



Norwegian University of  
Science and Technology

# Automating tasks in the design loop for mechanism design

**Anders K Kristiansen**  
**Eivind Kristoffersen**

Master of Science in Engineering and ICT

Submission date: June 2016

Supervisor: Bjørn Haugen, IPM

Co-supervisor: Ole Ivar Sivertsen, IPM

Norwegian University of Science and Technology  
Department of Engineering Design and Materials



**MASTER THESIS SPRING 2016  
FOR  
STUD.TECHN. EIVIND KRISTOFFERSEN AND ANDERS  
KRISTIANSEN**

**Automating tasks in the design loop for mechanism design**

***Automatisering av oppgaver ved mekanismedesign***

Knowledge Based Engineering (KBE) is a technology that is typically used for introducing automation in engineering work. This technology has been used with great success in Aker Solutions over the last 10-15 years based on a technology developed by the company TechnoSoft Inc. in Ohio, USA. The KBE applications are based on an object oriented programming languages called AML supported by the TechnoSoft Company. This master assignment is based on the candidates project assignment conducted the autumn 2015. The aim of this assignment is to develop the AML code from the project assignment further and implement more automation and optimization in mechanism design including dynamic simulation.

The master assignment includes the following:

1. Study design optimization techniques and the simulation program FEDEM
2. Establish a development infrastructure for implementing the KBE application, for instance version control of source code using GitHub
3. Study, improve, develop and test the following features:
  - a. Automated mesh generation including volume meshes (requires feedback from TechnoSoft Inc.)
  - b. Include modeling functionality for springs, dampers, forces and torques
  - c. Input to simulation from the AML code
  - d. FE RBE2 and RBE3 elements for link-joint connections (requires feedback from TechnoSoft Inc.)
  - e. Design optimization techniques
  - f. User friendly GUI for the design optimization loop
  - g. Parameterize the geometry generation
  - h. Implement more joint types including higher pairs
  - i. Criteria for automated sizing of joint geometries
  - j. Suggest how modeling of control system simulation could be included
  - k. Suggest tool for extended detailing of individual mechanism links to prepare for production

4. Develop a KBE pilot application for the mechanism design loop based on sub points 3a, 3b and 3c above and demonstrate with examples
5. As far as time allows, extend the pilot application with all the extended features in point 3 and demonstrate with examples

#### Formal requirements:


Three weeks after start of the thesis work, an A3 sheet illustrating the work is to be handed in. A template for this presentation is available on the IPM's web site under the menu "Masteroppgave" (<https://www.ntnu.edu/web/ipm/master-thesis>). This sheet should be updated one week before the master's thesis is submitted.


Risk assessment of experimental activities shall always be performed. Experimental work defined in the problem description shall be planned and risk assessed up-front and within 3 weeks after receiving the problem text. Any specific experimental activities which are not properly covered by the general risk assessment shall be particularly assessed before performing the experimental work. Risk assessments should be signed by the supervisor and copies shall be included in the appendix of the thesis.

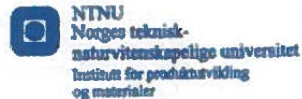
The thesis should include the signed problem text, and be written as a research report with summary both in English and Norwegian, conclusion, literature references, table of contents, etc. During preparation of the text, the candidate should make efforts to create a well arranged and well written report. To ease the evaluation of the thesis, it is important to cross-reference text, tables and figures. For evaluation of the work a thorough discussion of results is appreciated.

The thesis shall be submitted electronically via DAIM, NTNU's system for Digital Archiving and Submission of Master's theses.

The contact person is:  
Ole Ivar Sivertsen, IPM

  
Torgeir Welø  
Head of Division

  
Bjørn Haugen  
Associate professor/Supervisor





## **Preface**

This Master's Thesis, written within the field of Knowledge-Based Engineering, is the culmination of a Master of Science degree in Engineering and ICT. The work was carried out during the spring of 2016 for the Department of Engineering Design and Materials, Norwegian University of Science and Technology (NTNU) in Trondheim.

The thesis is a continuation of our project dissertation, conducted during the autumn of 2015. The project dissertation included a literary study of a pilot application created by Rasmus Korvald Skaare.

We would like to thank Ole Ivar Sivertsen and Bjørn Haugen for their continuous support and guidance throughout our work with this thesis. Our sincere thanks also goes to Technosoft Inc. and Jorgen Dahl for providing the AML software and technical support.

Trondheim, 10-06-2016

Eivind Kristoffersen

Anders Kristiansen



## **Abstract**

This thesis studies how Knowledge-Based Engineering (KBE) concepts can be used to (i) automate tasks in the mechanism design process, and (ii) support and enable the use of design optimization.

The pilot implementation of a KBE system has been reviewed and further developed in the Adaptive Modeling Language, and is hereby referred to as *the mechanism system*. Domain knowledge in the areas of mechanisms, Finite Element Analysis and design optimization, is captured, modeled and used for automation in the mechanism design process. Through simple user input a parametrized model including links, joints, springs, dampers, loads, and subsequent multi-point constraints, is created. The model is used to establish an interface with AML and the analysis software FEDEM. The interface provides the basis for including structural analyses in a fully automated design optimization.

