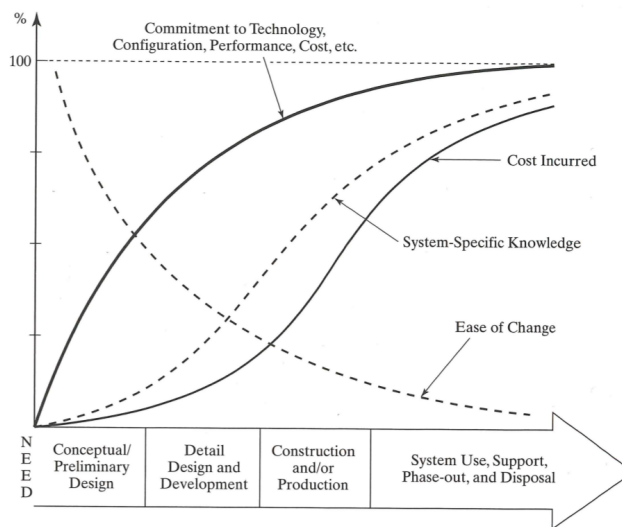


Figure 2.10: Life-cycle cost, product knowledge and freedom versus the product development timescale [20, p. 63].

Concurrent Engineering

Concurrent engineering [21], is a work methodology for product development, it involves carrying out all product development stages simultaneously, from design specification to production management. The principles of concurrent engineering can be introduced to the design of mechanisms, by replacing traditional simulation tools with multidisciplinary simulation applications [18], such as *FEDEM*, able to conduct simulations considering both kinematics and kinetics. Comparing figures 2.9 and 2.11 displays how a multidisciplinary simulation application can create a bridge between technological islands in the

design process, allowing a more concurrent design process. Enabling the designers to actively use simulations and optimisation earlier in the design process.

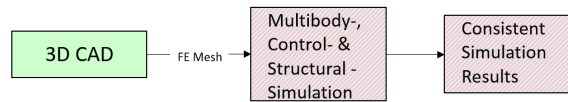


Figure 2.11: Bridging technological islands [18, p. 9].

2.3 Knowledge Based Engineering

Knowledge Based Engineering (KBE) is a way to systematically reuse product and process engineering knowledge, in order to reduce time and cost of production development, by automating repetitive and non creative design tasks [22]. This is done by creating dedicated software applications, where domain specific knowledge is programmed into the applications. KBE is used in big money industries such as: aerospace; shipbuilding; automotive; oil- and gas, and the success stories is many [19, 23, 24]. using KBE in design processes is in some cases reported to give time savings on over 90% compared to conventional design tools as CAD [1, 2, 25].

By the end of the preliminary design phase a high share of the product’s life-cycle cost is committed, employing KBE in the conceptual phase enables the designers to explore a larger design space before committing to any decisions, figure 2.12 presents KBE’s influence on the life-cycle cost, design knowledge and freedom in the production development timescale. Design freedom is increased bringing the available product knowledge up, and consequently lowering the committed cost [26].

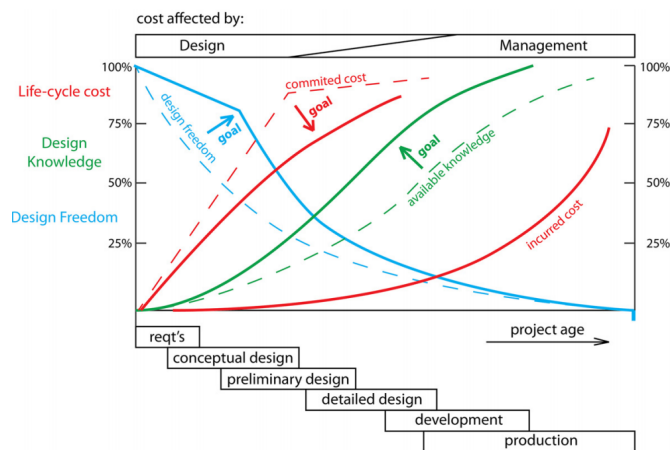


Figure 2.12: KBE’s influence on the life-cycle cost, design knowledge and freedom versus the product development timescale [26].

KBE Applications

The tools for creating KBE applications are object oriented programming languages with embedded geometry kernels and other design paradigms to aid the developer to create applications helping the design engineer in all the phases of the design model.

Product specific knowledge is stored in KBE applications, often including materials, geometric shape, stress analysis etc. and process knowledge such as manufacturing techniques, instructions etc. One major part of developing a KBE application is gathering such knowledge, it is typically done by interviewing domain experts, looking at previous designs and reading scientific articles.

KBE applications could also be distinguished by what their objective are, a typical classification of applications are *generative*, *advisory*, *innovative* or *selective*.

- Generative KBE applications create geometry, from specifications, rules and user input, an example of a generative application is presented in figure 2.13.
- Advisory KBE applications evaluate design based on product model knowledge.
- Innovative KBE applications uses model based reasoning and best practice to explore a large design space and presents possible designs to the user.
- Selective KBE applications uses user input together with domain knowledge to assist the user in selecting among various options.

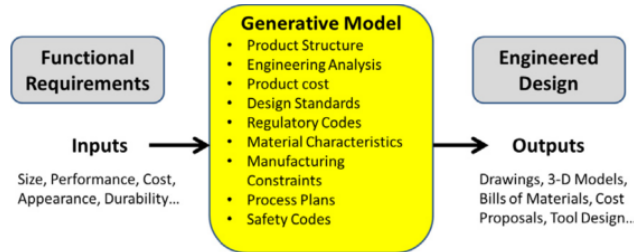


Figure 2.13: Example of a generative KBE application, taking functional requirements as input, applying relevant rules and automatically producing an engineering design [22].

2.4 Adaptive Modeling Language

Adaptive Modeling Language (AML) is a framework for creating KBE applications, developed and supported by TechnoSoft. More precisely, AML is an object oriented programming language³ with its own compiler, providing a common interface between a geometry modeller, mesh generator and a finite element analysis (FEA) solver. It uses *Parasolid* as a geometry kernel, same as state-of-the-art CAD systems as Siemens NX and SolidWorks; *MSC Patran* is used as a mesher, also used by highly regarded FEA solvers as Abaqus and Ansys; and *MSC Nastran* as FEA solver.

Classes and objects

Making a KBE application in AML is done by defining classes, and creating class and object hierarchies. AML comes with numerous built in classes for geometric modelling, meshing and finite element analysis. These classes are used as a base by the KBE developer to develop more specified classes, containing domain specific knowledge, to suit the purpose of the application.

The steps to define a class in AML is to give a class name, specify inheritance, set properties and set subobjects. Properties can be values specified by the user, given a default value or computed by the application. The subobjects needs to be given inheritance and can be given properties.

UI Classes

AML provides many pre-built UI classes, including classes to make action buttons, pull-down menus, widgets, tables, etc., the base classes are presented in figure A.1 in the appendix. The way to make a separate UI in AML is by combining these classes, and connecting them to a model through methods and functions.

A widget is made by using the `ui-form-class` as base, and placing widget tools on top of it, i.e. action buttons, pulldown menus etc. Figure 2.14 presents a simple widget to manipulate a `box-object`, the underlying code of the widget is presented in listing A.1 in the appendix.

³AML is a dialect of the object oriented programming language LISP.

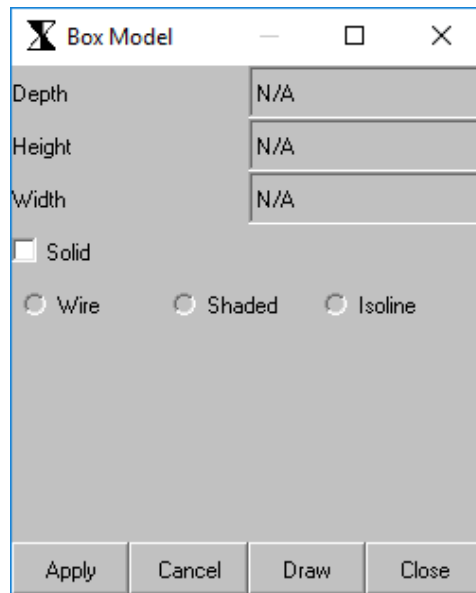


Figure 2.14: Box model widget to manipulate a `box-object`.

2.5 Graphical UI development

Generally, the objective of a UI is to help the human (user) to interact with the computer, or the underlying code of the application. A good UI is easy and intuitive for the user, but, the application's code is not always easy and intuitive to work with, hence UIs add layers of abstraction on top of the code, making it easier for the user to interact with the code. [27, p. 124-128]

2.5.1 UI Design Process

According to Gould [28], developing a good UI is a difficult process, he claims that *nobody get it right the first time (...) even if you have made the best system humanly possible, people will still make mistakes using it.*

To create a good UI, and avoiding the biggest pitfalls and obstacles Galitz [29] presents *seven commandments for designing for people* providing guidelines for the UI design process, the seven commandments are:

1. Provide a multidisciplinary design team.
2. Solicit early and ongoing user involvement.
3. Gain a complete understanding of users and their tasks.
4. Create the appropriate design.

5. Perform rapid prototyping and testing.
6. Modify and iterate the design as much as necessary.
7. Integrate the design of all the system components.

2.5.2 Model-View-Controller

Writing good code could be hard, however, by the use of design patterns to reuse code from previous working design, it could be a lot easier, as explained by Gamma et. al. [30, p. 2-4] One such design pattern is the Model-View-Controller, proposed by Trygve Reenskaug in 1979 [31], used to build UIs in the programming language *Smalltalk-80*. This way of structuring code has since been used by many when creating UIs for object oriented software [30, p. 4-6].

The principles of MVC is to decouple the software architecture into three objects: the *model*, being the application object; the *view*, being the screen representation; and the *controller*, defining how the UI reacts to user input. The organisation of MVC is presented in figure 2.15. The user interacts with the view, sending a notification to the controller, applying changes to model, the model notifies the controller of the consequences of the user action, and finally, the controller updates the view. This allows the data to change separately from its representations, and vice versa, representing the data in different ways without changing it.

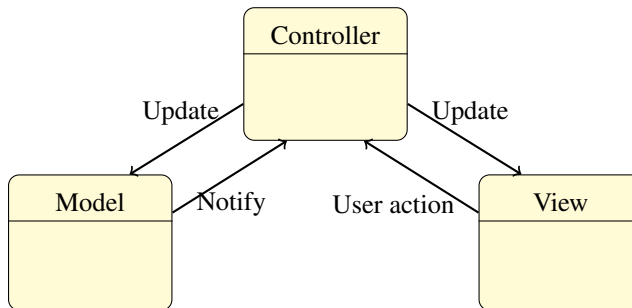


Figure 2.15: The organisation of Model-view-controller.

RAMMS

RaMMS is a KBE application combining the principles of concurrent engineering and KBE, in order to aid designers in exploring a large research space as early as possible in the design process, by creating an integrated link between geometry generation, meshing and mechanism definitions in AML and the multidisciplinary simulation tool FEDEM, enabling the designer to actively use simulations when iterating over design.

When developing a UI it is important to understand how the application works and behaves, and, to elaborate this, the following chapter describes the system architecture of RaMMS.

Figure 3.1 presents the inputs and outputs of RaMMS. It is a generative KBE application, reading a parametric mechanism definition as input, i.e. mechanism topology, incidence, external forces, springs and dampers, and generating: geometry, mesh and simulation definitions, before running dynamic simulation in FEDEM, with the final output of the RaMMS iteration process being simulation results.

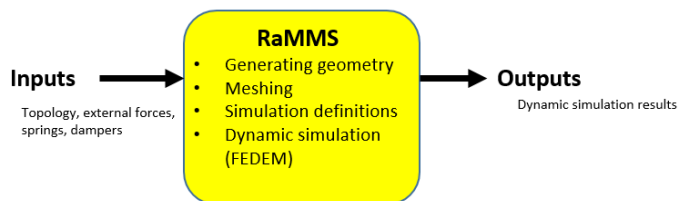


Figure 3.1: The generative KBE application RaMMS, reading mechanism definition, and outputs results.

A parametric mechanism definition of *Hoeken's linkage*, for use as input in RaMMS, is

presented in table 3.1. The mechanism definition is represented in RaMMS by five *.txt*-files, the files and contents are:

- *coordinates*: containing the coordinates (in global coordinates) of all keypoints with keypoint numbers;
- *constraints*: containing joint information, i.e. type of joint, joint direction, degrees of freedom, keypoint number and link incidence;
- *shapes*: containing link's shape information, with one line for each member, it is specified what link the members belong to, its cross section and member curve represented by NURBS¹, with keypoints and weights;
- *loads*: containing external forces and moment definitions;
- *spring-damper*: containing spring- and damper definitions.

Table 3.1: RaMMS input for the *Hoeken's linkage*

```
Coordinates.txt:
Index Name          X-pos Y-pos Z-pos
0      "crank-bearing" 0.0  0.0  0.0
1      "crank-top"    0.0  0.75 0.0
2      "rocker-bearing" 1.5  0.0  0.0
3      "rocker-top"   1.5  1.875 0.0
4      "coupler-end"  3.0  3.0  0.0
5      "spring-ground" 3.0  0.0  0.0

Constraints.txt:
Point Type          Link-incidence Joint-direction DOF Joint-variable
0      "revolute" (nil 0)          (0.0 0.0 1.0)  nil nil
1      "revolute" (1 0)           (0.0 0.0 1.0)  nil nil
2      "revolute" (nil 2)          (0.0 0.0 1.0)  nil nil
3      "revolute" (1 2)           (0.0 0.0 1.0)  nil nil
4      "free"     (nil 1)          (0.0 0.0 1.0)  nil nil

Shapes.txt:
Name      Link Member Cross-section Dimensions Points-list Weights-list
"crank"   0      0 "circular" (0.1 0.1)  nil          nil
"coupler" 1      0 "circular" (0.1 0.1)  nil          nil
"coupler" 1      2 "circular" (0.1 0.1)  nil          nil
"rocker"  2      0 "circular" (0.1 0.1)  nil          nil
"coupler" 1      1 "nil"      nil        nil          nil

load.txt:
Type      point direction      Magnitude Loaded-link
"torque"  0      (0.0 0.0 -1.0)  100        0

spring-damper.txt:
Type      from to incident-links stiffness/damping
"spring"  4    5 (1 nil)          750000.0
"damper"  1    2 (0 2)           3000.0
```

¹Non-uniform rational basis spline.

3.1 System Architecture

3.1.1 Collections

When RaMMS generates a mechanism model, one object is instantiated for each line in the input files, e.g. one line in the *coordinates.txt*-file is instantiated as a *point-object*, one line in the *constraints.txt*-file becomes a *constraint-object*, etc. For each input file there is a corresponding *collection-class* handling the input, i.e. reading files, storing input in *collection-lists* and instantiating mechanism parts as *series-objects*. The *collection-classes* are added as subobjects to the root class of RaMMS, *main-mechanism-class*, which is instantiated when RaMMS is started. The object tree for a *Hoeken's linkage* is presented in table 3.2. A UML diagram, created by Kristoffersen and Kristiansen [4], displaying the *main-mechanism-class* and the *collection-classes* is presented in figure 3.2.

Table 3.2: Object hierarchy of a *Hoeken's linkage*.

main-mechanism-class	[Level 1]
constraints	[Level 2]
revolute (nil 0)	[Level 3]
revolute (1 0)	[Level 3]
revolute (nil 2)	[Level 3]
revolute (1 2)	[Level 3]
free (nil 1)	[Level 3]
points	[Level 2]
crank-bearing	[Level 3]
crank-top	[Level 3]
rocker-bearing	[Level 3]
rocker-top	[Level 3]
coupler-end	[Level 3]
spring-top	[Level 3]
spring-ground	[Level 3]
links	[Level 2]
0 crank	[Level 3]
1 coupler	[Level 3]
2 rocker	[Level 3]
spring-dampers	[Level 2]
spring	[Level 3]
damper	[Level 3]
loads	[Level 2]
load	[Level 3]

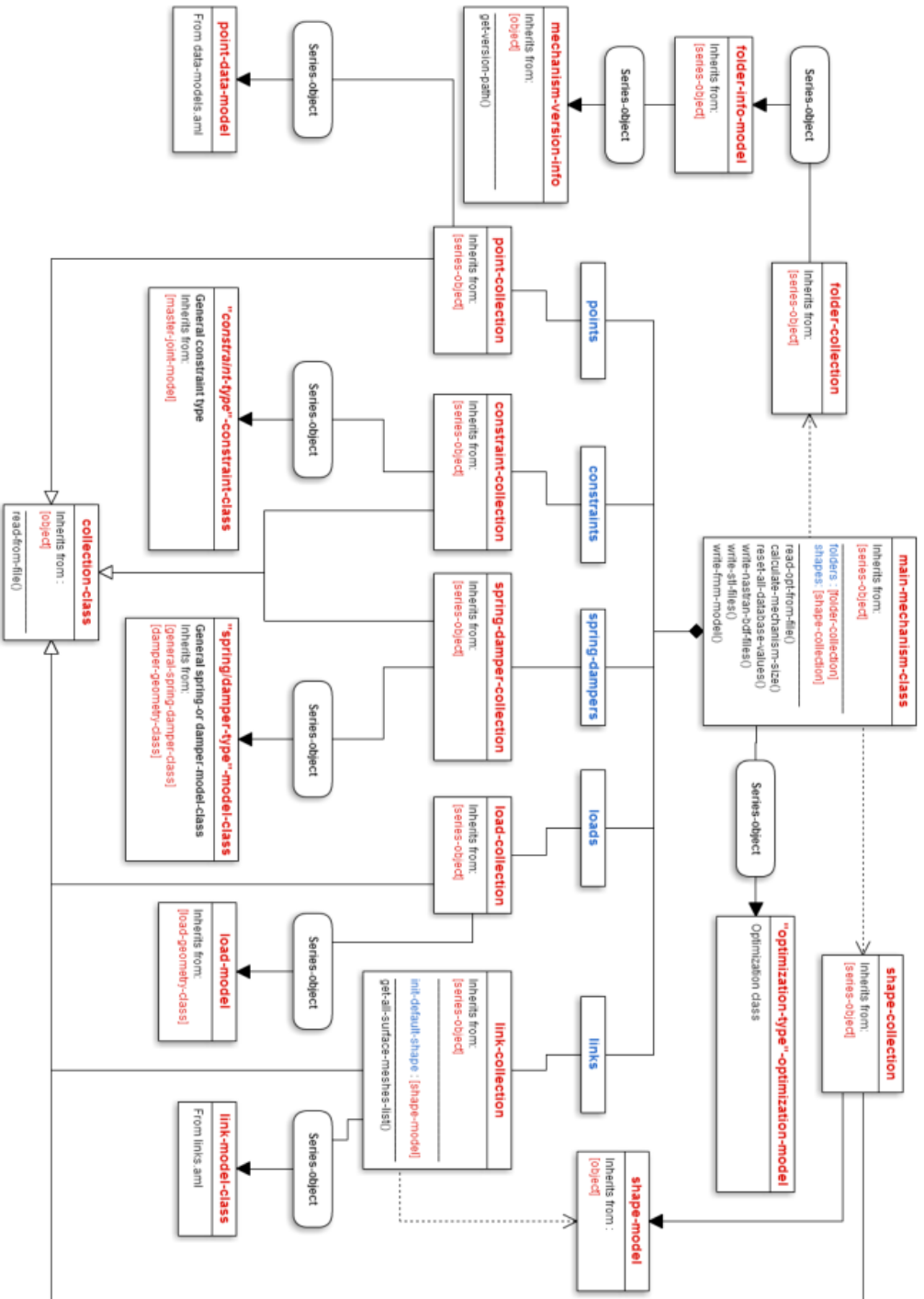


Figure 3.2: Class-object diagram of the main-mechanism-class and collections

3.1.2 Joints

RaMMS is able to generate revolute-, ball- and knuckle joints², as well as, rigid connections³ and free connections⁴. Joints are instantiated by the *constraint-collection* class, it reads joint type, DOF, position, direction and link connections from input, and instantiates the correct joint type geometry, with a male and a female object, sharing a common connection, placed at the joint's specified keypoint. The different joint types have their own specialised geometry, e.g the revolute joint consist of a male and a female part, where the female part is having a diameter of 1.2 times the biggest cross section diameter of its connecting members.

3.1.3 Links

Links are created as a result of the constraint definitions, when defining constraints in the *constraint.txt*, each constraint's link incidence is specified, and from these, links are generated between constraints.