The mechanism system enables a wide range of mechanisms to be automatically modeled, pre-processed and analysed, thereby demonstrating how routine design work in the mechanism design process, can be reduced.



## Sammendrag

Denne avhandlingen studerer hvordan Knowledge-Based Engineering kan bli brukt for å *(i)* automatisere oppgaver i en designprosess av mekanismer, og *(ii)* støtte og tilrettelegge bruk av designoptimalisering.

Pilotimplementasjonen av et mekanismesystem er blitt gjennomgått og videre utviklet i programmeringsspråket Adaptive Modeling Language. Domenekunnskap innenfor mekanismer, elementmetoden og designoptimalisering er formalisert, modellert og brukt til automatisering i designprosessen. Fra enkel inndata blir det laget en parametrisert modell med lenker, ledd, fjærer, dempere, laster, og påfølgende flerpunktsbegrensninger. Modellen brukes til å etablere et grensesnitt mellom AML og analyseprogramvaren FEDEM. Grensesnittet legger grunnlaget for å inkludere strukturelle analyser i en helautomatisert designoptimalisering.

Systemet tillater automatisk modellering, preprosessering og analysering av et stort utvalg av mekanismer. Dette viser hvordan rutinemessig arbeid i designprosessen for mekanismer kan bli redusert.



# Contents

|  |          |
|--|----------|
| Preface . . . . .  | iii      |
| Abstract . . . . .   | v        |
| Sammendrag . . . . .   | vii      |
| List of Figures . . . . .                                    | xiii     |
| List of Named Equations . . . . .                            | xvi      |
| Nomenclature . . . . .                                       | xvii     |
| <b>1 Introduction</b>  | <b>1</b> |
| 1.1 Background . . . . .                                     | 1        |
| 1.2 Research Questions . . . . .                             | 2        |
| 1.3 Structure . . . . .                                      | 2        |
| <b>2 Theory</b>  | <b>5</b> |
| 2.1 The Design Process . . . . .                             | 5        |
| 2.2 Knowledge-Based Engineering . . . . .                    | 7        |
| 2.3 Mechanisms . . . . .                                     | 9        |
| 2.3.1 Links . . . . .  | 9        |
| 2.3.2 Joints . . . . .                                       | 10       |
| 2.3.3 Degrees of Freedom of Planar Mechanisms . . . . .      | 13       |
| 2.3.4 Transformation and Rotations . . . . .                 | 15       |
| 2.3.5 Kinematic Modeling . . . . .                           | 17       |
| 2.4 Finite Element Analysis . . . . .                        | 20       |
| 2.4.1 Meshing . . . . .                                      | 20       |
| 2.4.2 Element Dimensions . . . . .                           | 21       |
| 2.4.3 Boundary Conditions in Structural Mechanisms . . . . . | 25       |

|          |  |           |
|----------|--|-----------|
| 2.5      | Control Systems . . . . .                      | 27        |
| 2.6      | Design Optimization . . . . .                  | 28        |
| 2.6.1    | Design Problem Formulation . . . . .           | 28        |
| 2.6.2    | Unconstrained Methods . . . . .                | 31        |
| 2.6.3    | Constrained Methods . . . . .                  | 32        |
| 2.6.4    | Multi-Objective Optimization Methods . . . . . | 33        |
| 2.6.5    | Structural Optimization . . . . .              | 34        |
| 2.6.6    | Final Notes . . . . .                          | 35        |
| 2.7      | Software Development . . . . .                 | 36        |
| 2.7.1    | Object-Oriented Development . . . . .          | 36        |
| 2.7.2    | Scrum . . . . .                                | 37        |
| <b>3</b> | <b>Methodology</b>                             | <b>39</b> |
| 3.1      | Runtime Environment . . . . .                  | 39        |
| 3.2      | Development Infrastructure . . . . .           | 39        |
| 3.3      | Adaptive Modeling Language . . . . .           | 40        |
| 3.3.1    | Framework . . . . .                            | 40        |
| 3.3.2    | Editor . . . . .                               | 40        |
| 3.3.3    | Source Code Management . . . . .               | 41        |
| 3.3.4    | AML Modeling Forms . . . . .                   | 41        |
| 3.3.5    | AMOpt . . . . .                                | 42        |
| 3.4      | FEDEM . . . . .                                | 43        |
| 3.5      | Modeling of the Mechanism System . . . . .     | 45        |
| <b>4</b> | <b>The Mechanism System</b>                    | <b>47</b> |
| 4.1      | Application Input . . . . .                    | 47        |
| 4.1.1    | Node Positions . . . . .                       | 48        |
| 4.1.2    | Constraints . . . . .                          | 48        |
| 4.1.3    | Link Shapes . . . . .                          | 49        |
| 4.1.4    | Springs and Dampers . . . . .                  | 51        |
| 4.1.5    | Loads . . . . .                                | 51        |
| 4.1.6    | Design Optimization . . . . .                  | 52        |