If no shapes are specified, the application will generate links as straight lines between constraints, with a default circular cross section. Link shapes have to be defined in the *shapes.txt*-file. The *shape-collection* class reads input from that file, and instantiates the total link geometries of the mechanism as subobjects. The class of these subobjects are *link-geometry-class*, joining members, surfaces and joints to form link substructure between joints.

3.1.4 Loads, Springs and Dampers

Loads are defined in RaMMS as external forces or torques. The loads can be constant or given a simple scaling, and its directions is specified in the input.

Spring and damper start points, end points, stiffness, damping and incident links are read from *spring-damper.txt* by the *spring-damper-collection* class. It creates springs and dampers as subobjects accordingly to the inputs given.

3.1.5 Mesh generation

The link's total geometry is meshed by the *link-mesh-class*, with individual mesh sizes for members, surfaces and joints. Members have a mesh size of 25% of their smallest member cross section, while the various joints types have different meshing rules suited for their geometry.

²Double revolute joints

³rigid joints

⁴User specified fixed translational- and rotational- directions

In the mesh, joint connections are represented by connecting rigid body elements (RBE2), a slave triad (node) on the female connection is constrained to a master triad on the male connection.

3.1.6 Simulation- and Geometry Export

In order to export the mechanism geometry to simulation in FEDEM, *.bdf* and *.fmm* files are written by RaMMS. One *.bdf* file is created for each link, containing mesh information including RBE2 definitions. And a single *.fmm* file, containing joint, spring, damper and load definitions, all referring to node placements in the FE mesh of the corresponding *.bdf* files.

3.2 Current UI

RaMMS comes with a simple user interface, containing functionality to do only the most important tasks, i.e. instantiating the mechanism, creating mesh and exporting to FEDEM. However, the current UI offers few options for editing mechanisms, where the main problem being the complicated way to edit and create input files [5], thus, the lack of a good user interface is slowing down the potentially very rapid iteration process.

The current way to iterate over a mechanism design in RaMMS is by following these steps:

1. Inspect the results from the previous iteration.
2. Use a separate text editor to edit input files, i.e.:
 - (a) Update keypoints in *coordinates.txt*
 - (b) Update joints and links in *constraints.txt*
 - (c) Update link shapes in *shapes.txt*
 - (d) Update loads in *loads.txt*
 - (e) Update springs and dampers in *spring-damper.txt*
3. Restart the RaMMS AML application.
4. From the UI do the following: draw (instantiate) the mechanism; create mesh and *.bdf* files; export to FEDEM (FEDEM opens automatically when exporting).
5. Run simulation.
6. Evaluate results, and start another iteration from point 1.

METHOD

4.1 User Interface Development

To design a user interface for RaMMS, and specifying functionality for the points of the master assignment (presented in chap. 1.1), Galitz' [29] seven commandments for designing for people were used as a guideline in the design process. Starting with an explore phase, meaning a study of mechanisms and user analysis; then doing *rapid prototyping*, in the form of iterating over, and discussing, *mock-ups*; and finally, implementing a design. The process is presented in figure 4.1, and elaborated further in the following subsections.

4.1.1 Theoretical Study

To gain a complete understanding of users and their tasks, a theoretical study was conducted on mechanisms and the mechanism design process. The basics of KBE and KBE applications were studied, together with GUI development processes and GUI development using AML.

4.1.2 Rapid prototyping

Designing user interfaces for applications is a complicated process, many factors must be considered, and trade-offs have to be made, e.g. restricting user freedom yields fewer errors [27], thus, a relevant question is: how much freedom should the user be given when interacting with the system? Answering such questions is hard for the developer, especially for specific problems — in this case, predicting errors when designing specific

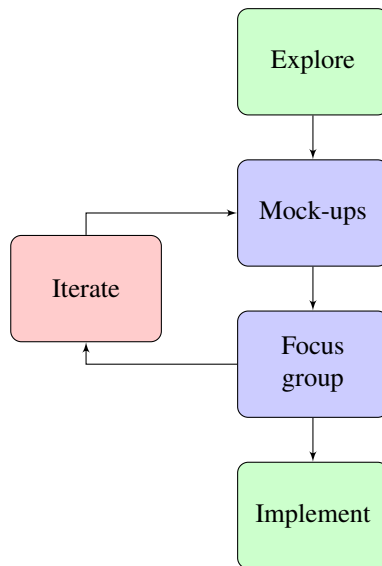


Figure 4.1: Iterative user interface design process.

parts of mechanisms — little, if any, data is to be found, leaving the UI developer simply relying on educated guesses.

For rapid prototyping of UI designs, *mock-ups* is a useful tool. A mock-up could be a hand drawn sketch or a dummy interface not connected to an application. They give an quick insight into how applications will look and work, making it easier to visualise and discuss UI solutions, and they can be created quickly, making them a powerful tool when iterating over UI designs.

Working with the RaMMS UI, mock-ups were created, to present interface solutions to a focus group of people with IT and mechanical engineering backgrounds. The mock-ups were dummy widget menus created by the tool *QT Designer*, and, based on feedback from the discussions mock-ups were iterated upon, and further discussed. The final result of this rapid prototyping process is presented in chapter 5.

4.1.3 Implementation

After doing rapid prototyping, the final mock-up menus were reviewed, and, the ones that seemingly best would enhance the usability of RaMMS, were chosen to be implemented. The implementation were done by writing AML code, first code creating standalone wid-gets mimicking the mock-up widgets, then writing methods interacting with the RaMMS application. Implementation details are explained in chapter 6.

AML UI development method

The AML reference manual [32] provides documentation for the UI classes, together with example codes for some classes. Prior to implementing the new UI of RaMMS, in order to gain an understanding on how UIs is created in AML, the examples of the reference manual were studied, and based on these examples, small simple widgets were created, e.g. the box model widget in figure 2.14. The knowledge learnt from creating and experimenting with these simple widgets were used when creating code for the RaMMS UI.

UI CONCEPTS

The objective of the updated RaMMS UI is to enhance the manual optimisation process. In order to achieve this, two *key-attributes* was given extra focus while creating UI solutions and discussing mock-ups, the values were:

- **Efficiency.** For rapidly iterating over designs, the ability to quickly create and modify mechanisms is essential, and thus, was given a high priority in creating GUI solutions.
- **Familiarity.** For the user's ease of use, terminology and symbols should be familiar, and users of RaMMS is most likely familiar with mechanisms and the FEDEM user interface, thus an effort has been made to make symbols and terminology as close to mechanism literature and FEDEM as possible.

5.1 UI Solutions

RaMMS relies on five separate input files, and the previous UI did not have any functionality for editing this input, imposing the user to use a separate text editor to create and edit input. To avoid this *bottleneck*, the UI should offer functionality to create and edit input from within RaMMS.

Further, defining mechanisms from within RaMMS could be done by a *top-down* approach. Thus, will the steps to define a mechanism, and conduct a simulation in RaMMS be the following:

1. Define keypoints.
2. Define joints, and links connected to joints.

3. Modify link substructures.
4. Add loads, springs or dampers to keypoints.
5. Mesh geometry and export to FEDEM.

The following sub-sections presents user interface solutions for defining and optimising mechanism design for each of these steps.

5.1.1 Global Coordinate Input and Editing

Keypoints are an essential part of the mechanism definition in RaMMS, it is they who define the mechanism topology, because all mechanism parts are connected to a keypoint. Keypoints are defined by coordinates in the global coordinate system, and defining keypoints is a straight-forward task, thus it should be simple for the user.

A table gives a quick overview of the keypoints, and allows for easy coordinate modification. Further, for easy visualisation, symbols and keypoint numbers should appear at keypoint positions in the modelling window, and if the user highlights a keypoint in the table, the symbol of that keypoint should change colour, demonstrated in figure 5.2. There should also be an option for the user to interactively place and *drag* keypoints in the modelling view, alongside manipulating coordinates from the table.

The final result of the mock-up process for defining keypoints, was the pop-up widget, displayed in figure 5.1. Overall, this widget gives a quick and easy way to manipulate global keypoint coordinates.

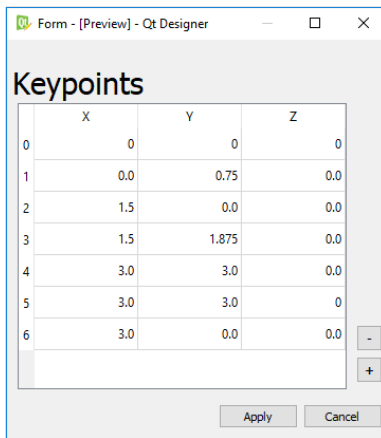


Figure 5.1: Keypoint widget mock-up.

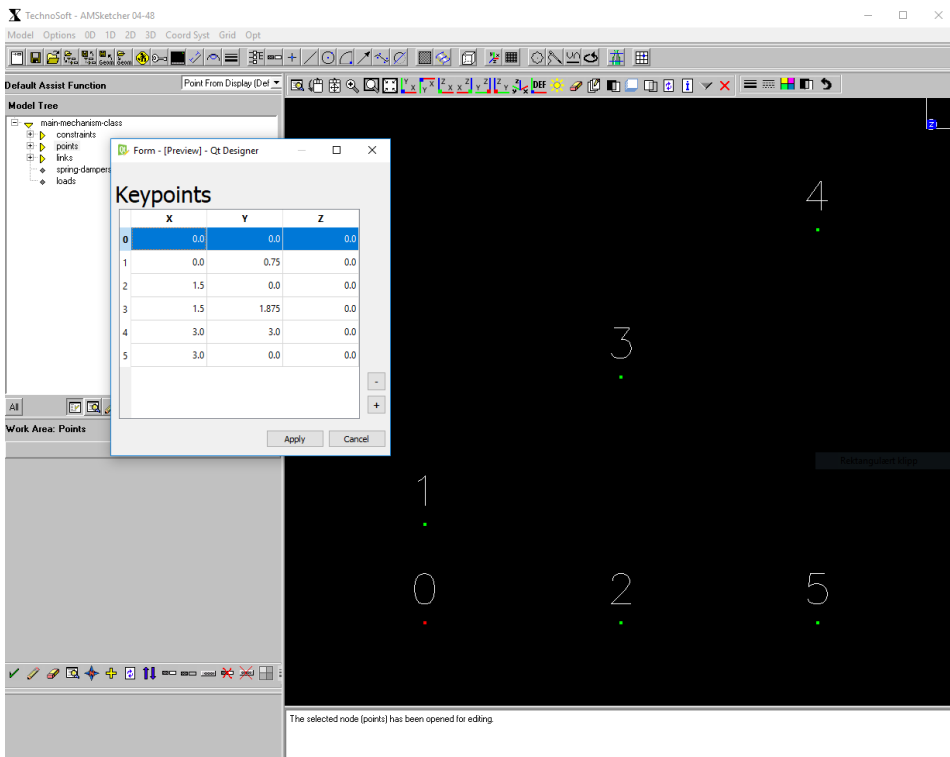


Figure 5.2: Mock-up displaying the keypoint widget in use, with keypoint labels and symbols.

5.1.2 Joint Modelling

Defining joints is done by specifying: joints' placement; type; connected links; directions; and in the case of a *free-connection*, specifying fixed degrees of freedom. Link topology are defined based on the joint definitions, joints with same incident link creates a link between them.

Joints in FEDEM

The menu to edit joints in FEDEM is presented in figure 5.3, FEDEM displays one joint at the time, and the menu consists of a window to the left with joint information, i.e. incident links and loads, and a summary table to the right containing info about the joint's degrees of freedom, as well as attached loads and springs. There is also a button for swapping master and slave connections. FEDEM's modelling view symbols representing the joints is presented in figure 5.4, for familiarity reasons the same symbols should in the RaMMS modelling viewer.

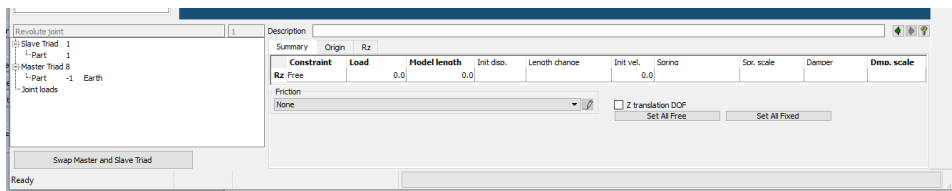


Figure 5.3: FEDEM menu for editing a joint

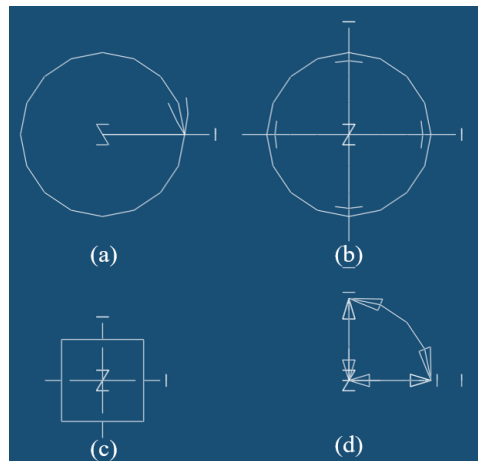


Figure 5.4: Joint symbols in FEDEM, (a) Revolute, (b) Ball, (c) Rigid, (d) Free

Joint Modelling Mock-ups

With inspiration from FEDEM mock-ups for defining joints was worked out. The early mock-ups were pop-up widgets (fig. 5.5) for defining one single joint at a time. After group discussions, this was found to be a bad design solution, unnecessary slowing the joint defining process, by imposing the user to open and close pop-ups for each joint.

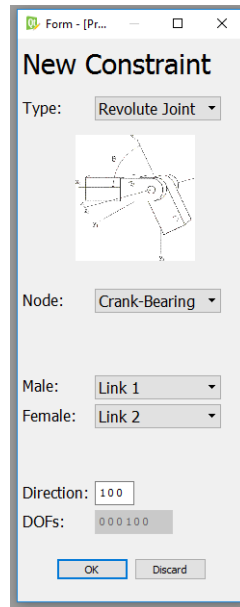


Figure 5.5: Early mock-up of a joint widget.

Instead a table widget, similar to the keypoint widget, was chosen, as it provides a faster way to edit multiple joints, and gives a good overview of a mechanism's joint configuration, and consequently via the *link-incidence*, the mechanism's link configuration.

To further improve the ease of defining joints, it was found beneficial to add the following functionality to a joint widget: allow the user to either *type-in* keypoint number, or choose keypoint interactively from the modelling window; having a drop-down menu displaying the joint types to choose from (as seen in the mock-up in figure 5.6); greying out the degrees of freedom not available to choose, because of the chosen joint type; and having a button to easily switch master and slave connections. Further, when the widget is opened, the keypoint symbols and numbers should appear together with joint symbols, also lines should appear when links are defined between joints.

The mock-up of the final design solution including all this functionality is presented in figure 5.6, and a mock-up of the widget in use, together with joint symbols, and lines representing links, is displayed in figure 5.7.

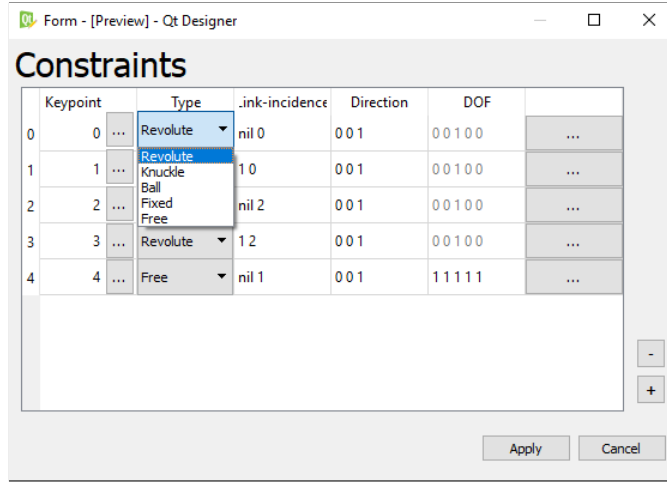


Figure 5.6: Final constraint widget mock-up.

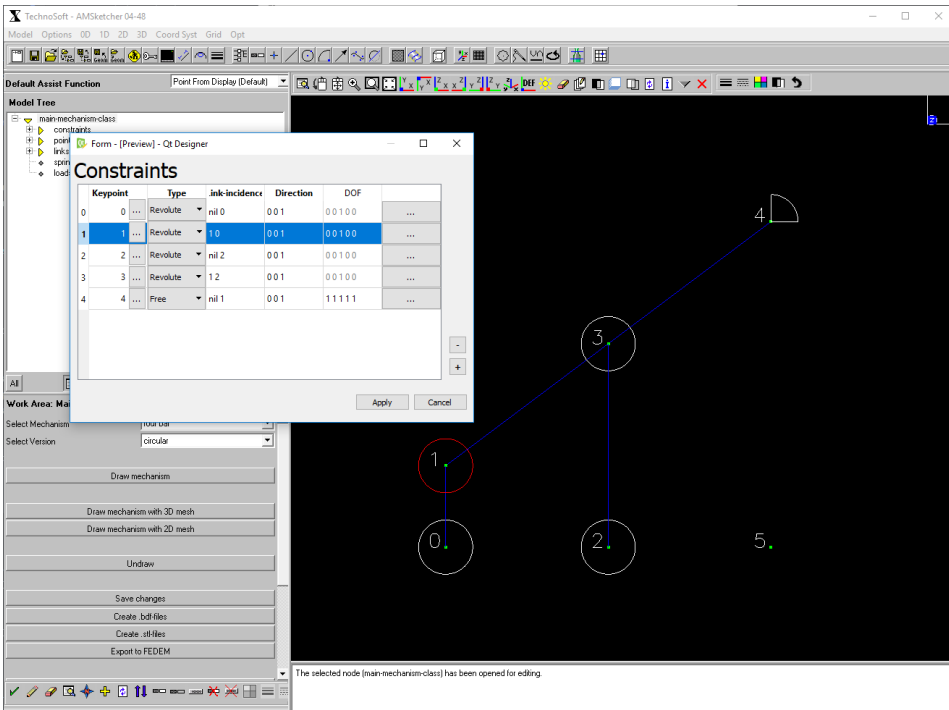


Figure 5.7: Mock-up displaying the constraints widget in use.

5.1.3 Mechanism Link Modelling

Mechanism link modelling is done by editing the substructure shape of links. The substructure does not affect the kinematics of a mechanism, as the link's connections always have the same rigid relative distance. By default RaMMS generates links' substructures as straight lines, with a default cross-section, however, the user can specify weight points in the input files to make NURBS curves for the substructures to follow, as well as cross sections and cross section variations along curves.

FEDEM does not have any functionality for editing link substructures, and building such functionality into RaMMS would mean that it can be used for detailed design, consequently dynamic simulations can be used in all the stages of the design process. Therefore was the mechanism link modelling an area of particular interest when working on the RaMMS user interface.

Link Modelling Inspiration

Implementing a link substructure editor into RaMMS would be somewhat similar to building a small CAD system with NURBS curve functionality into AML. Thus, user interface solutions for creating splines in *state-of-the-art* CAD systems were investigated. Systems like *Autodesk Inventor*, *Solidworks* and *Siemens NX* (fig. 5.8) all uses a highly interactive way to manipulate splines, by using *drag-and-drop* functions to move weight points, and instantly updating the curve when a point is moved.

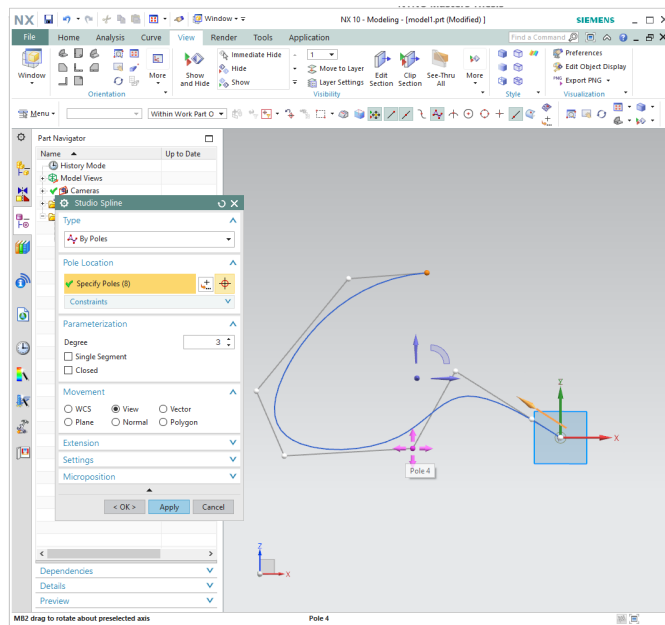


Figure 5.8: Creating and manipulating a spline in *Siemens NX*.