|          |   |           |
|----------|---|-----------|
| 4.1.7    | Mechanism Library . . . . .               | 52        |
| 4.2      | Initial Frame Placement . . . . .         | 52        |
| 4.3      | Links . . . . .                           | 53        |
| 4.4      | Joints . . . . .                          | 55        |
| 4.5      | Springs and Dampers . . . . .             | 56        |
| 4.6      | Loads . . . . .                           | 57        |
| 4.7      | Mechanism Assembly . . . . .              | 57        |
| 4.8      | Meshing . . . . .                         | 57        |
| 4.8.1    | Boundary Conditions . . . . .             | 59        |
| 4.9      | Analysis . . . . .                        | 59        |
| 4.10     | Results . . . . .                         | 60        |
| 4.11     | Discussion . . . . .                      | 73        |
| <b>5</b> | <b>Design Optimization</b>                | <b>77</b> |
| 5.1      | AMOpt . . . . .                           | 78        |
| 5.2      | The Iteration Process . . . . .           | 78        |
| 5.3      | Problem Formulation . . . . .             | 79        |
| 5.3.1    | The Implementation . . . . .              | 81        |
| 5.4      | Results . . . . .                         | 83        |
| 5.5      | Discussion . . . . .                      | 88        |
| <b>6</b> | <b>Implementation Details</b>             | <b>91</b> |
| 6.1      | General Development Methodology . . . . . | 91        |
| 6.2      | System Architecture . . . . .             | 92        |
| 6.2.1    | Collections . . . . .                     | 92        |
| 6.2.2    | Data Models . . . . .                     | 94        |
| 6.2.3    | Joints . . . . .                          | 94        |
| 6.2.4    | Links . . . . .                           | 95        |
| 6.2.5    | Loads, Springs and Dampers . . . . .      | 96        |
| 6.2.6    | Meshing and Analysis . . . . .            | 97        |
| 6.2.7    | Design Optimization . . . . .             | 98        |
| 6.3      | Results and Discussion . . . . .          | 100       |

|          |                                     |            |
|----------|-------------------------------------|------------|
| <b>7</b> | <b>Final Discussion</b>             | <b>103</b> |
| <b>8</b> | <b>Conclusions</b>                  | <b>107</b> |
| <b>9</b> | <b>Further Work</b>                 | <b>109</b> |
|          | <b>References</b>                   | <b>111</b> |
| <b>A</b> | <b>Installation Details</b>         | <b>A-1</b> |
| <b>B</b> | <b>Class Diagrams</b>               | <b>B-1</b> |
| <b>C</b> | <b>Graphical User Interface</b>     | <b>C-1</b> |
| <b>D</b> | <b>Example Model File</b>           | <b>D-1</b> |
| <b>E</b> | <b>Work Log</b>                     | <b>E-1</b> |
| <b>F</b> | <b>Source Code</b>                  | <b>F-1</b> |
| E1       | System.def . . . . .                | F-2        |
| E2       | Data-models.aml . . . . .           | F-2        |
| E3       | Springs-dampers.aml . . . . .       | F-6        |
| E4       | Loads.aml . . . . .                 | F-13       |
| E5       | Cross-sections.aml . . . . .        | F-17       |
| E6       | Optimizations.aml . . . . .         | F-25       |
| E7       | Constraints.aml . . . . .           | F-38       |
| E8       | Constraint-types.aml . . . . .      | F-45       |
| E9       | Meshing.aml . . . . .               | F-57       |
| E10      | Analysis.aml . . . . .              | F-59       |
| E11      | Link-member-geometry.aml . . . . .  | F-63       |
| E12      | Link-surface-geometry.aml . . . . . | F-75       |
| E13      | Links.aml . . . . .                 | F-78       |
| E14      | Collections.aml . . . . .           | F-86       |
| E15      | Geometry-export.aml . . . . .       | F-102      |
| <b>G</b> | <b>Risk Assessment</b>              | <b>G-1</b> |

# List of Figures

|      |   |    |
|------|---|----|
| 2.1  | The design process . . . . .  | 5  |
| 2.2  | Achievable design time . . . . .  | 8  |
| 2.3  | Binary, ternary and quaternary links . . . . .  | 10 |
| 2.4  | The six lower pairs . . . . .   | 11 |
| 2.5  | Redundant links . . . . .   | 14 |
| 2.6  | Four-bar mechanism . . . . .  | 15 |
| 2.7  | Coordinate transformation . . . . .   | 15 |
| 2.8  | Arbitrary configuration of links $L_i, L_j, L_k$ . . . . .  | 18 |
| 2.9  | Example frame placement and SU parameters . . . . .   | 20 |
| 2.10 | Example models and different mesh representations . . . . .   | 21 |
| 2.11 | Link with hybrid mesh . . . . .   | 22 |
| 2.12 | Element aspect ratios . . . . .   | 22 |
| 2.13 | Example element . . . . .   | 23 |
| 2.14 | Typical high stress areas . . . . .   | 25 |
| 2.15 | RBE2 and RBE3 . . . . .   | 27 |
| 2.16 | Open-loop and closed-loop system . . . . .  | 27 |
| 2.17 | Constraint areas . . . . .  | 31 |
| 2.18 | An object-oriented system consists of many well-encapsulated objects,<br>interacting with one another by sending messages . . . . . | 37 |
| 2.19 | The scrum process . . . . .   | 38 |
| 3.1  | The AML framework . . . . .   | 41 |
| 3.2  | Class diagram extensions . . . . .  | 46 |
| 4.1  | Frame placements for the male and female elements of a revolute joint . . . . .   | 53 |