Link Modelling Mock-ups

Various solutions were discussed, and simple mock-ups were made of a *Shapes Editor* widget. Which is a widget opening a single link in its own separate modelling window, and the ability to modify that link's substructure by manipulating a NURBS curve representation of the link's members. One of the early shapes editor mock-ups is presented in figure 5.9. It has quick access to manipulate weight points, and functionality to sweep the selected link from within the widget, in order to quickly see how the sweep of the new shape looks.

Furthermore, there is a lot of functionality that can be good to implement into a link substructure editor, essentially, all detailed design features that are normally added to a mechanism late in the design process, could be beneficial to access from the same widget. This includes features such as edge blending, cross-section details, surfaces between link members, and the ability to edit joint directions and override joint sizes.

RaMMS has built in knowledge about meshing routines, but, after editing substructure shapes, resulting geometries could be abnormal, and the built-in meshing would struggle to create a good mesh, thus, it would create a good work-flow to be able to edit and generate mesh from within the same menu as editing substructures, to quickly check meshes, and create specialised meshes in the case of abnormal geometries.

If a link has more than one member, the modelling window should display the curves of the whole link, while the selected member should have a different colour, and the weight points of the selected member should be displayed together with weight point numbers, and there should be functionality to interactively manipulate the points, ideally, by *drag-and-drop*.

The final solution after the mock-up iterating process, was an widget including all this functionality, and this mock-up is presented in figure 5.10.

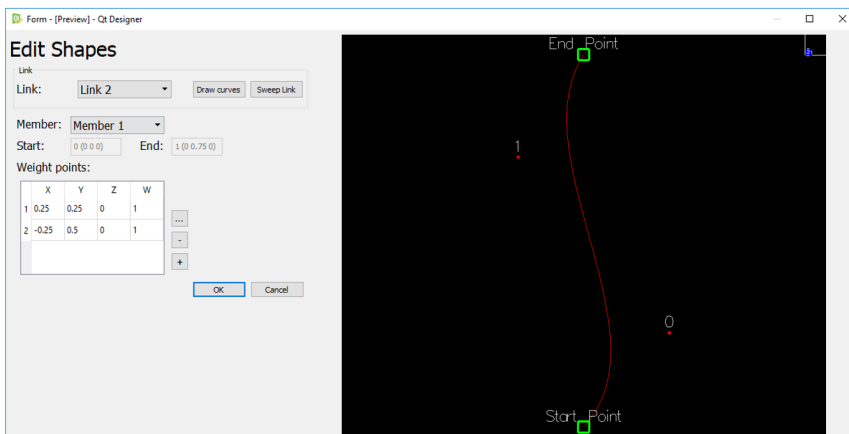


Figure 5.9: Early *shapes editor* mock-up.

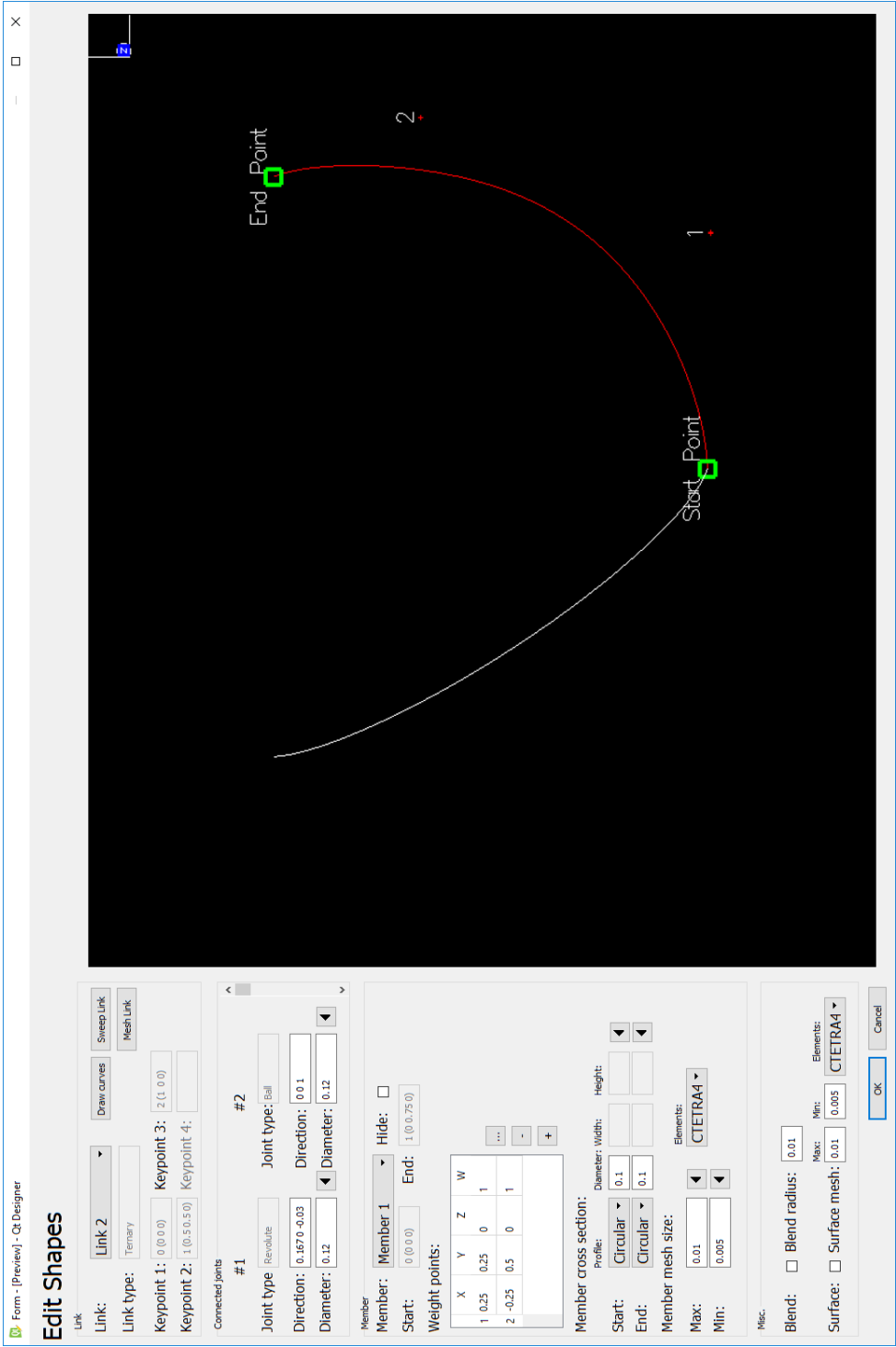


Figure 5.10: Final shape editor mock-up.

5.1.4 Load-, Spring- and Damper- Definitions

Adding and editing mechanism loads, springs and dampers can be done by table widgets, similar to the joint table widget mock-up.

The final mock-up for loads is presented in figure 5.11. In this mock-up the user can choose load type from a drop-down menu, specify directions, loaded links, and pressing a button opening an advanced load option widget, presented by the two mock-ups in figure 5.12, where the user can specify simulation options for dynamic loads, by simple built-in mathematical functions. However, FEDEM has a good interface for defining advanced mathematical functions, and it would be redundant to add more than only the simplest mathematical functions into the RaMMS UI.

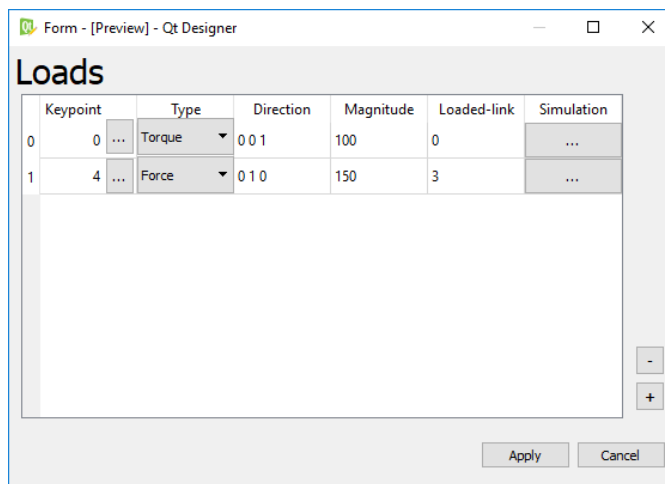


Figure 5.11: Mock-up of the load editor widget.

The final mock-up of the spring and damper table widget is presented in figure 5.13. From drop-down menus, the user can choose type, and place springs or dampers from keypoint to keypoint or in in joints. Springs and dampers could also have an advanced simulation options widget, similar to the advanced load options, where stiffness, damping and stress free lengths could be defined by mathematical functions.

5.1.5 FE Mesh Generation

When creating mesh of simple geometry, a goal is to keep the mesh generation as autonomous as possible, however, the user should be able to easily modify mesh for special cases. Because of this, the mesh could be controlled from within the shapes editor widget, where special geometry is added to the mechanisms. In the bottom part of the mock-up in figure 5.10, a solution to how mesh can be controlled from within such a widget is displayed.

Load

Type: Fixed Displacement

Node: 0 (0.0 0.0 0.0) ...

Magnitude (m/s): 0.1

Direction: 1 0 0

Simulation Options

Start (s): 0

Stop (s): 100

Function: Constant

OK Discard

Load

Type: Torque

Node: 0 (0.0 0.0 0.0) ...

Magnitude (Nm): 300

Direction: 1 0 0

Simulation Options

Start (s): 0

Stop (s): 100

Function: Linear

Slope start (s): 10

Slope end (s): 90

OK Discard

(a) Constant load.

(b) Linear load.

Figure 5.12: Mock-ups the of advanced load editor widget.

	Type	Config	Point-from	Point-to	Incident links	Stiff./Damp.
0	Spring	Point to pair	5 ...	6 ...	1 nil	750 000
1	Damper	Joint	4	3000
2	Damper	Point to pair	1 ...	2 ...	0 2	3000

-
+

Apply Cancel

Figure 5.13: Final spring and damper table widget mock-up.

5.2 UI Discussion

The solutions presented in this chapter, have been discussed and iterated upon, while focusing on the *key-attributes*, presented in section 5.

Using tables for manipulating mechanism definitions, would give users a quick overview of definitions, and with familiar symbols together with interactive manipulating functions, should increase the ease of using RaMMS. Furthermore, the *Shapes Editor* is of special interest, because it would enable for RaMMS to be applied even further into the design process.

UI IMPLEMENTATION

Implementing the proposed user interface solutions into RaMMS was done by writing AML code creating widgets, then connecting the widgets to the application by writing methods, all whilst following the principles of model-view-controller. After group discussions, the UI solutions, were prioritised, and a great portion of the implementation time was given to implement the *Shapes Editor*, while the *table-widgets* were implemented with less diligence. This chapter gives an details on how the UI implementation was conducted, and presents screenshots of the final widgets.

6.1 Model-view-controller in RaMMS

Essentially, when RaMMS generates a mechanism, the collection classes read input from input files, and stores it in various list, e.g. input from *coordinates.txt* is stored in *points-list* and input from *constraints.txt* is stored in *constraints-list*, based on these lists the parts of the mechanism is instantiated.

In the sense of model-view-controller, these collection-classes is the controller of the system, reading input and creating objects based on that input, when the input is modified the collection classes reads the new input, keeping track of which objects that have been modified, and recreates the modified objects.

A user interface should only interact with the controller, thus, the user interface should read the lists of the collection classes, present them in such a way that the user can interact with them. Further, after user manipulation it should update the lists of the collection classes with new data, and finally, the collection classes should update the model with the new data, separate from the user interface. A visual representation of this process is given in figure 6.1.

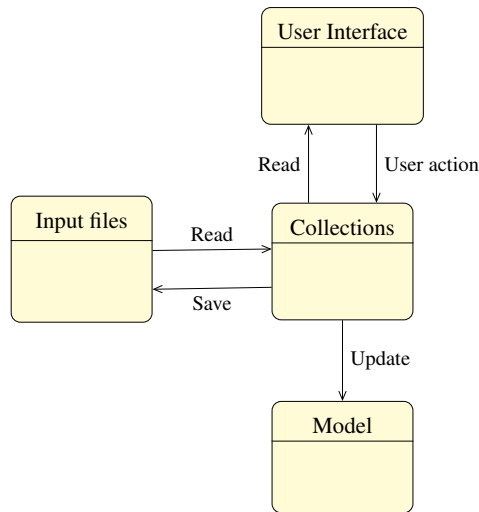


Figure 6.1: Model-view-controller in RaMMS, user interface and input files interacting with the collections classes.

6.2 Implementing the *Shapes Editor*

The *Shapes Editor*, presented in chapter 5.1.3, is a pop-up widget consisting of many different UI constructs, it contains: a modelling view, *actions buttons*, *drop-down menus*, *form fill-ins*, *radio-buttons*, *tick-boxes* and a table for editing weight points.

In order to create this widget, the documentation for the AML UI classes were studied, and based on the examples in the documentation, small separated widgets containing the UI constructs were created. Then, the constructs of these separated widgets were combined into larger widgets, such as the *NURBS module*. And finally, the *Shapes Editor* were created, based on the *NURBS module*.

6.2.1 NURBS module

The *NURBS module* is a standalone widget for manipulating *NURBS curves* in AML. It was created to learn how a separate modelling view can be implemented into a pop-up widget, and how a *spreadsheet table* can be used to manipulate data.

The module is presented in figure 6.2, and its code can be found in listings A.2 in appendix A.2. As input, it reads coordinates for start and end points, as well as coordinates and weights for weight points, based on this input a *NURBS curve* is generated in the modelling view, and the curve can be manipulated by editing table values, or *dragging* points in the modelling view.

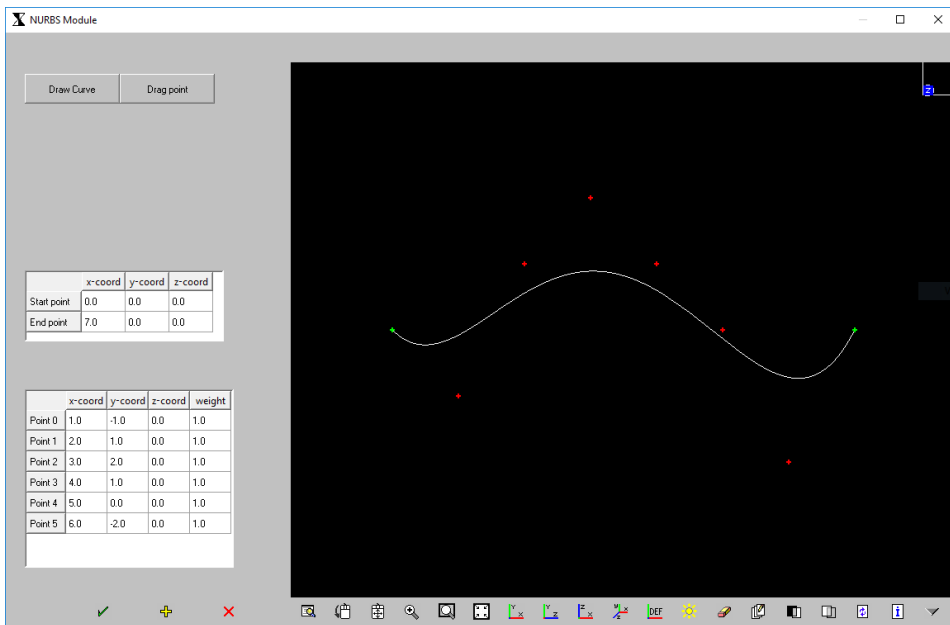


Figure 6.2: The NURBS module.

6.2.2 The *Shapes Editor*

To write the *Shapes Editor* code, the NURBS module was used as base, and to add the UI constructs, code from the earlier created *simple widgets* was simply copied and pasted into this base. This final code of the *Shapes Editor* can be found in listing A.3 in appendix A.3

The *Shapes Editor* reads RaMMS' collection classes, and presents link and member shape information. From drop-down menus can the user choose which link to display, and which member on the selected link to manipulate. The modelling view presents the link shapes by NURBS curves. A red line for the selected member, and white lines for the other of the link's members. An annotated figure of the *Shapes editor* going in depth on its functionality, is presented in figure 6.3.

For the *Shapes Editor* to communicate back to RaMMS, methods was written to update RaMMS' collection classes when the user interacts with the *Shapes Editor*, e.g. whenever the user edits the weight points table, a method loops over the table contents, and updates the *weight-collections* class with the updated weights list, the collections notifies the *Shapes Editor* back, and the modelling view is updated with the updated weight points.

When the *Shapes Editor* widget is closed, the total mechanism geometry is updated and displayed in the regular modelling view of RaMMS.

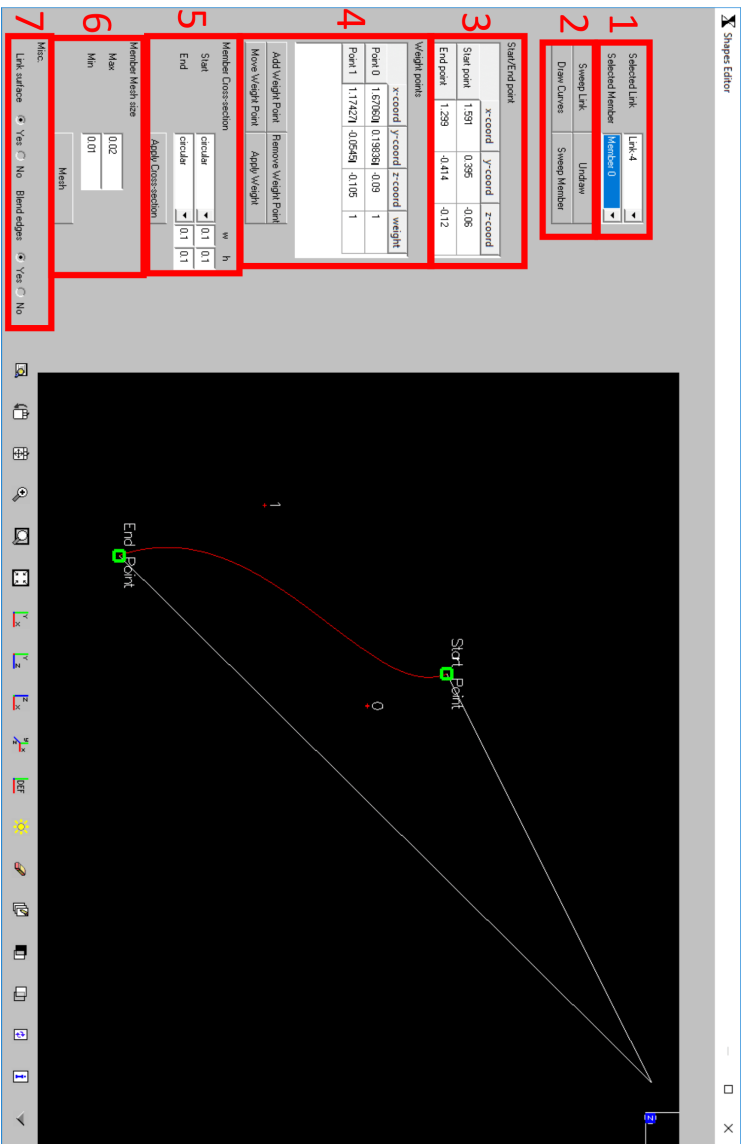


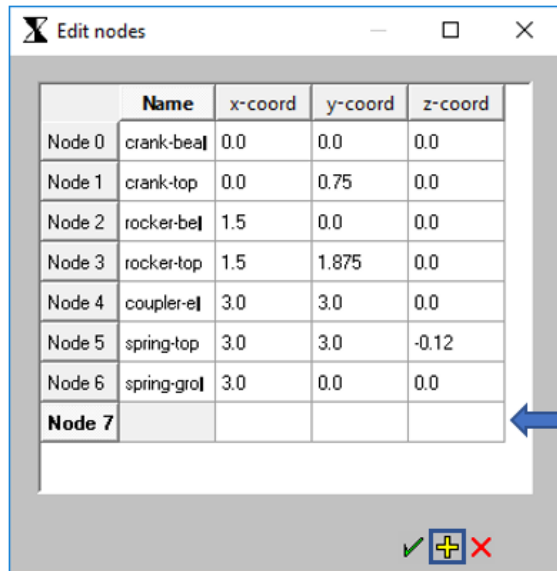
Figure 6.3: Annotated figure of the *Shapes Editor*. (1) A link is selected from the top dropdown menu, then a member is selected from a dynamic dropdown menu showing the selected link's members. (2) Options to sweep from the total link geometry, including joints; draw NURBS curves with weight points, as seen in the modeller; undraw everything; and to sweep individual member geometry. (3) The member's start and end point coordinates, these are fixed coordinates not editable by the user. (4) Weight points table, editable by the user, with options to move weight points interactively in the modeller, add and remove points. (5) Editable member cross section profile and dimension. Can vary from member-start to member-end. The available cross sections can be selected from a drop-down menu. (6) Individual member mesh size can be set, and the link's geometry can be meshed from within the *Shapes Editor* to verify and control the mesh. (7) Surface between link members and edge blend can be turned on or off.