|      |   |    |
|------|---|----|
| 4.2  | Smooth NURBS curve (illustrated with red dots) generated automatically<br>from joint-directions . . . . . | 54 |
| 4.3  | Ternary and Quaternary links . . . . .  | 55 |
| 4.4  | Kinematic model fixes . . . . .   | 61 |
| 4.5  | Old joint definition . . . . .  | 62 |
| 4.6  | Knuckle and double revolute joints . . . . .  | 62 |
| 4.7  | Joint fillets . . . . .   | 63 |
| 4.8  | Free joint RBE2 Example . . . . .   | 64 |
| 4.9  | Thickened link surface . . . . .  | 64 |
| 4.10 | Spring and Damper Example . . . . .   | 65 |
| 4.11 | Force and Torque Example . . . . .  | 65 |
| 4.12 | Mesh configurations . . . . .   | 66 |
| 4.13 | Refined tetrahedron mesh . . . . .  | 66 |
| 4.14 | Double wishbone suspension represented in the mechanism system and<br>FEDEM . . . . .                     | 67 |
| 4.15 | Four-bar analysis . . . . .   | 67 |
| 4.16 | Object tree in FEDEM . . . . .  | 68 |
| 4.17 | Double wishbone translational deformation . . . . .   | 69 |
| 4.18 | RBE2 nodes . . . . .  | 69 |
| 4.19 | Load and spring/damper connection . . . . .   | 70 |
| 4.20 | 14-bar steering linkage . . . . .   | 71 |
| 4.21 | Log grabber . . . . .   | 72 |
| 4.22 | Slider crank . . . . .  | 72 |
| 5.1  | I-beam cross section . . . . .  | 80 |
| 5.2  | AMOpt loop . . . . .  | 82 |
| 5.3  | Initial vs. Optimal Design . . . . .  | 83 |
| 5.4  | Initial vs. Optimal Design . . . . .  | 86 |
| 5.5  | Area Evaluation Plot . . . . .  | 87 |
| 5.6  | Constraint Function Plot . . . . .  | 87 |
| 6.1  | Class-object diagram of the initial collections . . . . .   | 93 |

|      |   |     |
|------|---|-----|
| 6.2  | Class-object diagram of the initial collections . . . . . | 94  |
| 6.3  | General joint models class-object diagram . . . . .       | 95  |
| 6.4  | Joint types class-object diagram . . . . .                | 95  |
| 6.5  | Link structure class-object diagram . . . . .             | 96  |
| 6.6  | Loads, springs and dampers class-object diagram . . . . . | 97  |
| 6.7  | Meshing and analysis class-object diagram . . . . .       | 98  |
| 6.8  | Optimization models class-object diagram . . . . .        | 99  |
| 6.9  | System model tree . . . . .                               | 101 |
|      |   |     |
| C.1  | Mechanism selection and export GUI . . . . .              | C-1 |
| C.2  | Coordinate system frame GUI . . . . .                     | C-2 |
| C.3  | Joint GUI . . . . .                                       | C-2 |
| C.4  | GUI controlling all links . . . . .                       | C-3 |
| C.5  | Link geometry and member GUI . . . . .                    | C-3 |
| C.6  | Overall links GUI . . . . .                               | C-3 |
| C.7  | Dampers and loads . . . . .                               | C-4 |
| C.8  | Overall links GUI . . . . .                               | C-4 |
| C.9  | Link material properties . . . . .                        | C-5 |
| C.10 | Design optimization GUIs . . . . .                        | C-5 |

# List of Named Equations

|      |  |    |
|------|--|----|
| 2.3  | The Kutzbach Criterion . . . . .                                   | 14 |
| 2.7  | The Direction Cosine . . . . .                                     | 16 |
| 2.8  | Coordinate Transformation . . . . .                                | 16 |
| 2.9  | Frame Definition . . . . .   | 17 |
| 2.10 | The Homogeneous Displacements Matrix . . . . .                     | 18 |
| 2.11 | Spatial Deformation . . . . .                                      | 19 |
| 2.12 | Decomposition of Joint-link Displacement . . . . .                 | 19 |
| 2.13 | Screw Displacement of Joint-link Pair . . . . .                    | 19 |
| 2.14 | Master Stiffness Equation . . . . .                                | 21 |
| 2.15 | Element translation function . . . . .                             | 23 |
| 2.16 | Translational error . . . . .                                      | 23 |
| 2.17 | Translational error ratio . . . . .                                | 23 |
| 2.18 | Element size reduction example . . . . .                           | 24 |
| 2.19 | Translational error . . . . .                                      | 24 |
| 2.20 | Objective function . . . . .                                       | 28 |
| 2.21 | Inequality and equality constraints . . . . .                      | 29 |
| 2.22 | Design space . . . . .   | 29 |
| 2.23 | Constraint normalization . . . . .                                 | 29 |
| 2.24 | Inequality and equality constraints with response vector . . . . . | 30 |
| 5.3  | Case Problem Formulation . . . . .                                 | 80 |
| 5.4  | Final Constraint Function . . . . .                                | 81 |

# Nomenclature

## Symbols

**F** frame, local coordinate system

**F** set of forces

$g(\mathbf{x})$  inequality constraints

$h$  element length

$h(\mathbf{x})$  equality constraints

$\mathcal{J}$  sorted set of joints of mechanism  $\mathcal{M}$

**K** stiffness matrix

$K_t$  stress concentration factor

$\mathcal{L}$  sorted set of links of mechanism  $\mathcal{M}$

$\lambda$  direction cosine

**M** homogeneous matrix of rotation and displacement

$\mathcal{M}$  a mechanism defined as the tuple  $\mathcal{M} = (\mathcal{L}, \mathcal{J})$

$O$  big O

**p** location vector of a frame  $F$

**R** rotation matrix

$\mathbf{r}(\mathbf{x}, t)$  response vector

$\$$  skew

$\sigma(\mathbf{x})$  stress constraints

$\Delta\sigma$  stress error estimate

**T** transformation matrix

**u** set of unit deformations

$\Delta v$  error estimate

**x, y, z** axes-vectors of a frame  $F$

## **Acronyms**

**AI** Artificial Intelligence

**AML** Adaptive Modeling Language

**CAD** Computer Aided Design

**CMS** Component Mode Synthesis

**DOF** Degree of Freedom

**DOT** Design Optimization Tools

**FBS** Frame Based System

**FEDEM** Finite Element Dynamics in Elastic Mechanisms

**FEA** Finite Element Analysis

**GA** Genetic Algorithms

**GUI** Graphical User Interface

**KBE** Knowledge Based Engineering

**LISP** List Processing

**MMA** Method of Moving Asymptotes

**MOOM** Multi-objective Optimization Methods

**MPC** Multi Point Constrain

**NURBS** Non-uniform Rational Basis Spline

**OOA** Object-Oriented Analysis



|             |  |
|-------------|--|
| <b>OOD</b>  | Object-Oriented Development                      |
| <b>OOP</b>  | Object-Oriented Programming                      |
| <b>OMG</b>  | Object Management Group                          |
| <b>PDE</b>  | Partial Differential Equations                   |
| <b>RBE</b>  | Rigid Body Element                               |
| <b>SCF</b>  | Stress Concentration Factor                      |
| <b>SLP</b>  | Sequential Linear Programming                    |
| <b>SPC</b>  | Single Point Constraints                         |
| <b>SU</b>   | Sheth-Uicker                                     |
| <b>SQP</b>  | Sequential Quadratic Programming                 |
| <b>SUMT</b> | Sequential Unconstrained Minimization Techniques |
| <b>UML</b>  | Unified Modeling Language                        |

## Definitions

The majority of the following definitions are retrieved from the sources used in the theory. However, since most of the definitions are rather long, we have chosen not to use quotation marks in this section, even for direct quotes.

### Abstraction

*Abstraction* is, when used in the context of object-oriented programming, a technique used to manage the complexity of a system by hiding all but the relative data of an object. Thus it relates both encapsulation and data hiding [29]

### Class

A *class* is a template that defines an entity 'type' with properties and subobjects. It is meant to be instantiated into an object, and can be instantiated an indefinite number of times. Methods are also defined for a class and considered part of the character of the class even though they are not defined within the class. A class can be thought of as a recipe for creating instances. For example the same cake recipe may be used many times, and each time, a different cake instance is created [47].

### Constraint

In the context of design optimization, a *constraint* defines a requirement on the design and performance which the system must satisfy. It is often denoted as a function of a set of design variables,  $\mathbf{x}$ , where the constraint function is  $g(\mathbf{x})$ .

### Control system

A *control system* is a device or process that regulates the behavior of another device or system with regard to a particular condition [4]. Control systems can be found in a traffic light, water boiler, spacecraft, CNC machine etc.

### Degrees of Freedom

The *degrees of freedom* is defined as the number of independent movements a rigid body, or mechanism, has. A degree of freedom can either be translational or rotational.

### Demand-Driven Calculation

*Demand-driven calculation* means that properties and subobjects are not created until

the first time they are needed/demanded. If a property is not referred to (demanded) it will not calculate its value. The first time a property is demanded the value is calculated and that value is retained by the property. Subsequent demanding of the same property will not cause re-calculation because the first calculation is retained, unless the value of that property has been smashed. A property value is smashed when other properties or objects that it depends on are changed or smashed [47].

### **Dependency**

A property is said to *depend* on another property if its formula (or any method/function called from the formula) references that other property. When a dependency is established, the property value smashes anytime the property it depends on changes value, formula or smashes [47].

### **Dependency Backtracking**

*Dependency backtracking* is the mechanism that causes properties to recalculate their retained values (from their formula) when other properties that were used to calculate the value change. That is if volume is calculated by multiplying height by width by depth and the height, width, or depth change, the volume property value will become unbound. The value will not be calculated until it is needed again because of demand driven calculations [47].