6.3 Implementing the Table Widgets

The final mock-ups for defining keypoints, joints, loads, springs and dampers are all table-based widgets, for manipulating RaMMS input. This input is stored in lists in the collection classes, therefore, the table widgets should read these lists, and present the contents to the user. After user manipulation, the modified *collection-class* list should be updated, and followed by an update of the model.

While working on the *Shapes Editor*, a simple table widget was created, and this widget was used as a base to create the table widgets. To keep it simple, a reusable *table-widget-class* was created, able to instantiate all the table widgets. The table widgets consists of a table and three buttons, a keypoint table is presented in figure 6.4. When the table is opened the appropriate symbols appear in the modelling view, e.g. upon opening the joint widget, joint symbols and link member lines are drawn.

Its code is presented, with annotations, in figure 6.5, the main *table-widget-class* inherits from *ui-form-class* defining the widget form, and the UI constructs is added as subobjects to the table-class. The *sheet*, i.e. table, inherits from *ui-spreadsheet-class* to create the table, its cell values is specified upon instantiation. The instantiation of a keypoint widget is presented in figure 6.6, the cell values is defined by the *input-list* property, reading values from the *points* object, an instantiation of the *points-collection* class.



	Name	x-coord	y-coord	z-coord
Node 0	crank-bea	0.0	0.0	0.0
Node 1	crank-top	0.0	0.75	0.0
Node 2	rocker-be	1.5	0.0	0.0
Node 3	rocker-top	1.5	1.875	0.0
Node 4	coupler-e	3.0	3.0	0.0
Node 5	spring-top	3.0	3.0	-0.12
Node 6	spring-gro	3.0	0.0	0.0
Node 7				

Figure 6.4: The keypoint table widget. Pressing the plus button adds a new line to the table, and pressing the red cross removes the last line.

```

1  (define-class table-widget-class
2    :inherit-from(ui-form-class)
3    :properties(
4      measurement 'pixels
5      width 320 height 350 x-offset 40 y-offset 40
6      input-list-length (length `input-list)
7      number-of-columns (length `column-labels)
8    )
9  )
10
11  :subobjects(
12    (sheet :class '(ui-spreadsheet-class)
13      measurement 'percentage
14      x-offset 5 y-offset 5 width 90 height 80
15      column-labels ``column-labels
16      row-labels (loop for row from 0 to ``input-list-length
17                    collect (format nil "~a ~a"
18                                   (``row-label-text)
19                                   row)
20                    )
21      cell-values ``input-list
22      number-of-columns ``number-of-columns
23      number-of-rows ``input-list-length
24      row-height 25
25      column-width 60
26      attachment-info-list '(top bottom left right)
27      editable? t
28    )
29    (toolbar :class ui-toolbar-class
30      measurement 'percentage
31      height 10 width 20 y-offset 90 x-offset 70
32      Button1-action-list '(
33        (apply-button-pressed (the superior superior))
34        (add-row (the superior superior))
35        (remove-row (the superior superior))
36      )
37      Labels-list (list "Apply" "Add node" "Remove node")
38      Number-of-buttons (length `Button1-action-list)
39      Orient (nth 0 '(horizontal vertical))
40    )
41  )

```

Figure 6.5: Annotated code for the *table-widget-class*. (1) Widget form definition. (2) Table definition. (3) Button definition. (4) Methods to be called when buttons is pressed.

```

1  (defun display-keypoint-widget ()
2    (if (subobject? (the interface forms) 'point-editor) (delete-object (the interface forms point-editor)))
3    (add-object (the interface forms) 'point-editor 'file-editor-class
4      :init-form '(
5        type 'points
6        label "Edit Keypoints"
7        input-list (get-points-list (the main-mechanism-class points))
8        column-labels(list "Name" "x-coord" "y-coord" "z-coord")
9        row-label-text "Node"
10       )
11    )
12    (display (the interface forms point-editor))
13    (undraw (the main-mechanism-class))
14    (loop for label in (get-node-labels (the main-mechanism-class points node-label)) do
15      (draw label)
16    )
17    (loop for point in (children (the points))
18      do (draw point :draw-subobjects? nil))
19  )

```

Figure 6.6: Code for the function instantiating the Keypoint Widget, within the red rectangle, keypoint widget properties are defined.

6.4 Mechanism Library Control

To ensure the ease of using RaMMS an option to save the current mechanism design was added, and as a consequence, some alterations were made to the structure of input files.

A *save*-button has been added to main GUI menu of RaMMS, by pressing the button, a save function is called. The save function writes all the lists of the collection-classes to input-files, saving the current mechanism design to input files in the mechanism library.

When testing the new save function, multiple bugs occurred. The application struggled to keep track on the number of white spaces in the input files, consequently, to reduce errors, a change was made to the input file style: instead of delimiting on tab, they now delimit on comma. Further, did weight points get their own input file, for easier differentiation between weight points, and keypoints. However, these modifications to input files is of little significance to the user, as the new UI removes the need to edit input files manually.

EXAMPLES OF MECHANISM DESIGN

To demonstrate the functionality of the new user interface, two mechanism designs were created using the updated user interface of RaMMS. First, the *Hoeken's linkage*, which has a simple planar geometry, was defined and iterated upon, to demonstrate the usage of table widgets for design iterations.

And, a double wishbone suspension, with a more complex spatial geometry, to demonstrate the use of the *Shapes Editor*.

7.1 Case 1: *Hoeken's linkage*

The planar *Hoeken's linkage* (fig. 2.3) was created in RaMMS and analysed in FEDEM. The mechanism input were created from the table widgets, keypoints in figure 7.1; constraints in figure 7.2; loads in figure 7.3; springs and dampers in figure 7.4.

After the geometry generation, the mechanism was meshed with a solid mesh (fig. 7.5), and exported to FEDEM (fig. 7.6). A small analysis of the mechanism's straight line motion was conducted by plotting the X versus Y coordinates of the coupler's end (fig. 7.7).

Further was the model modified, using the keypoints widget (fig. 7.8), and analysed again in FEDEM. With a different motion compared to the straight line motion of the original mechanism, the plot of the motion is presented in figure 7.9.

Chapter 7. Examples of Mechanism Design

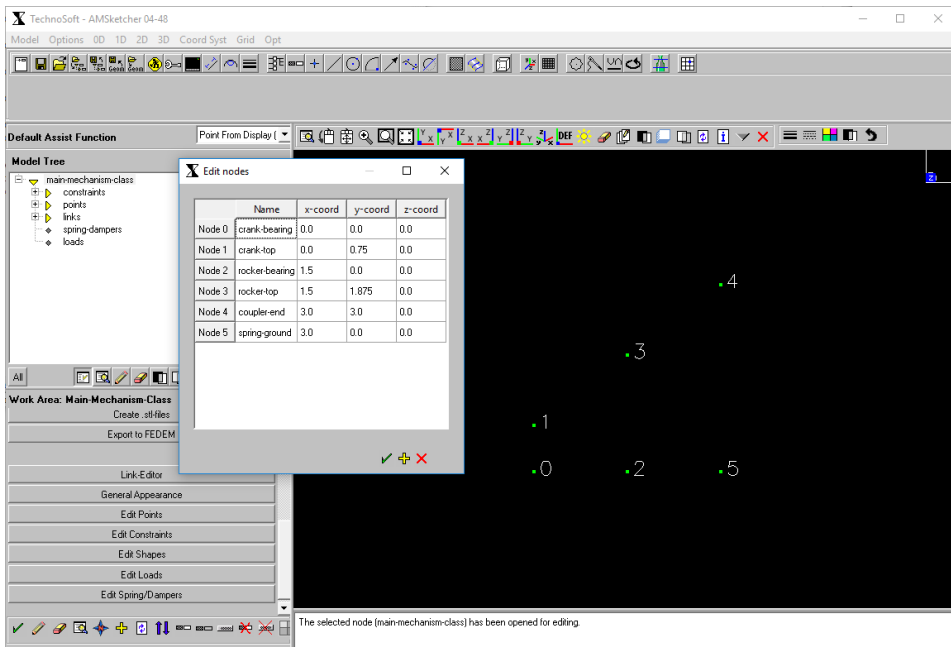


Figure 7.1: Defining keypoints for the *Hoeken's linkage* using the keypoint table widget.

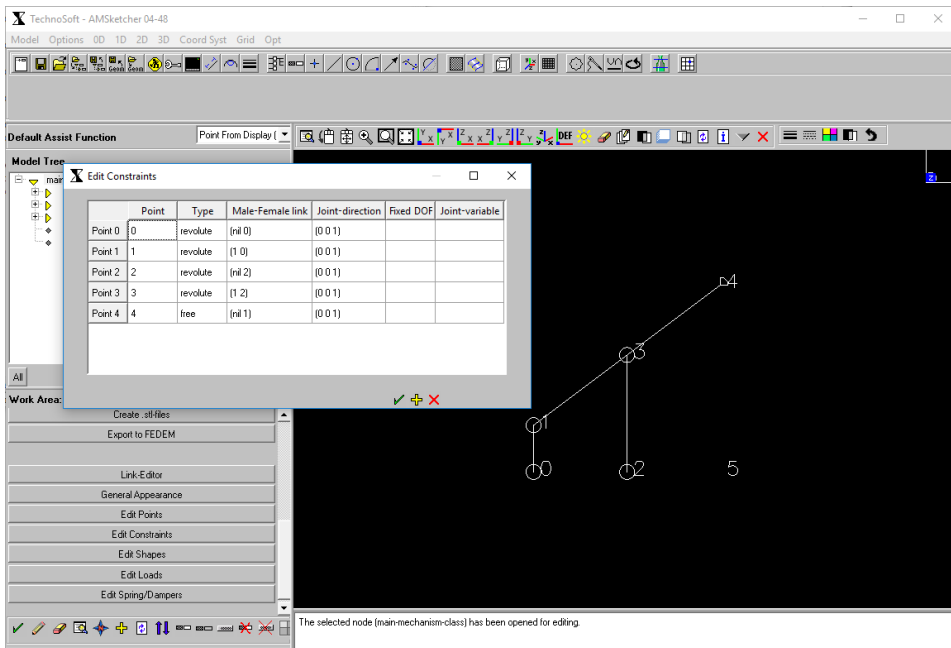


Figure 7.2: Defining constraints and links for the *Hoeken's linkage* using the constraints table widget.

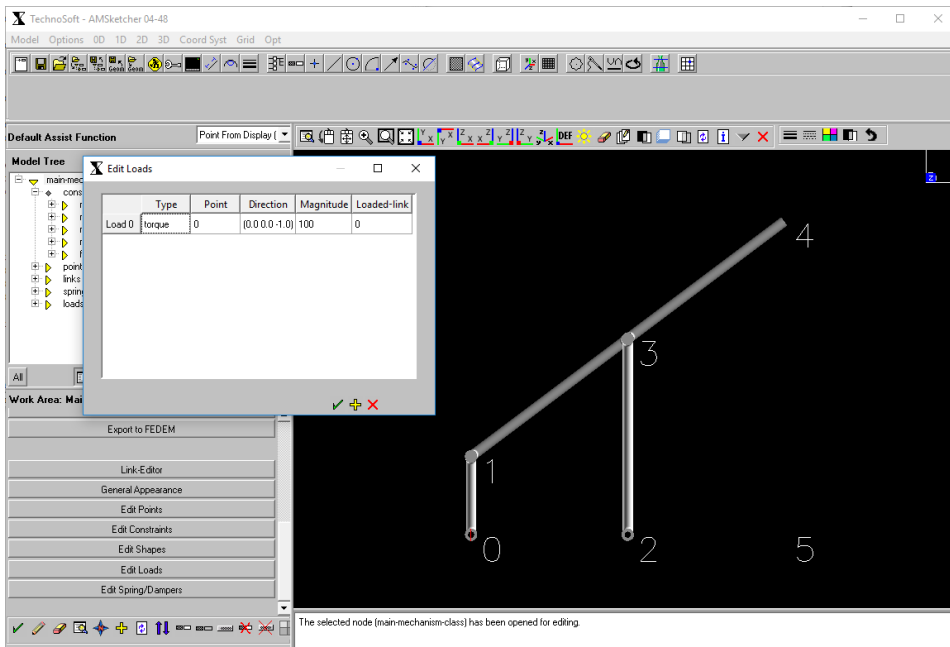


Figure 7.3: Defining a load for the *Hoeken*' linkage with the load table widget.

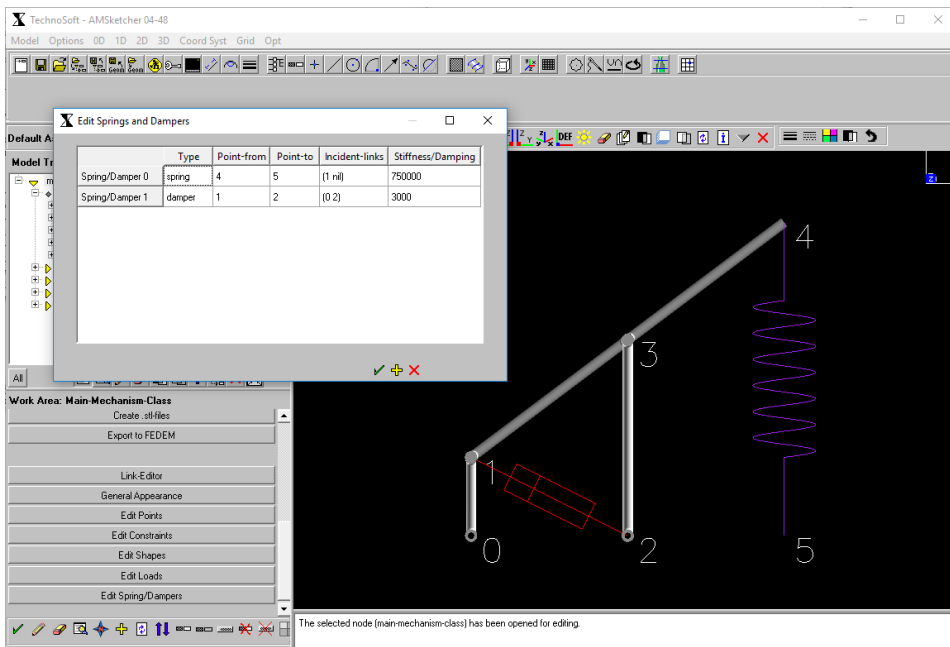


Figure 7.4: Defining one spring and one damper for the *Hoeken*' linkage, using the spring- and damper table widget.

Chapter 7. Examples of Mechanism Design

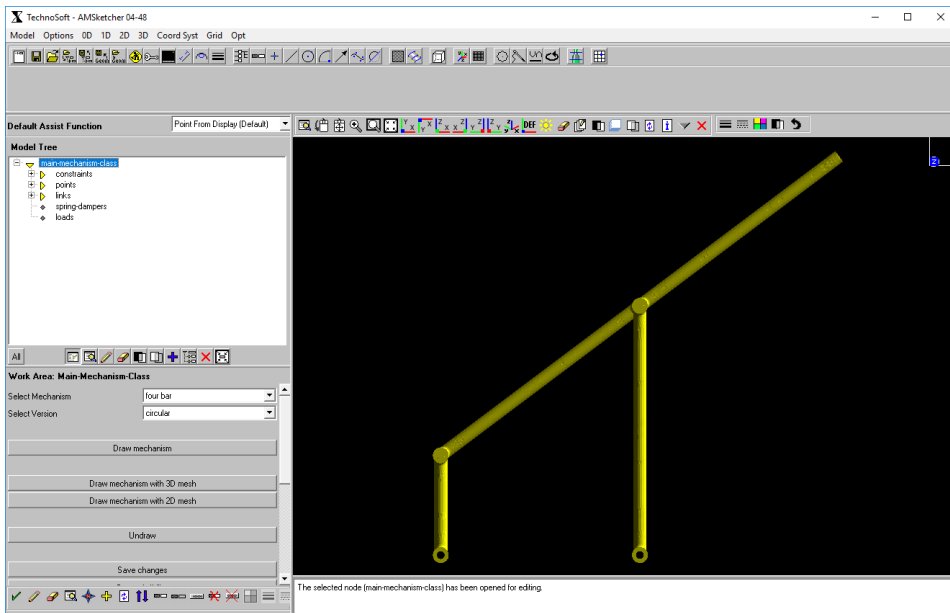


Figure 7.5: Mesh of the *Hoeken's linkage*.

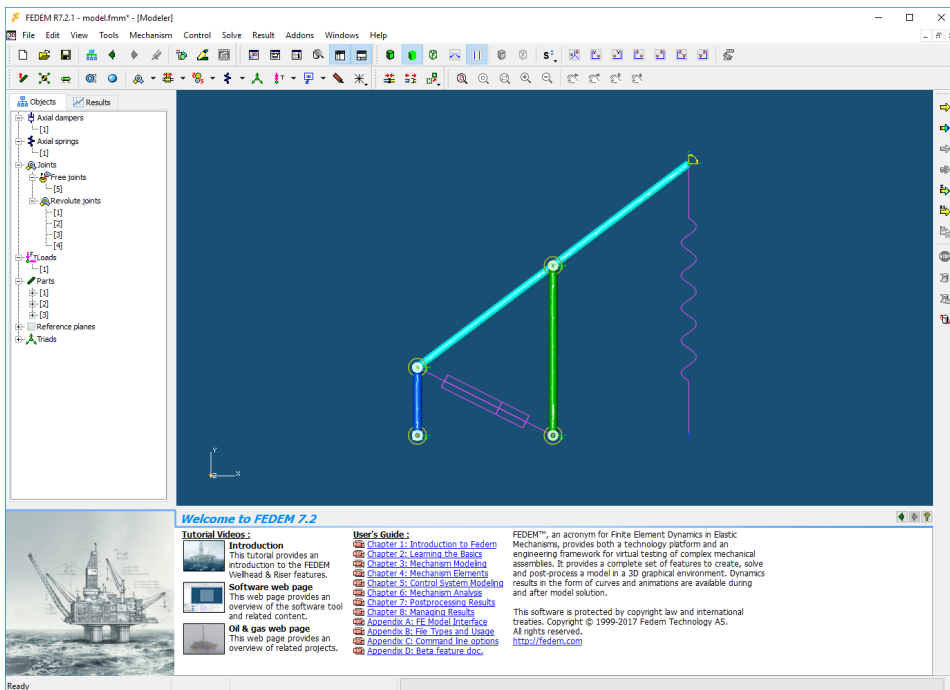


Figure 7.6: The *Hoeken's linkage* exported to FEDEM, with no errors.

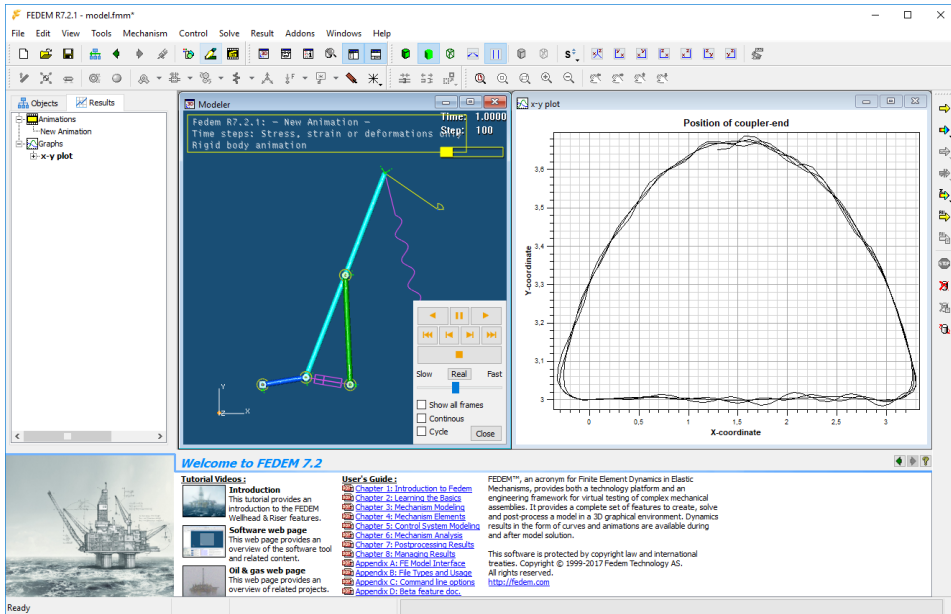


Figure 7.7: Analysis of the *Hoeken*' linkage in FEDEM. Showing a plot of the straight line motion, by plotting the x and y coordinates of the *coupler end*.