### **Design Variable**

A design variable is a numerical input that refers to a concrete parameter of a system, and is the only variable allowed to change during the design optimization.

### **Design Optimization**

*Design optimization* is the act of finding the best performance of a system under given constraints.

### **Dynamic Simulation**

*Dynamic simulation* refers to the calculations of motion in a mechanical system on the computer where both constraint forces and the forces necessary to drive the system are taken into account [38].

## **Expanding**

*Expanding* an AML object is the action of demanding all its subobjects, and can also include expanding the subobjects too. When an instance of a class is created, if this class contains defined subobjects, the subobject instances are not created until referenced/demanded [47].

## **Finite Element Analysis**

*Finite Element Analysis* (FEA) is a numerical method for approximating solutions to boundary value problems for partial differential equations.

## **Inheritance**

*Inheritance* is a mechanism for class (code) reuse. Through inheritance a class will have the same properties, subobjects, and methods as the class (superclass) that is being inherited from. The tree structure created by class inheritance is referred to as the class hierarchy [47].

## **Instance**

An *instance* is an object that is created using the class definition. All instances of the same class have the same properties although the values of the properties may be different depending on the operations that have been performed on the instances. For example each time a cake is baked makes an instance of the cake recipe which could be thought of as a class. Throughout the manual, the expression 'instance of Class-A' means an instance of Class-A or of any other class that directly or indirectly inherit from Class-A [47].

## **Joint**

A *joint* is a connection between two adjacent links. The primary kinematic function of a joint is to constrain the relative motion allowed between the connected links.

## **Kinematics**

*Kinematics* refers to the calculations of motion in a system with no reference to forces and torques in the system or the input necessary to achieve the motion [38].

**Link**

A *link* is a body with the primary function of fixing geometric relationships between its joint elements.

**Machine**

A *machine* is a device which transforms energy in some available form and utilises it to do some particular type of work [32].

**Mechanism**

A *mechanism* is a device which transforms motion to a desirable pattern. It is an assembly of links where at least two of the links have relative motion. The links may be connected to each other by joints that constrain their relative motion [38].

**Meshing**

*Meshing* is the process of dividing a geometric model into a finite number of elements.

**Method**

A *method* is an operation or function that is defined specifically for a class. A method name may have definitions for many classes that must perform the same operation but in different ways. For example suppose the method volume is defined for each of the classes box, cylinder, and sphere. By calling the volume method with an instance of one of those classes the correct code will be executed automatically and the volume of the instance returned. Inheriting from a class that has methods defined for it will also inherit the methods [47].

**Mobility**

The *mobility* of a mechanism is the number of degrees of freedom it possesses. An equivalent definition of mobility is the minimum number of independent parameters required to specify the location of every link within a mechanism [37].

**Model**

A *model* is defined as a more or less simplified representation of the engineering aspects of a system. The model must represent the properties of the aspects of a system that you would like to study, as accurately as possible [38].

## **Modularity**

The *modularity* of a system is a term used in context of object-oriented programming. It is used to determine how easily a system's modules (or components) may be separated and modified.

## **Object**

An *object* is a term that is used to refer to an instance of a class and all of the methods of that class and its superclasses. It may be considered to be a 'packet' of software characterized by a set of operations (methods), a set of variables (properties) to store the results of operations, and a set of objects (subobjects) that define the structure. In fact an object-oriented system does not have objects, but instead classes and instances. Defining an object is really defining a class and making an object is really creating an instance [47].

## **Objective Function**

In design optimization, an *objective function* is a measure of the performance of a system, based on the design variables.

## **Semantic Web**

*The Semantic Web* is an extension of the current World Wide Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [10].

## **Simulation**

*Simulation* is the imitation of certain properties of a system in a mathematical computational model [38].

## **Spaghetti Code**

*Spaghetti code* refers to poorly constructed, disorganized, and unstructured source code [22].

**Subclass**

*Subclass* is a term that refers to any class that inherits from a given class. Subclasses are thought of as down the class hierarchy. For example the class baseball is a subclass of a class sphere [47].

**Superclass**

*Superclass* is a term that refers to any class that is inherited by a given class. A Superclass of a class C is a class that can be found by tracing up the class hierarchy of C. For example the class sphere is a superclass of the class baseball [47].





# Chapter 1

## Introduction

### 1.1 Background

Most of the early architectural wonders of the world were designed by individuals with only a small team of assistants. The great Papal Basilica of St. Peter in the Vatican was designed primarily by Michelangelo, Bramante, Maderno and Bernini. When Joe Sutter designed the Boeing 747, he started with a few hundred engineers in his team, but ended with around 4,500. With time, the number of individuals involved in the design process has grown, reflecting both the sheer complexity of the products and the design process itself. Complex products and systems today impose intricate cross-connections between the parts and their abstractions in the underlying mathematical models.

Faced with an increasingly competitive and demanding marketplace, companies are forced to decrease production time and costs, while enhancing the quality of their products. Two technologies which can help cope with this situation are Knowledge-Based Engineering (KBE), and design optimization. KBE allows for automation in routine engineering work (such as calculations, geometry generation and documentation), by acquiring, storing and reusing domain knowledge. This frees up time to explore a larger solution space, thus enabling further enhancement and innovation of products. The sheer complexity of products today makes finding an optimal design a challenging task.