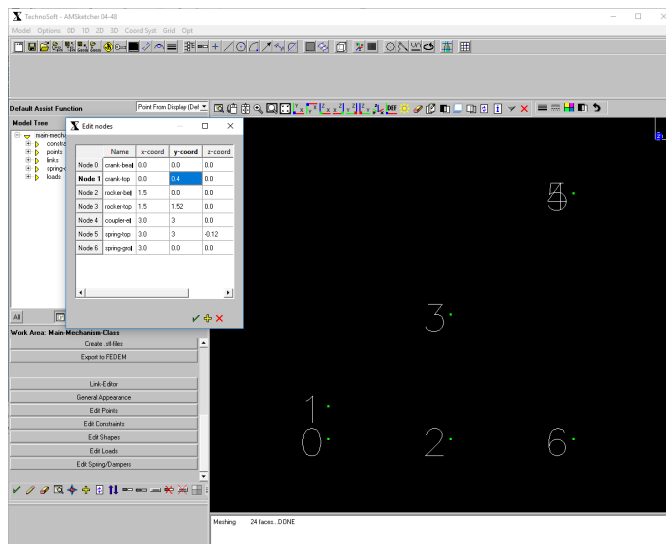


Figure 7.8: Modifying the *Hoeken*' linkage with the keypoint widget.

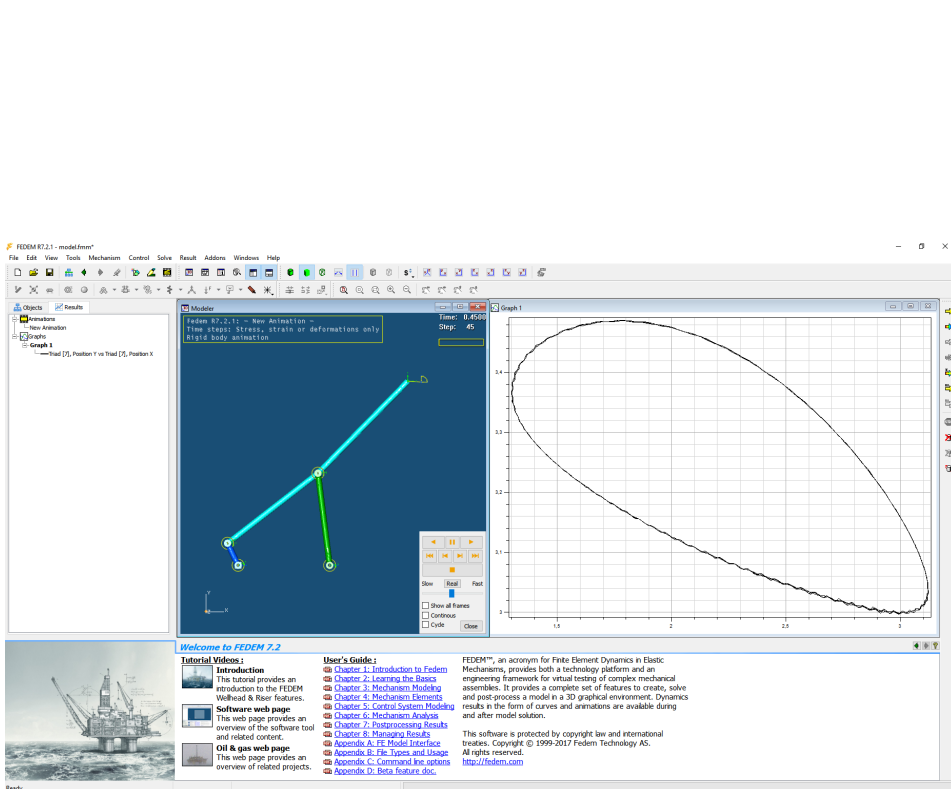


Figure 7.9: Analysis of the modified *Hoeken's linkage* in FEDEM. Displaying the new motion of the coupler's end.

7.2 Case 2: Double Wishbone Suspension

The spatial mechanism of a double wishbone suspension was created in RaMMS. Keypoint definitions are presented in figure 7.10, constraints in figure 7.11. Due to the abnormal form of the middle link's default link substructures, RaMMS failed to sweep its member geometries, errors are presented in figures 7.12 and 7.13. However, the link's can still be opened and its curves drawn in the *Shapes Editor*. They were, and the design was edited, so that it could be swept. The upper link's shapes were edited in the shapes editor, starting with straight lines (fig. 7.14), and finishing with a specialised design, presented in figures 7.15 and 7.16. The shapes of the middle link were modified to fit its correct design (fig. 7.17), and the cross section of the upper suspension arm and lower lower suspension arm member were altered to imitate the wheel mount (fig. 7.18), the middle revolute joint's dimensions is calculated from the end cross section of its biggest member. The final double wishbone design's geometry, ready for meshing and analysis, is presented in figure 7.19, and the mesh is in figure 7.20, finally, the model was exported to FEDEM, figure 7.21, with joint definitions.

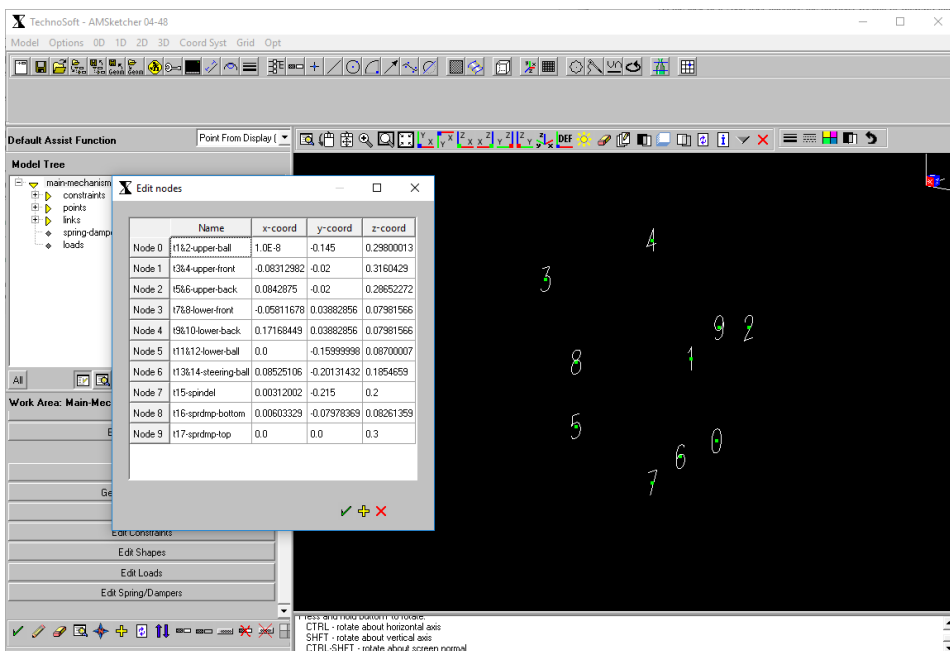


Figure 7.10: Defining keypoint coordinates for the double wishbone suspension, using the keypoints table widget.

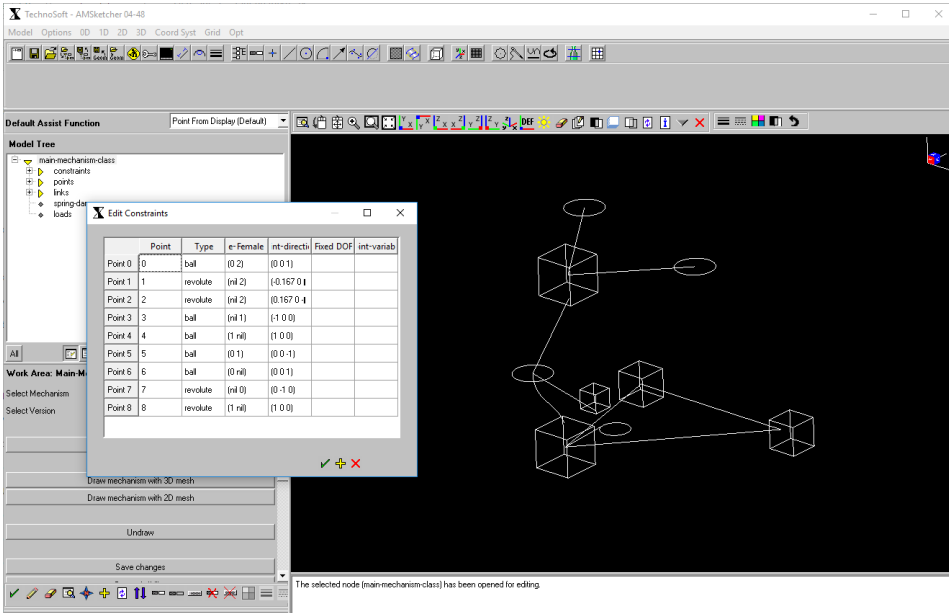


Figure 7.11: Defining constraints for the double wishbone suspension, using the constraints table widget.

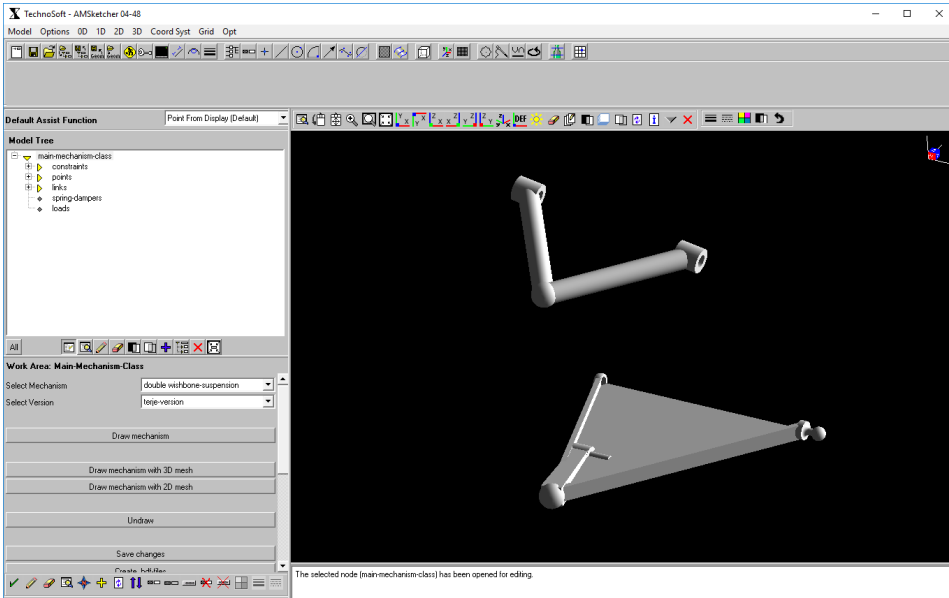


Figure 7.12: RaMMS failed to sweep the middle part of the double wishbone suspension.

```

*aml*
Retrieved lights parameters from file C:\Program Files\Technosoft\AML\AML6.31.1_x64\modules\amsketcher-module-04-48\sketcher-system\sketcher-general-info-file_s\lights.lit
"startup-done"
nil
>
> NULL GEOM ==>1704210c
NULL GEOM ==> 17041f0c
NULL GEOM ==>17041d0c
NULL GEOM ==>1703008c
NULL GEOM ==>1703d18c

Raw--*-XEmacs: *aml* (ILISP :ready)----Bot-----
Finished initializing aml

```

Figure 7.13: AML error when trying to sweep the geometry for the middle link of the double wishbone suspension: NULL GEOM.

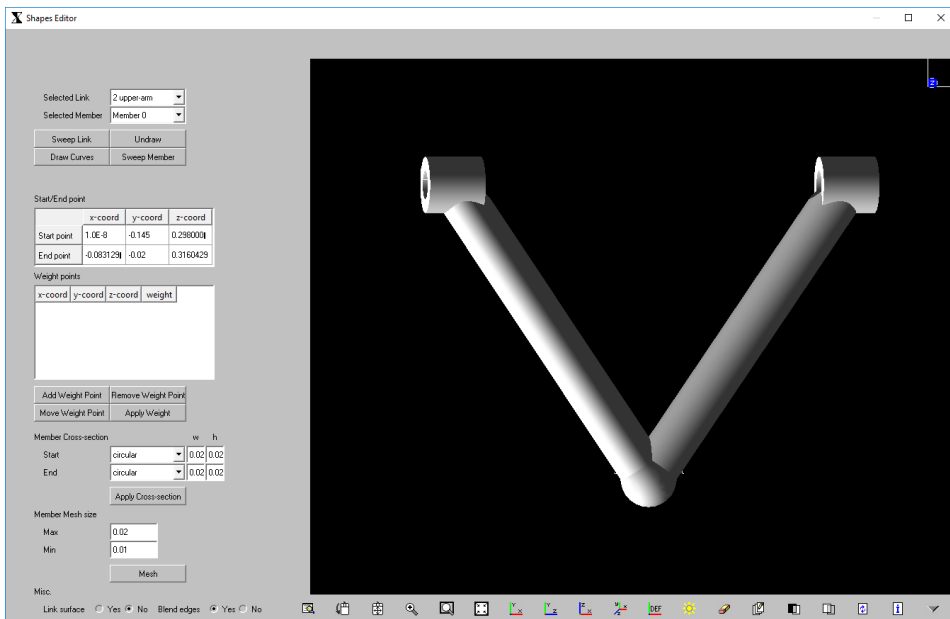
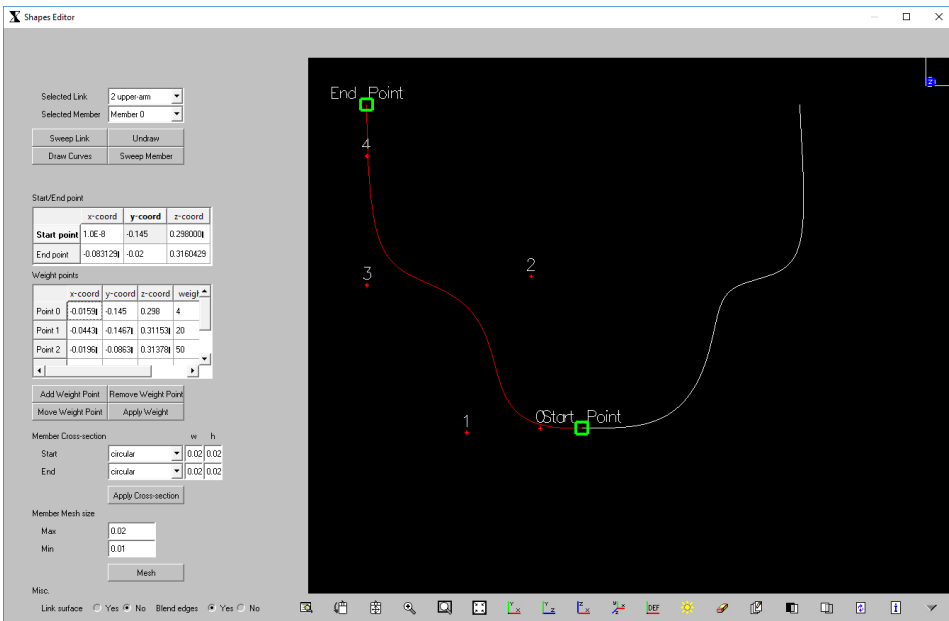


Figure 7.14: The upper link of the double wishbone suspension in the shape editor, prior to editing.



F

Figure 7.15: Modified NURBS curve of the upper link of the double wishbone suspension, displaying the weight points for forming of the left member of the link.

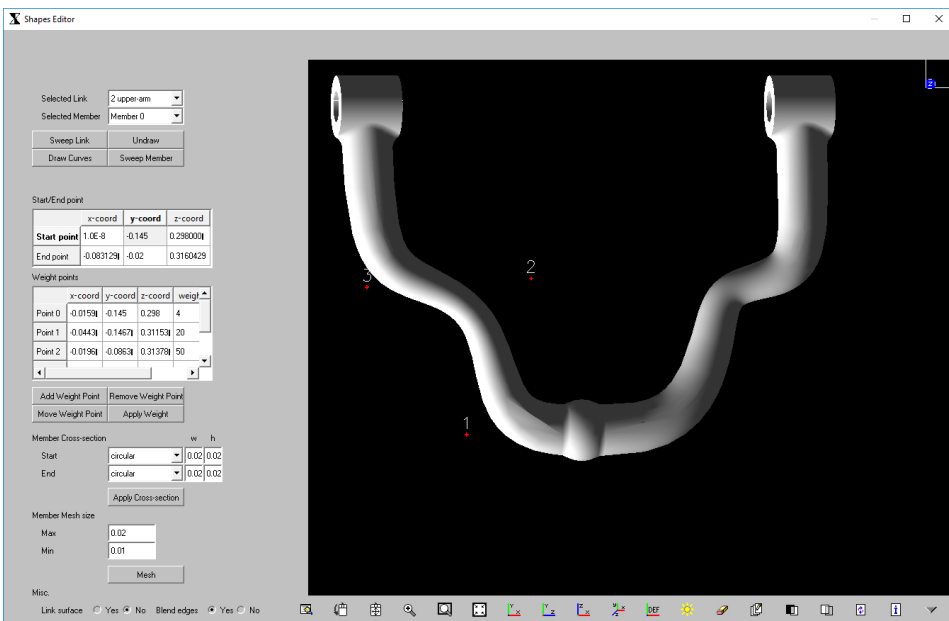


Figure 7.16: Sweep of a modified upper link of a double wishbone suspension.

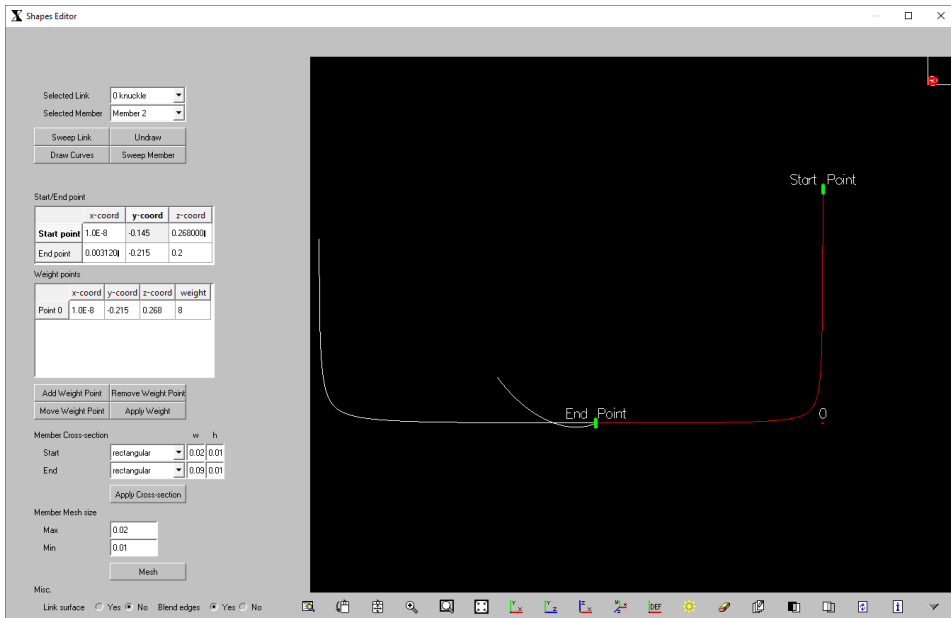


Figure 7.17: Modified NURBS curve of the middle link of the double wishbone suspension, displaying the single weight point to create the form of the lower member of the link.

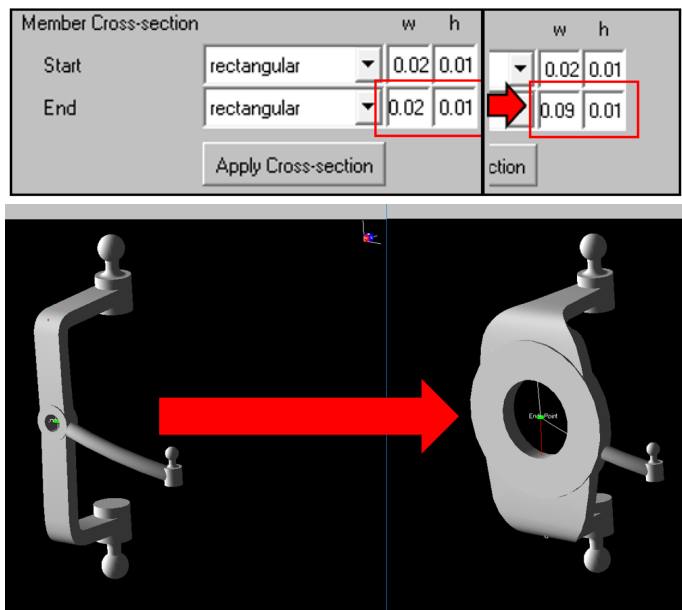


Figure 7.18: The wheel mount is imitated by varying the start and end cross section dimension for the members on the middle link of the double wishbone suspension.

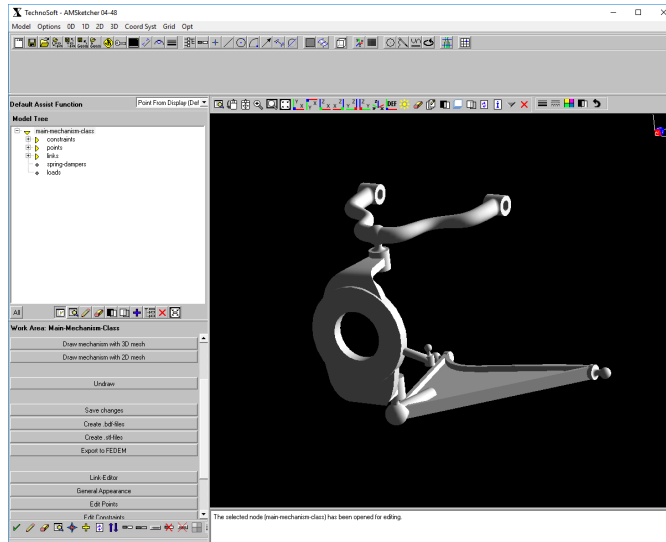


Figure 7.19: The geometric model of the double wishbone suspension after modifications in the shapes editor.

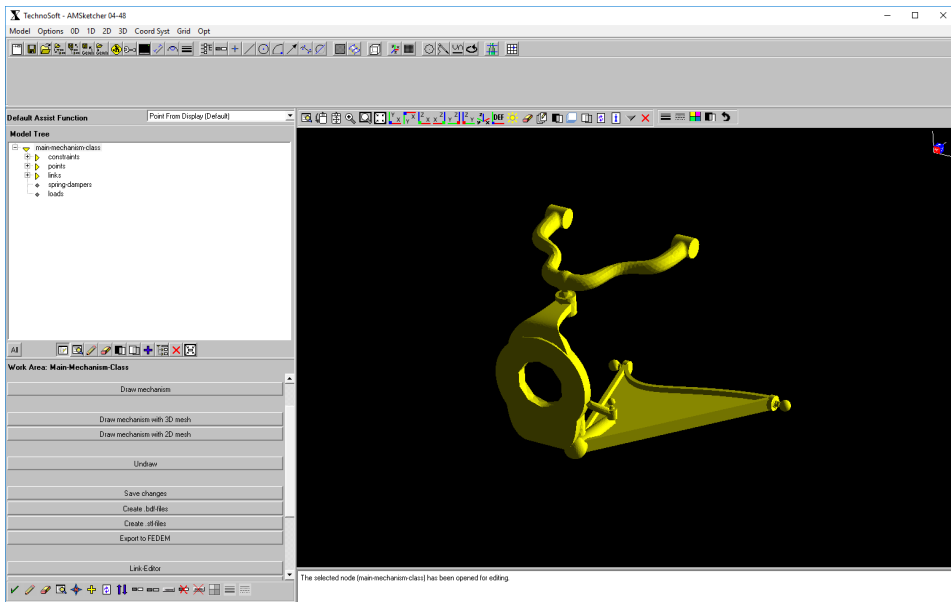


Figure 7.20: The double wishbone suspension is meshed in AML.

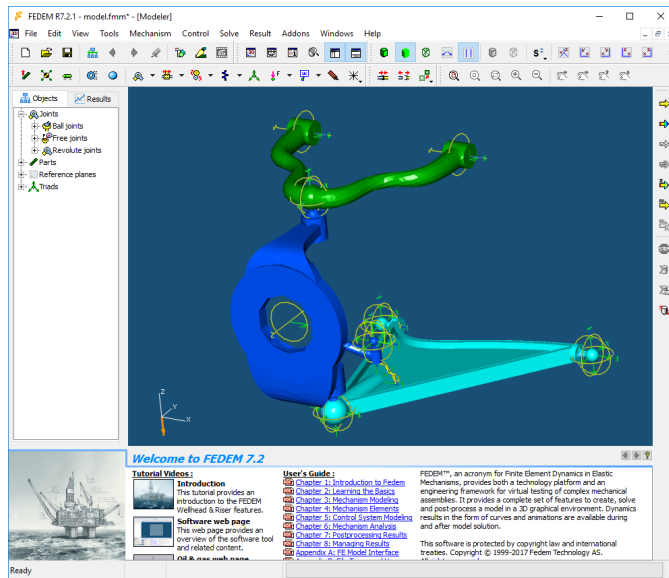


Figure 7.21: The double wishbone suspension in FEDEM.

DISCUSSION

The RaMMS UI has been updated, and it is interesting to discuss the application with the updated UI from a wide perspective, by reflecting on the following three topics: How RaMMS with the updated UI can be used in the design process; the new *Shapes Editor* adds CAD functionality to RaMMS, thus, comparing RaMMS to conventional CAD tools, and how it combines KBE and CAD; and finally, how the work-flow and overall usability of the updated application is.

8.1 RaMMS in the Design Process

Early in the design process few commitments have been made, and the design freedom is high (fig. 2.10), thus, an objective in product development, is to explore a large design space early in the design process. And, both KBE [26] and concurrent engineering [18] is reported to do this.

RaMMS is combining these two methodologies, by creating a bridge between geometry generation, meshing, mechanism knowledge and the multidisciplinary simulation tool FEDEM. Enabling for rapid generation and simulation of mechanism models, based on parametric mechanism topology as input. Consequently, by modifying the RaMMS input, various mechanism designs can be quickly generated and simulated, aiding the designer in quickly exploring a large design space.

However, the old version of RaMMS did not have any efficient way to modify input, thus, restricting the user in utilising the full potential of RaMMS, resulting in exploration of a smaller design space. The UI concepts presented in this thesis are solutions developed with the aim to solve this *bottleneck*.

And, the *table widgets*, presented in chapter 5, is made with the intention of enabling rapid modification of input, by quickly giving the user an overview of mechanism topology, and presenting an easy way to edit it.

While, the aim of the, *Shapes Editor* module, presented in chapter 5.1.3 and implemented in chapter 6.2, is not to speed up the iterations, but, rather to expand the usage of RaMMS, further into the design process. The module is a mini CAD module, built on top of RaMMS, containing many detailed features, allowing for detailed design to be applied to mechanisms. Thus, opening for an even more concurrent design process, where detailed design also can be explored at an early stage.

8.2 CAD v. RaMMS

An obvious benefit of using KBE instead of CAD is the time massive saving [25, 26, 22]. While, another benefit is that the KBE generated models are *knowledge based*, meaning that KBE generated models are rooted in product knowledge. While, CAD models may also be based on intricate product knowledge, the models are merely visualisations of a product, where the designer's product knowledge and reasoning is not attached to the model.

The mechanism models generated by RaMMS is a good example of knowledge based models. A RaMMS model is more than its geometric visualisation, mechanism specific knowledge is attached to the geometric model, and each mechanism part is given real life properties, e.g. a joint in RaMMS is represented as a joint would have been in the real world, with constraints telling it how to move and which links to attach. Together with application specific knowledge, this way of modelling allows KBE applications to create a bridge between various engineering tools, transferring models, without the need of manually defining the same properties several times.

The *Shapes Editor* introduces CAD functionality into RaMMS, to aid designers in adding detailed design to mechanisms. Which is a very good thing in the perspective of the design process, and concurrent engineering. But, not necessarily in the perspective of KBE, where everything strives to be knowledge based.

RaMMS is a KBE application, and should continue to be a KBE application. Mechanism models should still be generated based on knowledge, and real life properties should still be attached to the models. The *Shapes Editor* brings in design elements not following these principles.

However, the introduced CAD functionality are used for detailed design of members, where links still have fixed start and end points, thus, changes made with the *Shapes Editor* will not affect the overall mechanism behaviour, only substructure shapes are altered. Mechanism knowledge is still attached to models.

Consequently, the *Shapes Editor* can be viewed as a supplement to the KBE application, for adding detailed design, which would normally be added by a conventional CAD tool, later in the design process.

8.3 Usability

The objective of the updated UI was to enhance the manual optimisation process. To do this, UI concepts focusing on the *key-attributes*, efficiency and familiarity, were developed and implemented.

To verify the workflow and usability of a UI, requires extensive testing, involving numerous users [27], in the case of this work, such testing was not a valid option. However, throughout the development process user interface solutions and mock-ups have been discussed with a small group of people with experience in mechanism design, as well as computer tools for engineering design, and frequently asking the questions: *are we doing the right thing?* And *are we doing it correctly?* Thus, the presented concepts should be seen as qualitative suggestions for how a UI trying to fulfill its objectives should be.

Moreover, after simple testing of the UI's functionality, the implemented *table widgets*, proved as great tools for iterating on mechanism designs from within the RaMMS application, as demonstrated by the analysis and quick modification of the *Hoeken's linkage* in chapter 7.1, and, the *Shapes Editor* enables more advanced mechanisms to be created, e.g. the double wishbone suspension in chapter 7.2. Whereas, using RaMMS to iterate on designs of such advanced spatial mechanisms is more difficult. It is hard to use the table widgets and interactive *dragging* of points to do small alterations of keypoints in a three-dimensional environment, and the UI does not offer any functionality to aid the user in doing this.

Overall, the updated UI presents an better way to create both simple planar and more advanced spatial mechanisms, and offers a way to quickly iterate over simple mechanisms, however, when the mechanisms become too advanced, the UI struggles to aid the user in quickly modifying designs.

CONCLUSION

This thesis investigates the mechanism design process, KBE, as well as UI design and implementation.

User interface concepts for the RaMMS KBE application was developed and discussed using mock-ups. Solutions were implemented to the application, using the principles of the model-view-controller design pattern, and combining AML's predefined user interface classes.

A special focus was given to implement a *Shapes Editor*, i.e. a mini CAD module built on top of the RaMMS application, for conducting detailed link design. This module adds CAD functionality to the RaMMS, enabling it to be used in a larger part of the design process.

The user interface has improved the RaMMS, and with the updated UI it provides a good tool for iterating over mechanism design.

BIBLIOGRAPHY

- [1] A. Corallo, R. Laubacher, A. Margherita, and G. Turrisi. Enhancing product development through knowledge-based engineering (KBE): A case study in the aerospace industry. *Journal of Manufacturing Technology Management*, 20(5):1070–1083, 2009. URL: <http://dx.doi.org/10.1108/17410380910997218>.
- [2] S. Danjou, N. Lupa, and P. Koehler. Approach for Automated Product Modeling Using Knowledge-Based Design Features. *Computer-Aided Design and Applications*, 5(5):622–629, 2008-01. ISSN: 1686-4360. DOI: 10.3722/cadaps.2008.622-629. URL: <http://www.tandfonline.com/doi/abs/10.3722/cadaps.2008.622-629>.
- [3] R. K. Skaare. *Mechanism parametrization, modeling and FE-meshing*. Master's thesis, Norwegian University of Science and Technology, Trondheim, 2015-06.
- [4] A. K. Kristiansen and E. Kristoffersen. *Automating tasks in the design loop for mechanism design*. Master's thesis, Norwegian University of Science and Technology, Trondheim, 2016-06.
- [5] T. C. Coward. Automation (KBE) in model generation for dynamic simulation. Project Work, Norwegian University of Science and Technology. Unpublished Work, 2016.
- [6] F. Reuleaux. *The Kinematics of Machinery: Outlines of a Theory of Machines*. Macmillan, London, 1876. ISBN: 1143576519. URL: <http://historical.library.cornell.edu/kmoddl/index.html#kennedy3>.
- [7] R.S. Hartenberg and J. Denavit. *Kinematic Synthesis of Linkages*. McGraw-Hill, 1964, page 133. ISBN: 0070269106. DOI: 10.1115/1.3609993.
- [8] R. Norton. *Design of Machinery (McGraw-Hill Series in Mechanical Engineering)*. McGraw-Hill Education, 2011. ISBN: 007742171X.
- [9] P. L. Chebyshev. *On transformation of rotary movement into movement along some lines using joined systems / After: The complete works of P. L. Tchebyshev. Vol. IV. Theory of mechanisms*. USSR Academy of Sciences, 1948, pages 161–166. URL: http://www.tcheb.ru/docs/pdf/TchebRu_cw4_161-166.pdf.
- [10] K. Hoeken. *Steigerung der Wirtschaftlichkeit durch zweckmässige Anwendung der Getriebelehre, Werkstattstechnik*. IG-Farbenindustrie AG, 1926. URL:

-
- <https://www.deutsche-digitale-bibliothek.de/item/J3N6CM46D2QTZNV336J3UL5OUQKK4SOY>.
- [11] S. Molian. *Mechanism Design*. Pergamon, 1997. ISBN: 0080422640.
 - [12] A. Rodriguez and M. T. Mason. Effector form design for 1DOF planar actuation. In *2013 IEEE International Conference on Robotics and Automation*, pages 349–356. IEEE, 2013-05. ISBN: 978-1-4673-5643-5. DOI: 10.1109/ICRA.2013.6630599.
 - [13] S. Lu, D. Zlatanov, X. Ding, and R. Molfino. A new family of deployable mechanisms based on the Hoekens linkage. *Mechanism and Machine Theory*, 73:130–153, 2014. DOI: 10.1016/j.mechmachtheory.2013.10.007. URL: <http://www.sciencedirect.com/science/article/pii/S0094114X13002139>.
 - [14] J.J. Breen and M.A. Hayner. Actuator including mechanism for converting rotary motion to linear motion, 2013-01. URL: <https://www.google.com/patents/US8360387>. US Patent 8,360,387.
 - [15] C. Knabe, B. Lee, and D. Hong. An Inverted Straight Line Mechanism for Augmenting Joint Range of Motion in a Humanoid Robot. In *Volume 5B: 38th Mechanisms and Robotics Conference*, V05BT08A015. ASME, 2014-08. ISBN: 978-0-7918-4637-7. DOI: 10.1115/DETC2014-35123. URL: <http://proceedings.asmedigitalcollection.asme.org/proceeding.aspx?doi=10.1115/DETC2014-35123>.
 - [16] J. J. Uicker, G. R. Pennock, J. E. Shigley, and J. M. McCarthy. *Theory of Machines and Mechanisms*, 2003. DOI: 10.1115/1.1605769.
 - [17] W. C. Hurty. Dynamic analysis of structural systems using component modes. *AIAA Journal*, 3:678–685, 1965-04. DOI: 10.2514/3.2947.
 - [18] O. I. Sivertsen. *Virtual testing of Mechanical Systems*. Swets & Zeitlinger, Trondheim, Norway, 2001.
 - [19] C. B. Chapman and M. Pinfold. Design engineering - a need to rethink the solution using knowledge based engineering. *Knowledge-Based Systems*, 12(5-6):257–267, 1999. ISSN: 09507051. DOI: 10.1016/S0950-7051(99)00013-1. URL: <http://www.sciencedirect.com/science/article/pii/S0950705199000131>.
 - [20] B. S. Blanchard and W. J. Fabrycky. *Systems Engineering and Analysis (3rd Edition)*. Prentice Hall, 1998. ISBN: 0131350471.
 - [21] B. Prasad. Sequential versus Concurrent Engineering—An Analogy. *Concurrent Engineering*, 3(4):250–255, 1995-12. ISSN: 1063-293X. DOI: 10.1177/1063293X9500300401. URL: <http://cer.sagepub.com/cgi/doi/10.1177/1063293X9500300401>.
 - [22] G. La Rocca. Knowledge based engineering: Between AI and CAD. Review of a language based technology to support engineering design. *Advanced Engineering Informatics*, 26(2):159–179, 2012. ISSN: 14740346. DOI: 10.1016/j.aei.2012.02.002.
 - [23] M. Pinfold and C. Chapman. Application of KBE techniques to the FE model creation of an automotive body structure. *Computers in Industry*, 44(1):1–10, 2001. ISSN: 01663615. DOI: 10.1016/S0166-3615(00)00079-8.
-

-
- [24] H. Z. Yang, J. F. Chen, N. Ma, and D. Y. Wang. Implementation of knowledge-based engineering methodology in ship structural design. *CAD Computer Aided Design*, 44(3):196–202, 2012-05. ISSN: 00104485. DOI: 10.1016/j.cad.2011.06.012. URL: <http://ade.sagepub.com/lookup/doi/10.1177/1687814015584239>.
- [25] W. Skarka. Application of MOKA methodology in generative model creation using CATIA. *Engineering Applications of Artificial Intelligence*, 20:677–690, 2007. DOI: 10.1016/j.engappai.2006.11.019.
- [26] W. J. C. Verhagen, P. Bermell-Garcia, R. E. C. Van Dijk, and R. Curran. A critical review of Knowledge-Based Engineering: An identification of research challenges. *Advanced Engineering Informatics*, 26(1):5–15, 2012. ISSN: 14740346. DOI: 10.1016/j.aei.2011.06.004.
- [27] B. Shneiderman and C. Plaisant. *Designing the user interface : strategies for effective human-computer interaction*. Addison-Wesley, 2010, page 606. ISBN: 9780321537355.
- [28] J. D. Gould. How to Design Usable Systems. In *Handbook of Human-Computer Interaction*, pages 757–789. Elsevier, 1988. DOI: 10.1016/B978-0-444-70536-5.50040-3. URL: <http://linkinghub.elsevier.com/retrieve/pii/B9780444705365500403>.
- [29] W. O. Galitz. *The essential guide to user interface design : an introduction to GUI design principles and techniques*. Wiley Pub, 2007, page 857. ISBN: 0470053429.
- [30] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994. ISBN: 9780321700698.
- [31] T. M. Reenskaug. Thing-model-view-editor an example from a planning system, 1979.
- [32] TechnoSoft Inc. *AML Basic Training Manual*. TechnoSoft Inc., v.3.06 edition, 2007.

APPENDIX

A

APPENDIX

A.1 AML UI Classes

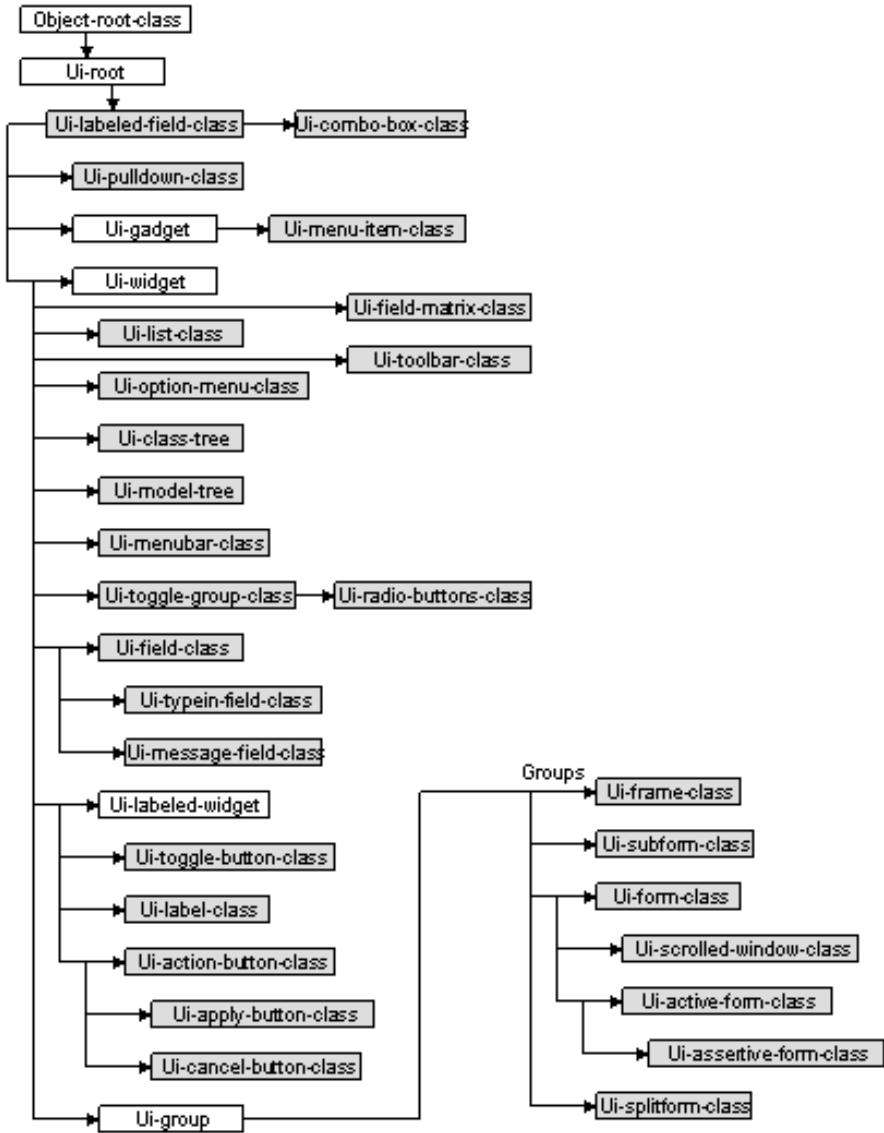


Figure A.1: Hierarchy of AML GUI base classes. The filled boxes represent classes that can be directly instantiated by the user.