Experienced designers may be able to use their skills and intuition in order to obtain optimal designs for simple products. However, this becomes increasingly difficult or even impossible for complex products. One approach is to use optimization methods to aid the designer in finding the optimal design. By incorporating optimization methods in the design process the designer may be relieved of complicated calculations. Also, by identifying an explicit set of parameters in which the product can be changed and measured, the designer may increase his/her insight and knowledge.

The mechanism design domain could greatly benefit from integrating KBE and design optimization. Mechanisms are, in general, very complex and difficult to analyze. In many cases, they consist of a large number of individual components acting together as a single entity. Thus, the governing kinematic equations of mechanisms are highly non-linear, accentuating the need of numerical computations and optimization algorithms. A KBE system with integrated optimization methods for mechanism design, could not only decrease the overall time-to-market of mechanisms, but also aid designers in finding better designs to more complex systems.

## 1.2 Research Questions

The following research questions has been developed in accordance with the supervisor.

**RQ1** How can a KBE system automate tasks in the mechanism design process?

**RQ2** How can a KBE system support and enable the use of design optimization?

## 1.3 Structure

The remaining part of the thesis is organized as follows.

**Chapter 2** describes the underlying theory for the thesis. The main sections here are mechanisms and design optimization, in virtue of the assignment. Different design optimizations techniques are studied and further discussed in Chapter 5. In addition, a

general overview of the design process, KBE, FEA, control systems and software development, is presented.

**Chapter 3** presents the tools and methods used for the work with the thesis. Only a brief explanation of the development infrastructure is given (alluded by supervisor), and further discussed in Chapter 6.

**Chapter 4** describes the functionality and main synthesis of the mechanism system. Related results and findings are also presented and discussed.

**Chapter 5** shows how the mechanism system enables automatic execution of analyses and how this can be integrated as a part of a design optimization loop.

**Chapter 6** presents the general development methodology used and takes an object-oriented view of the system architecture. Class-object diagrams are presented both for visualization and documentation purposes.

**Chapter 7** discuss the results and findings from Chapters 4 to 6 in light of the research questions presented in Section 1.2.

**Chapter 8** draws conclusions from Chapter 7.

**Chapter 9** reviews areas for further work.

All the features in point 3 of the master assignment have been studied, implemented and tested, with the exception of higher pair joints and tools for extended detailing, in accordance with supervisor. In addition, a video demo has been made to demonstrate the functionality of the mechanism system. The video can be found on <https://vimeo.com/170026181>, and the reader is encouraged to view it, as a supplement to this thesis. Also, pictures describing the system's Graphical User Interface (GUI), can be found in Appendix C.



# Chapter 2

## Theory

### 2.1 The Design Process

Design engineers are faced with demands of obtaining optimal designs of complex mechanical and structural systems whilst keeping the time-to-market of the products as small as possible. The design process can be a complex process as it includes all activities necessary to transform a set of design specifications into a final design, which can be manufactured and sold. During this process, a given design is analyzed and modified in order to meet a set of requirements. It is an iterative process as it often requires several rounds of modifications and analysis before the final design is reached.

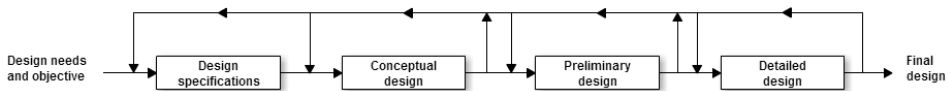


Figure 2.1: The design process

The design process can be divided into four major steps: *design specification*, *conceptual design*, *preliminary design* and *detailed design* [49]. The model shown in Figure 2.1 is based on Arora's [1] system evolution model, with some minor modifications introduced by Trier [49]. As the design might fail to satisfy the design specifications, the designer may have to go back and do modifications across all of the four steps. This is illustrated with feedback loops in the model. Since most designs require a numerous

amount of modifications, it is essential for the designer that these feedback loops take as little time as possible.

The first step of the design process is the *design specification*. This is where the operation tasks, requirements and performance of the system are formulated. From these specifications the designer formulate the design problem at hand. The second step is to create the *conceptual design*. For mechanisms, this may involve defining the configuration, or topology, as well as functional and structural properties. The third step is to take the design model from the *conceptual design* and transform them into a *preliminary design*. The final step is to specify the *detailed design*. For mechanisms, this may involve extended production detailing.

A classical approach for the design process involves using design tools for mechanism modeling and then separating analysis tools for meshing, calculations, pre- and post-processing. If modifications are needed, the designer have to go back to the design tool to modify the design before further analyses can be run. This process of switching between different applications and tools can be very tedious. One of the aims of using KBE tools is to facilitate the connection between the design process and numerical analyses. The connection can be made by integrating the design and analysis tools in a joint platform, or a single model [8]. By keeping a parametrized model throughout the design process, the designer is able to create a real link between the geometry of a mechanism and the simulation. Since all of the different steps in the process refer to the same computer model, the designer avoids the time-consuming tasks of rebuilding the design model between the steps.