Listing A.1: The AML code to create a box model widget

```
1 (define-class box-model-form
2   :inherit-from(ui-form-class)
3
4   :properties(
5     x-offset 50 y-offset 50 height 280 width 250 label "Box Model"
6     measurement 'percentage
7   )
8   :subobjects(
9     (bdepth :class 'ui-labeled-field-class
10      x-offset 0 y-offset 0 width 100 height 10
11      label "Depth"
12      content (if ^^current-model (the depth (:from ^^current-model)) "N/A")
13      apply-action (when ^^current-model '(change-value
14        (the depth (:from ^^current-model))
15        (get-value (the superior))))
16      cancel-action '(smash-value ^content))
17     (bheight :class 'ui-labeled-field-class
18      x-offset 0 y-offset 10 width 100 height 10
19      label "Height"
20      content (if ^^current-model (the height (:from ^^current-model)) "N/A")
21      apply-action (when ^^current-model
22        '(change-value
23          (the height (:from ^^current-model))
24          (get-value (the superior))))
25      cancel-action '(smash-value ^content)
26      )
27     (bwidth :class 'ui-labeled-field-class
28      x-offset 0 y-offset 20 width 100 height 10
29      label "Width"
30      content (if ^^current-model (the width (:from ^^current-model)) "N/A")
31      apply-action (when ^^current-model
32        '(change-value
33          (the width (:from ^^current-model))
34          (get-value (the superior))))
35      cancel-action '(smash-value ^content)
36      )
37     (solid? :class 'ui-toggle-button-class
38      x-offset 0 y-offset 30 width 100 height 10
39      label "Solid"
40      status (when ^^current-model (the solid? (:from ^^current-model)))
41      apply-action (when ^^current-model
42        '(change-value
43          (the solid? (:from ^^current-model)) ^status))
44      cancel-action '(smash-value ^status)
45      )
46     (render :class 'ui-radio-buttons-class
47      x-offset 0 y-offset 40 width 100 height 10
48      labels-list '("Wire" "Shaded" "Isoline")
49      status (when ^^current-model
50        (case (the render (:from ^^current-model))
51          ('boundary 0)
52          ('shaded 1)
53          ('isoline 2)))
54      apply-action (when (and ^^current-model ^status)
55        '(change-value
56          (the render (:from ^^current-model))
57          (nth ^status (list 'boundary 'shaded 'isoline))))
58      cancel-action '(smash-value ^status))
59     (apply :class 'ui-apply-button-class
60      x-offset 0 y-offset 90 width 25 height 10)
61     (cancel :class 'ui-cancel-button-class
62      x-offset 25 y-offset 90 width 25 height 10
63      update-form? t)
64     (draw :class 'ui-action-button-class
65      x-offset 50 y-offset 90 width 25 height 10
66      label "Draw"
67      button1-action (when ^^current-model
68        '(draw ^^current-model))
69      button3-action (when ^^current-model
70        '(undraw ^^current-model)))
71     (close :class 'ui-action-button-class
72      x-offset 75 y-offset 90 width 25 height 10
73      label "Close"
74      button1-action '(hide (the superior superior)))
75   )
76 )
```

A.2 NURBS Module Source Code

Listing A.2: NURBS module AML code

```
1 ;To display the nurbs module run the following function:
2 ;(display-nurbs-module)
3 (in-package :aml)
4 ;;;; Model ;;;;
5 (define-class point-collection
6   :inherit-from (object)
7   :properties (
8     (ref-coord-sys :class coordinate-system-class
9       origin (list 0.0 0.0 0.0)
10      )
11     points-list '((0.0 0.0 0.0) (7.0 0.0 0.0))
12     weight-points '((1.0 -1.0 0.0 1.0) (2.0 1.0 0.0 1.0) (3.0 2.0 0.0 1.0) (4.0 1.0 0.0 1.0) (5.0 0.0
13      0.0 1.0) (6.0 -2.0 0.0 1.0))
14   )
15   :subobjects (
16     (nurbs-curve :class nurbs-curve-object
17       reference-coordinate-system ^^ref-coord-sys
18       start-point (first ^^points-list)
19       end-point (nth 1 ^^points-list)
20       start-weight (list (append ^start-point (list 1)))
21       end-weight (list (append ^end-point (list 1)))
22       points (append ^start-weight ^^weight-points ^end-weight)
23       rational? t
24       homogeneous? t
25     )
26   )
27 (define-method get-points-list point-collection ()
28   !points-list
29 )
30 (define-method get-weight-points point-collection ()
31   !weight-points
32 )
33 (define-method get-ref-coord-sys point-collection ()
34   !ref-coord-sys
35 )
36 (define-method smash-points-method point-collection ()
37   (smash-value !points-list)
38 )
39 (define-method change-points-list point-collection (new-points-list)
40   (change-value !points-list new-points-list)
41 )
42 (define-method change-weight-points point-collection (new-weight-points)
43   (progn
44     (smash-value !weight-points)
45     (change-value !weight-points new-weight-points)
46   )
47 )
48 ;;;; Controller ;;;;
49 (define-class nurbs-controller
50   :inherit-from (object)
51   :properties (
52     model-points (get-points-list (the point-collection))
53     model-weight (get-weight-points (the point-collection))
54     model-coord-sys (get-ref-coord-sys (the point-collection))
55   )
56 )
57 (define-method get-controller-points nurbs-controller ()
58   !model-points
59 )
60 (define-method get-controller-weight nurbs-controller ()
61   !model-weight
62 )
63 (define-method get-controller-coord-sys nurbs-controller ()
64   !model-coord-sys
65 )
66 (define-method change-controller-weight nurbs-controller (new-weight-points)
67   (change-value !model-weight new-weight-points)
68   (change-weight-points (the point-collection) !model-weight)
69 )
70 (define-method change-controller-points nurbs-controller (new-points)
71   (change-value !model-points new-points)
72   (change-points-list (the point-collection) !model-points)
73 )
74 (define-method change-controller-weight-coords nurbs-controller (point-number position)
75   (progn (change-value !model-weight (let
76     ((weight-points !model-weight)
77      (new-point (let ((l1 (nth point-number weight-points)) (l2 position)) (replace l1 l2
78      ) 11))
79     (replace (nth point-number weight-points) new-point)
80     (smash-value !model-weight)
81     weight-points
```

```

82         )
83     )
84     (change-weight-points (the point-collection) !model-weight)
85 )
86 )
87 ;;;; View ;;;;
88 (define-class helping-points
89   :inherit-from (object)
90   :properties(
91     points-list (get-controller-points (the nurbs-controller))
92     weight-points (loop for w in (get-controller-weight (the nurbs-controller))
93                          collect (butlast w)
94                          )
95     ref-coord-sys (get-controller-coord-sys (the nurbs-controller))
96   )
97   :subobjects(
98     (ui-end-points :class series-object
99       quantity (length ^points-list)
100      class-expression 'box-object
101      series-prefix 'p
102      init-form '(
103        id ^index
104        reference-coordinate-system ^ref-coord-sys
105        line-width 4
106        color 'green
107        height 0.03
108        width 0.03
109        depth 0.001
110        ;orientation (list (translate ^index 0.0 0.0))
111        orientation (list (translate (nth ^index ^points-list)))
112        ;coordinates (list (nth 0 (nth ^index ^points-list))
113                          (nth 1 (nth ^index ^points-list)) (nth 2 (nth ^index ^points-list)))
114      )
115     )
116     (ui-weight-points :class series-object
117       quantity (length ^weight-points)
118       class-expression 'box-object
119       series-prefix 'w
120       init-form '(
121         id ^index
122         reference-coordinate-system ^ref-coord-sys
123         line-width 4
124         color 'red
125         height 0.03
126         width 0.03
127         depth 0.001
128         ;orientation (list (translate ^index 0.0 0.0))
129         orientation (list (translate (nth ^index ^weight-points)))
130         ;coordinates (list (nth 0 (nth ^index ^points-list))
131                          (nth 1 (nth ^index ^points-list)) (nth 2 (nth ^index ^points-list)))
132       )
133     )
134   )
135 )
136 (define-method get-ui-weight-point-instances helping-points ()
137   (series-members (the ui-weight-points))
138 )
139 (define-method get-ui-weight-point-coords helping-points ()
140   !weight-points
141 )
142 (define-method translate-helping-points helping-points ()
143   (let (
144     (point-number (mouse-select-point-from-display !weight-points))
145     (point-instance (nth point-number (series-members (the ui-weight-points))))
146   )
147     (progn
148       (interactive-translate point-instance)
149       (change-controller-weight-coords (the nurbs-controller) point-number (nth 0 (the position (:from
150 point-instance))))
151     )
152   )
153 )
154 (define-class link-editor-form-class
155   :inherit-from(ui-form-class)
156   :properties(
157     ;background-color 'snow2
158     label "NURBS Module"
159     height (* 0.6 (nth 1 (get-screen-size)))
160     width (* 0.6 (nth 0 (get-screen-size)))
161     y-offset (- ^height (* 0.5 ^height))
162     x-offset (- ^width (* 0.5 ^width))
163     button-y-offset 95
164     weight-points (get-controller-weight (the nurbs-controller))
165     weight-points-quantity (length ^weight-points)
166     points-list (get-controller-points (the nurbs-controller))
167   )
168   :subobjects(
169     (canvas :class 'ui-canvas-class
170       measurement 'percentage
171       x-offset 30 y-offset 5 width 70 height 90

```

```

171 )
172 (graphic-toolbar :class 'ui-graphic-control-toolbar-class
173   canvas-object `canvas
174   measurement `percentage
175   height 5
176   width 70
177   x-offset 30
178   y-offset 95
179 )
180 (draw-curve-action-button :class 'ui-action-button-class
181   measurement `percentage
182   height 5
183   width 10
184   y-offset 7
185   x-offset 2
186   label "Draw Curve"
187   Button1-action '(progn (draw (the helping-points))
188                         (draw (the point-collection))
189                         (update (the superior link-editor-form-class))
190                         (update (the superior canvas))
191                         (regen)
192                         (zoom :all)
193                         )
194 )
195 (move-point-action-button :class 'ui-action-button-class
196   measurement `percentage
197   height 5
198   width 10
199   y-offset 7
200   x-offset 12
201   label "Drag point"
202   Button1-action '(progn (translate-helping-points (the helping-points))
203                         (update (the superior weight-sheet))
204                         (regen)
205                         (draw (the helping-points)))
206 )
207 (point-sheet :class 'ui-spreadsheet-class
208   measurement `percentage
209   x-offset 2
210   y-offset 40
211   width 21
212   height 12
213   column-labels (list "x-coord" "y-coord" "z-coord")
214   row-labels (list "Start point" "End point")
215   cell-values ``points-list
216   number-of-columns (length `column-labels)
217   number-of-rows 2
218   row-height 25
219   column-width 53
220   attachment-info-list '(top bottom left right)
221   editable? t
222 )
223 (weight-sheet :class 'ui-spreadsheet-class
224   measurement `percentage
225   x-offset 2
226   y-offset 60
227   width 22
228   height 30
229   column-labels (list "x-coord" "y-coord" "z-coord" "weight")
230   row-labels (loop for row from 0 to `weight-points-quantity
231                collect (format nil "Point ~a" row)
232                )
233   cell-values ``weight-points
234   number-of-columns (length `column-labels)
235   number-of-rows ``weight-points-quantity
236   row-height 25
237   column-width 50
238   attachment-info-list '(top bottom left right)
239   editable? t
240 )
241 (toolbar :class 'ui-toolbar-class
242   measurement `percentage
243   height 5
244   width 20
245   y-offset 95
246   x-offset 7
247   Availability-list nil
248   Button1-action-list '( (weight-sheet-apply (the superior link-editor-form-class))
249                         (weight-sheet-add-row (the superior link-editor-form-class))
250                         (weight-sheet-remove-row (the superior link-editor-form-class))
251                         )
252   Button3-action-list nil
253   Images-list (list (concatenate (logical-path :ui-bitmaps) "apply.bmp") (concatenate (logical-path :
254 ui-bitmaps) "add-object.bmp") (concatenate (logical-path :ui-bitmaps) "delete-object.bmp"))
255   Labels-list (list "Apply" "Add node" "Remove node")
256   Number-of-buttons (length `Button1-action-list)
257   Orient (nth 0 '(horizontal :vertical))
258   Tooltips-list nil
259 )

```

```

260 )
261 (define-method weight-sheet-apply link-editor-form-class ()
262   (progn
263     (change-controller-weight (the nurbs-controller) (loop for row from 0 to (- (the weight-sheet number-of-rows) 1)
264       collect (loop for cell from 0 to 3
265         collect (read-from-string (get-cell-value (the
266           weight-sheet) row cell))
267         )
268       )
269     (change-controller-points (the nurbs-controller) (loop for row from 0 to (- (the point-sheet number-of-rows) 1)
270       collect (loop for cell from 0 to 2
271         collect (read-from-string (get-cell-value (the
272           point-sheet) row cell))
273         )
274       )
275     (update (the weight-sheet))
276     (regen)
277     (draw (the helping-points))
278   )
279 )
280 (define-method weight-sheet-add-row link-editor-form-class ()
281   (progn
282     (change-controller-weight (the nurbs-controller) (append
283       number-of-rows 1)
284       (loop for row from 0 to (- (the weight-sheet
285         number-of-rows) 1)
286         collect (loop for cell from 0 to 3
287           collect (read-from-string (get-cell-value
288             (the weight-sheet) row cell))
289           )
290         (list '(0.0 0.0 0.0 1.0))
291       )
292     (update (the weight-sheet))
293     (regen)
294     (draw (the helping-points))
295   )
296 )
297 (define-method weight-sheet-remove-row link-editor-form-class ()
298   (progn
299     (change-controller-weight (the nurbs-controller) (butlast (loop for row from 0 to (- (the weight-sheet
300       number-of-rows) 1)
301       collect (loop for cell from 0 to 3
302         collect (read-from-string (get-cell-value (the
303           weight-sheet) row cell))
304         )
305       ))
306     (update (the weight-sheet))
307     (regen)
308     (draw (the helping-points))
309   )
310 )
311 (defun display-nurbs-module ()
312   (progn
313     (create-model 'point-collection)
314     (create-model 'nurbs-controller)
315     (create-model 'helping-points)
316     (add-object (the interface forms) 'link-editor 'link-editor-form-class)
317     (display (the interface forms link-editor))
318     (activate-display (the interface forms link-editor canvas))
319     ;(set-current-display-background-color 'grey49)
320   )
321 )

```

A.3 Shapes Editor Source code

Listing A.3: Shapes Editor AML code

```
1 (in-package :aml)
2 (define-class helping-points-class
3   :inherit-from (object)
4   :properties(
5     points-list (get-view-points-list (the interface forms link-editor))
6     weight-points (get-view-weight (the interface forms link-editor))
7     weight-points-coord (loop for w in ^weight-points
8                           collect (butlast w)
9                           )
10    height (* (points-distance (nth 0 ^points-list) (nth 1 ^points-list)) 0.03)
11    width ^height
12    depth (* ^height 0.1)
13  )
14  :subobjects(
15    (ui-end-points :class series-object
16      quantity (length ^^points-list)
17      class-expression 'box-object
18      series-prefix 'p
19      init-form '(
20        id ^index
21        line-width 4
22        color 'green
23        height ^^height
24        width ^^width
25        depth ^^depth
26        orientation (list (translate (nth ^index ^^points-list)))
27      )
28    )
29    (ui-weight-points :class series-object
30      quantity (length ^^weight-points)
31      class-expression 'box-object
32      series-prefix 'w
33      init-form '(
34        id ^index
35        line-width 4
36        color 'red
37        height (* ^^height 0.1)
38        width (* ^^width 0.1)
39        depth (* ^^depth 0.1)
40        orientation (list (translate (nth ^index ^^weight-points-coord)))
41      )
42    )
43    (ui-end-points-label :class series-object
44      quantity (length ^^points-list)
45      class-expression 'text-object
46      series-prefix 'plabel
47      init-form '(
48        coordinates (list (nth 0 (nth ^index ^^points-list))
49                          (+ (nth 1 (nth ^index ^^points-list)) ^^height)
50                          (nth 2 (nth ^index ^^points-list)))
51      )
52      text-string (if (oddp ^index) "End Point" "Start Point")
53    )
54    (ui-weight-points-label :class series-object
55      quantity (length ^^weight-points-coord)
56      class-expression 'text-object
57      series-prefix 'wlabel
58      init-form '(
59        coordinates (list (nth 0 (nth ^index ^^weight-points-coord))
60                          (+ (nth 1 (nth ^index ^^weight-points-coord)) ^^height)
61                          (nth 2 (nth ^index ^^weight-points-coord)))
62      )
63      text-string (write-to-string ^index)
64    )
65  )
66  )
67 )
68 )
69 (define-method translate-weight-points helping-points-class ()
70   (let (
71     (point-number (mouse-select-point-from-display !weight-points-coord))
72     (point-instance (nth point-number (series-members (the ui-weight-points))))
73   )
74     (interactive-translate point-instance)
75     (change-model-weight-coords
76       selected-option)))
77     (the connection (:from (the interface forms link-editor member-selector
78                             point-number
79                             (nth 0 (the position (:from point-instance))))
80   )
81 )
82 (define-class link-editor-form-class
```

```

83 :inherit-from(ui-form-class)
84 :properties(
85   ;background-color 'snow2
86   label "NURBS Module"
87   height 840 ;pixels (* 0.7 (nth 1 (get-screen-size)))
88   width 1344 ;pixels (* 0.7 (nth 0 (get-screen-size)))
89   y-offset (* 0.2 (nth 1 (get-screen-size)))
90   x-offset (* 0.2 (nth 0 (get-screen-size)))
91   button-height 3
92   button-width 8
93   button-first-row-x-offset 3
94   button-second-row-x-offset 11
95   cross-section-offset 67
96   mesh-offset 80
97   surface-offset 93
98   weight-points (get-weight-points (the connection (:from (the member-selector selected-option))))
99   weight-points-quantity (length ^weight-points)
100  weight-points-coord (loop for w in ^weight-points
101    collect (butlast w)
102    )
103  points-list (get-points-list (the connection (:from (the member-selector selected-option))))
104  close-action (close-nurbs-module)
105 )
106 :subobjects(
107 (canvas :class 'ui-canvas-class
108   measurement 'percentage
109   x-offset 32 y-offset 5 width 68 height 90
110 )
111 (graphic-toolbar :class 'ui-graphic-control-toolbar-class
112   canvas-object ^canvas
113   measurement 'percentage
114   height 5
115   width 70
116   x-offset 30
117   y-offset 95
118 )
119 (link-selector-label :class 'ui-label-class
120   label "Selected Link"
121   label-align :left
122   height ^button-height width ^button-width x-offset (+ ^button-first-row-x-offset 1) y-offset 10
123 )
124 (link-selector :class 'ui-option-menu-class
125   Button1-action '(progn
126     (update (the superior link-editor-form-class))
127     (draw-member-curves)
128     (zoom :all)
129   )
130   options-list (series-members (the main-mechanism-class links))
131   labels-list (loop for member in (series-members (the main-mechanism-class links))
132     collect (the label (:from member))
133   )
134   selected-option (nth 0 ^options-list)
135   height ^button-height width ^button-width x-offset ^button-second-row-x-offset y-offset 10
136 )
137 (member-selector-label :class 'ui-label-class
138   label "Selected Member"
139   label-align :left
140   height ^button-height width ^button-width x-offset (+ ^button-first-row-x-offset 1) y-offset 13
141 )
142 (member-selector :class ui-option-menu-class
143   Button1-action '(progn
144     (update (the superior link-editor-form-class))
145     (draw-member-curves)
146     ;(blink (the connection (:from ^selected-option)) 2 100)
147   )
148   options-list (series-members (the link-geometry sweeps (:from (the superior link-selector
selected-option))))
149   labels-list (loop for i from 0 to (- (length ^options-list) 1)
150     collect (format nil "Member ~a" i))
151   selected-option (nth 0 ^options-list)
152   height ^button-height width ^button-width x-offset ^button-second-row-x-offset y-offset 13
153 )
154 (Sweep-action-button :class 'ui-action-button-class
155   measurement 'percentage
156   height ^button-height width ^button-width x-offset ^button-first-row-x-offset y-offset 17
157   label "Sweep Link"
158   Button1-action '(progn
159     (draw-link-wo-mesh (the link-geometry (:from (the interface forms link-editor
link-selector selected-option))))
160     (draw (the superior main-mechanism-class helping-points))
161   )
162 )
163 (undraw-all-button :class 'ui-action-button-class
164   measurement 'percentage
165   height ^button-height width ^button-width x-offset ^button-second-row-x-offset y-offset 17
166   label "Undraw"
167   Button1-action (undraw (the main-mechanism-class))
168 )
169 (draw-member-nurb-button :class 'ui-action-button-class
170   measurement 'percentage

```

```

171         height ^button-height width ^button-width x-offset ^button-first-row-x-offset y-offset 20
172         label "Draw Curves"
173         Button1-action '(draw-member-curves)
174     )
175     (sweep-member-nurb-button :class 'ui-action-button-class
176         measurement 'percentage
177         height ^button-height width ^button-width x-offset ^button-second-row-x-offset y-offset 20
178         label "Sweep Member"
179         Button1-action '(draw (the member-selector selected-option) :draw-subobjects? nil)
180     )
181     (point-sheet-label :class 'ui-label-class
182         label "Start/End point"
183         label-align :left
184         y-offset 27
185         x-offset ^button-first-row-x-offset
186         height 3
187     )
188     (point-sheet :class 'ui-spreadsheet-class
189         measurement 'percentage
190         x-offset ^button-first-row-x-offset
191         y-offset 30
192         width 19
193         height 10
194         column-labels (list "x-coord" "y-coord" "z-coord")
195         row-labels (list "Start point" "End point")
196         cell-values ^^points-list
197         number-of-columns (length ^column-labels)
198         number-of-rows 2
199         row-height 28
200         column-width 61
201         attachment-info-list '(top bottom left right)
202         editable? nil
203     )
204     (weight-sheet-label :class 'ui-label-class
205         label "Weight points"
206         label-align :left
207         y-offset 40
208         x-offset ^button-first-row-x-offset
209         height 3
210     )
211     (weight-sheet :class 'ui-spreadsheet-class
212         measurement 'percentage
213         x-offset ^button-first-row-x-offset
214         y-offset 43
215         width 19
216         height 16
217         column-labels (list "x-coord" "y-coord" "z-coord" "weight")
218         row-labels (loop for row from 0 to ^weight-points-quantity
219                     collect (format nil "Point %a" row)
220                 )
221         cell-values ^^weight-points
222         number-of-columns (length ^column-labels)
223         number-of-rows ^weight-points-quantity
224         row-height 27
225         column-width 50
226         attachment-info-list '(top bottom left right)
227         editable? t
228     )
229     (add-row-action-button :class 'ui-action-button-class
230         measurement 'percentage
231         height ^button-height width ^button-width x-offset ^button-first-row-x-offset y-offset 60
232         label "Add Weight Point"
233         Button1-action '(weight-sheet-add-row (the superior link-editor-form-class))
234     )
235     (remove-row-action-button :class 'ui-action-button-class
236         measurement 'percentage
237         height ^button-height width ^button-width x-offset ^button-second-row-x-offset y-offset 60
238         label "Remove Weight Point"
239         Button1-action '(weight-sheet-remove-row (the superior link-editor-form-class))
240     )
241     (move-point-action-button :class 'ui-action-button-class
242         measurement 'percentage
243         height ^button-height width ^button-width x-offset ^button-first-row-x-offset y-offset 63
244         label "Move Weight Point"
245         Button1-action '(progn (translate-weight-points (the superior main-mechanism-class
246                             helping-points))
247                             (update (the superior weight-sheet))
248                             (regen)
249                             (draw (the superior main-mechanism-class helping-points)))
250     )
251     (weight-apply-action-button :class 'ui-action-button-class
252         measurement 'percentage
253         height ^button-height width ^button-width x-offset ^button-second-row-x-offset y-offset 63
254         label "Apply Weight"
255         Button1-action '(weight-sheet-apply (the superior link-editor-form-class))
256     )
257     ;;;;Cross-section;;;
258     (cross-section-selector-label :class 'ui-label-class
259         label "Member Cross-section"
260         label-align :left

```



```

260         height ^button-height width ^button-width x-offset ^button-first-row-x-offset y-offset ^
cross-section-offset
261     )
262     (cross-section-start-label :class 'ui-label-class
263         label "Start"
264         label-align :left
265         height ^button-height width ^button-width x-offset (+ ^button-first-row-x-offset 1) y-offset
(+ ^cross-section-offset 3)
266     )
267     (cross-section-end-label :class 'ui-label-class
268         label "End"
269         label-align :left
270         height ^button-height width ^button-width x-offset (+ ^button-first-row-x-offset 1) y-offset
(+ ^cross-section-offset 6)
271     )
272     (start-cross-section-selector :class ui-option-menu-class
273         height ^button-height width ^button-width x-offset ^button-second-row-x-offset y-offset (+ ^
cross-section-offset 3)
274         Button1-action '(progn
275             (update (the superior link-editor-form-class))
276             (draw-member-curves)
277         )
278         options-list (reverse (class-direct-defined-subclasses 'cross-section-model))
279         labels-list (loop for option in !options-list
280             collect (remove "-section" (write-to-string option))
281         )
282         selected-option (nth
283             (position
284                 (write-to-string (the type (:from (get-cross-section_D (the superior link-editor
member-selector selected-option))))
285                 !labels-list
286             )
287             !options-list)
288         )
289         (end-cross-section-selector :class ui-option-menu-class
290             height ^button-height width ^button-width x-offset ^button-second-row-x-offset y-offset (+ ^
cross-section-offset 6)
291             Button1-action '(progn
292                 (update (the superior link-editor-form-class))
293                 (draw-member-curves)
294             )
295             options-list (reverse (class-direct-defined-subclasses 'cross-section-model))
296             labels-list (loop for option in !options-list
297                 collect (remove "-section" (write-to-string option))
298             )
299             selected-option (nth
300                 (position
301                     (write-to-string (the type (:from (get-cross-section_A (the superior
link-editor member-selector selected-option))))
302                     !labels-list
303                 )
304                 !options-list)
305             )
306             (cs-width-label :class 'ui-label-class
307                 label "w"
308                 height ^button-height width 2 x-offset (+ ^button-second-row-x-offset ^button-width) y-offset ^
cross-section-offset
309             )
310             (cs-height-label :class 'ui-label-class
311                 label "h"
312                 height ^button-height width 2 x-offset (+ ^button-second-row-x-offset ^button-width ^width)
y-offset ^cross-section-offset
313             )
314             (start-width :Class 'ui-typein-field-class
315                 height ^button-height width 2 x-offset (+ ^button-second-row-x-offset ^button-width) y-offset (+ ^
^cross-section-offset 3)
316                 editable? t
317                 model-property-object (the width (:from (the superior link-editor member-selector selected-option))
)
318                 content ^model-property-object
319                 Focusout-validation? t
320             )
321             (start-height :Class 'ui-typein-field-class
322                 height ^button-height width 2 x-offset (+ ^button-second-row-x-offset ^button-width ^width)
y-offset (+ ^cross-section-offset 3)
323                 model-property-object (the height (:from (the superior link-editor member-selector selected-option))
))
324                 content ^model-property-object
325                 Focusout-validation? t
326             )
327             (end-width :Class 'ui-typein-field-class
328                 height ^button-height width 2 x-offset (+ ^button-second-row-x-offset ^button-width) y-offset (+ ^
cross-section-offset 6)
329                 model-property-object (the width-end (:from (the superior link-editor member-selector
selected-option)))
330                 content ^model-property-object
331                 Focusout-validation? t
332             )
333             (end-height :Class 'ui-typein-field-class

```

```

334         height `button-height width 2 x-offset (+ `button-second-row-x-offset `button-width `width)
y-offset (+ `cross-section-offset 6)
335 model-property-object (the height-end (:from (the superior link-editor member-selector
selected-option)))
336 content `model-property-object
337 Focusout-validation? t
338 )
339 (apply-cross-section-button :Class 'ui-action-button-class
340 height `button-height width `button-width x-offset `button-second-row-x-offset y-offset (+ `
cross-section-offset 10)
341 label "Apply Cross-section"
342 Button1-action '(change-cross-section (the superior link-editor))
343 )
344 ;;;;;;;;;Cross-section end;;;;;;;;;;;;;
345 ;;;;;;;;;Mesh-size;;;;;;;;;;;;;
346 (mesh-label :class 'ui-label-class
347 label "Member Mesh size"
348 label-align :left
349 height `button-height width `button-width x-offset `button-first-row-x-offset y-offset `mesh-offset
350 )
351 (mesh-max-size-label :class 'ui-label-class
352 label "Max"
353 label-align :left
354 height `button-height width `button-width x-offset (+ `button-first-row-x-offset 1) y-offset (+ `mesh-offset
355 3)
356 )
357 (mesh-max-field :Class 'ui-typein-field-class
358 height `button-height width 5 x-offset `button-second-row-x-offset y-offset (+ `mesh-offset 3)
359 editable? t
360 model-property-object (the link-geometry max-element-size (:from (the superior link-editor link-selector
selected-option)))
361 content `model-property-object
362 Focusout-validation? t
363 )
364 (mesh-min-size-label :class 'ui-label-class
365 label "Min"
366 label-align :left
367 height `button-height width `button-width x-offset (+ `button-first-row-x-offset 1) y-offset (+ `mesh-offset
368 6)
369 )
370 (mesh-min-field :Class 'ui-typein-field-class
371 height `button-height width 5 x-offset `button-second-row-x-offset y-offset (+ `mesh-offset 6)
372 editable? t
373 model-property-object (the link-geometry min-element-size (:from (the superior link-editor link-selector
selected-option)))
374 content `model-property-object
375 Focusout-validation? t
376 )
377 (apply-mesh-button :Class 'ui-action-button-class
378 height `button-height width `button-width x-offset `button-second-row-x-offset y-offset (+ `mesh-offset 10)
379 label "Mesh"
380 Button1-action '(progn
381 (draw (first (get-link-surface-mesh-elements-query-objects-list
382 (get-mesh-model-object (the superior link-editor link-selector selected-option))))
383 (change-max-mesh-size
384 (the link-geometry (:from (the superior link-editor link-selector selected-option)))
385 (get-value (the superior link-editor mesh-max-field)))
386 (change-min-mesh-size
387 (the link-geometry (:from (the superior link-editor link-selector selected-option)))
388 (get-value (the superior link-editor mesh-min-field)))
389 )
390 ;;;;;;;;;Mesh-size end;;;;;;;;;;;;;
391 (misc-label :class 'ui-label-class
392 label "Misc."
393 label-align :left
394 height `button-height width `button-width x-offset `button-first-row-x-offset y-offset `surface-offset
395 )
396 (surface-label :class 'ui-label-class
397 label "Link surface"
398 label-align :left
399 height `button-height width `button-width x-offset (+ `button-first-row-x-offset 1) y-offset (+ `
surface-offset 3)
400 )
401 (surface-field :class ui-radio-buttons-class
402 height `button-height width 7 x-offset (+ `button-first-row-x-offset 6) y-offset (+ `surface-offset 3)
403 Labels-list '("Yes" "No")
404 Status (let(
405 (surface (series-members (the link-geometry surfaces (:from (the superior link-editor
link-selector selected-option))))
406 )
407 (if surface
408 (if (the display? (:from (nth 0 surface))) 0 1)
409 1)
410 )
411 Button1-action '(progn
412 (let(
413 (surface (series-members (the link-geometry surfaces (:from (the superior
link-editor link-selector selected-option))))
414 )

```

```

414         (if surface
415             (loop for member in surface
416                 do (change-value (the display? (:from member)) (if (= ^status 0) t nil)))
417             (change-value ^status 1))
418         )
419         (update (the link-editor))
420         (regen)
421     )
422 )
423 (blend-label :class 'ui-label-class
424   label "Blend edges"
425   label-align :left
426   height ^button-height width ^button-width x-offset (+ ^button-second-row-x-offset 5) y-offset (+ ^
surface-offset 3)
427 )
428 (blend-field :class ui-radio-buttons-class
429   height ^button-height width 7 x-offset (+ ^button-second-row-x-offset 10) y-offset (+ ^surface-offset 3)
430   Labels-list '("Yes" "No")
431   Status (if (the link-geometry blend? (:from (the superior link-editor link-selector selected-option))) 0 1)
432   Button1-action '(progn
433     (change-value (the link-geometry blend? (:from (the superior link-editor link-selector
434 selected-option))) (if (= ^status 0) t nil))
435     (update (the link-editor))
436     (regen)
437   )
438 )
439 )
440 (define-method weight-sheet-apply link-editor-form-class ()
441   (update-weight-points
442     (the index (:from (the interface forms link-editor link-selector selected-option)))
443     (the index (:from (the interface forms link-editor member-selector selected-option)))
444     (loop for row from 0 to (- (the interface forms link-editor weight-sheet number-of-rows) 1)
445         collect (loop for cell from 0 to 3
446             collect (read-from-string (get-cell-value (the interface forms link-editor
447 weight-sheet) row cell))
448         )
449     )
450     (update (the weight-sheet))
451     (regen)
452     (draw-member-curves)
453   )
454 )
455 (define-method weight-sheet-add-row link-editor-form-class ()
456   (update-weight-points
457     (the index (:from (the interface forms link-editor link-selector selected-option)))
458     (the index (:from (the interface forms link-editor member-selector selected-option)))
459     (append
460       (loop for row from 0 to (- (the interface forms link-editor weight-sheet number-of-rows) 1)
461         collect (loop for cell from 0 to 3
462             collect (read-from-string (get-cell-value (the interface forms link-editor weight-sheet)
463 row cell))
464         )
465       )
466       (list (append (get-new-wpoint-coords (the interface forms link-editor)) (list 1)))
467     )
468     (update (the weight-sheet))
469     (regen)
470     (draw-member-curves)
471   )
472 )
473 (define-method weight-sheet-remove-row link-editor-form-class ()
474   (update-weight-points
475     (the index (:from (the interface forms link-editor link-selector selected-option)))
476     (the index (:from (the interface forms link-editor member-selector selected-option)))
477     (butlast (loop for row from 0 to (- (the interface forms link-editor weight-sheet number-of-rows) 1)
478         collect (loop for cell from 0 to 3
479             collect (read-from-string (get-cell-value (the interface forms link-editor
480 weight-sheet) row cell))
481         )
482     )
483     (update (the weight-sheet))
484     (regen)
485     (draw-member-curves)
486   )
487 )
488 (define-method get-view-weight link-editor-form-class ()
489   !weight-points
490 )
491 (define-method get-view-weight-coords link-editor-form-class ()
492   !weight-points-coord
493 )
494 (define-method get-view-points-list link-editor-form-class ()
495   !points-list
496 )
497 (define-method get-new-wpoint-coords link-editor-form-class ()
498   (if (get-view-weight-coords (the))
499       (mid-point (nth 1 (get-view-points-list (the))) (nth (- (LENGTH (get-view-weight-coords (the))) 1) (
500 get-view-weight-coords (the))))
501       (mid-point (nth 0 (get-view-points-list (the))) (nth 1 (get-view-points-list (the))))))

```

```

498 )
499 )
500 (define-method get-selected-link link-editor-form-class ()
501   (the link-selector selected-option)
502 )
503 (define-method get-selected-member link-editor-form-class ()
504   (the member-selector selected-option)
505 )
506 (define-method change-cross-section link-editor-form-class ()
507   (change-cross-section-start-width
508     (the superior link-editor member-selector selected-option)
509     (get-value (the superior link-editor start-width)))
510   (change-cross-section-start-height
511     (the superior link-editor member-selector selected-option)
512     (get-value (the superior link-editor start-height)))
513   (change-cross-section_D
514     (the superior link-editor member-selector selected-option)
515     (the superior link-editor start-cross-section-selector selected-option))
516   (change-cross-section-end-width
517     (the superior link-editor member-selector selected-option)
518     (get-value (the superior link-editor end-width)))
519   (change-cross-section-end-height
520     (the superior link-editor member-selector selected-option)
521     (get-value (the superior link-editor end-height)))
522   (change-cross-section_D
523     (the superior link-editor member-selector selected-option)
524     (the superior link-editor start-cross-section-selector selected-option))
525   (regen)
526 )
527 (defun update-weight-points (link member weight-sheet)
528   (let
529     (
530       (member-shape (get-member-shape (the main-mechanism-class shapes) link member))
531       (point-number (nth 5 member-shape))
532       (new-point-numbers (if (> (length weight-sheet) (length point-number))
533         (if (get-weight-list (the main-mechanism-class weights))
534           (append point-number
535             (list (+ 1 (nth 0 (sort (loop for point in (get-weight-list (the
536               main-mechanism-class weights)) append (list (nth 0 point))) '>))))
537           (list 0))
538         point-number))
539     (new-w-list
540       (loop for w-s-line in weight-sheet
541         for w-point in new-point-numbers
542         collect (append (list w-point (read-from-string (format nil "l-d,m-d" link member)))
543           w-s-line)
544         )
545     (new-w-ids
546       (loop for new-point-numbers in new-w-list
547         append (list (nth 0 new-point-numbers)))
548       )
549     )
550     (change-weight-points-list (the main-mechanism-class weights) new-w-list)
551     (change-shape-weight (the main-mechanism-class shapes) link member new-w-ids)
552     (smash-value (the connection (:from (the interface forms link-editor member-selector selected-option))))
553     (regen)
554     )
555 )
556 (defun draw-member-curves ()
557   (undraw (the main-mechanism-class))
558   (draw (the superior main-mechanism-class helping-points))
559   (loop for member in (series-members (the link-geometry sweeps (:from (the interface forms link-editor
link-selector selected-option))))
560     do (progn (draw (the connection (:from member)) :draw-subobjects? nil)
561       (change-color (the connection (:from member)) 'white)
562       )
563     )
564   (change-color (the connection (:from (the interface forms link-editor member-selector selected-option))) '
red)
565 )
566 (defun display-nurbs-module ()
567   (add-object (the) 'helping-points 'helping-points-class)
568   (add-object (the interface forms) 'link-editor 'link-editor-form-class)
569   (display (the interface forms link-editor))
570   (activate-display (the interface forms link-editor canvas))
571   (add-light :name 'light1 :color 'white :x 0.7 :y 0.5 :z -0.3)
572   (draw-member-curves)
573   (change-color (the connection (:from (the interface forms link-editor member-selector selected-option))) 'red)
574   (zoom :all)
575   ;(set-current-display-background-color 'grey49)
576 )
577 (defun close-nurbs-module ()
578   (hide (the interface forms link-editor))
579   (delete-object (the main-mechanism-class helping-points))
580   (delete-object (the interface forms link-editor))
581   (delete-current-display-lights)
582   (activate-display (the model-manager interface sketcher-layout-form-class sketcher-main-form sketcher-main-form
work-area-canvas-form canvas canvas canvas canvas))

```

583
584

(regen)
)

NTNU	Hazardous activity identification process		Prepared by	Number	Date
HSE			HSE section	HMSRV2601E	09.01.2013
			Approved by		Replaces
			The Rector		01.12.2006
					

Unit: *M7P* Date: 18.01.2017

Line manager: Torgeir Velo

Participants in the identification process (including their function): Thor Christian Coward

Short description of the main activity/main process: Master project for student Thor Christian Coward. Software development of a KBE system.

Is the project work purely theoretical? (YES/NO): Yes Answer "YES" implies that supervisor is assured that no activities requiring risk assessment are involved in the work. If YES, briefly describe the activities below. The risk assessment form need not be filled out.

Signatures: Responsible supervisor: *Byrn Huggum* Student: *T.C Coward*

ID nr.	Activity/process	Responsible person	Existing documentation	Existing safety measures	Laws, regulations etc.	Comment
1	Software development					

A.4 Risk analysis