## 2.2 Knowledge-Based Engineering

Many definitions can be found of KBE. These typically reflect the different views from various stakeholders. KBE for a company manager can be seen as something entirely different than for a KBE developer. In order to grasp the concept of KBE, one have to look at its fundamentals. KBE is the merging of artificial intelligence (AI), computer aided design (CAD) and object-oriented programming (OOP). KBE merges these disciplines into one common platform. G. La Rocca has the following extended definition [33]:

*“Knowledge-Based Engineering (KBE) is a technology based on the use of dedicated software tools called KBE systems, which are able to capture and systematically reuse product and process engineering knowledge, with the final goal of reducing time and costs of product development by means of the following:*

- *Automation of repetitive and non-creative design tasks*
- *Support of multidisciplinary design optimization in all the phases of The Design Process”*

AI, CAD and OOP are widely represented in scientific literature. This is not the case with KBE. To date, there are only a few scientific books to be found on the topic. This is mainly due to the fact that KBE for many years has been in the exclusive domain of a few competitive industries, such as aerospace and automotive. KBE has not yet entered into the mainstream of academic research, thus limiting the amount of available information.

There is also a limited amount of information regarding KBE success stories. Even though the KBE principles make big promises to both reduce product development time and costs, few good metrics can be found to estimate these advantages [34]. Nevertheless, the potential advantages of KBE goes beyond the reduction of time and costs. Seen from the engineer’s perspective, KBE helps to automate routine tasks such as calculations, geometry generation and documentation, thus making the feedback loops faster, as illustrated in Figure 2.1. This frees up time which can be used to further enhance and innovate the product. When up to 80% of design time is spent on routine

task, as illustrated in Figure 2.2, the potential for reuse is huge. From a company's perspective, KBE proves to be a promising solution as a knowledge portfolio as well. The ability to capture, store and reuse the company's intellectual property can prove crucial in order for the company to remain competitive within a growing international marketplace [9].

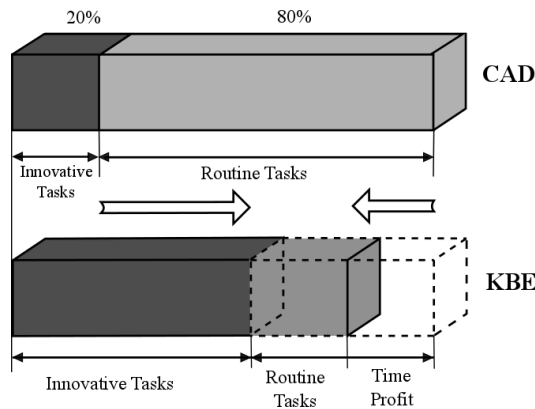


Figure 2.2: Achievable design time

Advantages aside, KBE holds several challenges that need to be met. In Verhagen's journal entry [50], three key challenges are identified: (i) The need for improved methodological support. The lack of exchange standards and excessive ad hoc character of KBE development makes this one of the most important challenges. (ii) Difficulty with transparency and traceability of knowledge. As there is a clear tendency towards development of 'black-box' applications in KBE, more attention needs to be paid to the semantics of knowledge sharing, in order to establish a clearer 'knowledge connection'. With an improved knowledge connection, one can achieve greater interoperability between heterogeneous systems and KBE tools. (iii) The necessity for a quantitative framework to assess the viability and success of KBE development. To date, there is no framework or method available to determine whether a design task, product or process is suitable for KBE development.



## 2.3 Mechanisms

Most of the theory in this section is adopted from the textbooks by Khurmi [23], Singh [37], Thornton and Marion [48], Sivertsen [38] and Finger [17].

A mechanism, or multibody system, is defined as a device which transforms motion to a desirable pattern [38]. Mechanisms usually take part in a larger mechanical system, for instance a steering mechanism of a car. In order to transform a given input motion to a desirable output motion, mechanisms consist of links interconnected by joints which constrain their relative motion. When two links are in contact with each other, they are said to form a pair. Furthermore, if the relative motion between the links are successfully constrained, e.g. in a particular direction, the links form a *kinematic pair*. When two or more kinematic pairs are coupled together in such a way that the last link is interconnected to the first link to transmit definite motion, it is called a *kinematic chain*. When one of the links in a kinematic chain is fixed, the chain is known as a mechanism. If the mechanism is required to transmit power or to do work, it is known as a *machine* [23].

Mechanisms are generally divided into the following three categories: *planar*, *spherical* and *spatial* mechanisms. In planar mechanisms all moving points lie in parallel planes. In spherical mechanisms all moving points moves in such a way that their trajectories lie in concentric spheres. In spatial mechanisms there is no restrictions as to which planes relative motion can occur.

### 2.3.1 Links

Each moving part in a mechanism is known as a *kinematic link* (or simply, link). A link, in itself, may consist of several parts, however these are rigidly fastened together and thus can not move relative to one another. The individual parts that makes up a link are called *members*. A link, also known as a body, do not necessarily need to be rigid in order to transmit motion, but it must be resistant. A resistant link in this context means a link which is capable of transmitting the required force with negligible deformation. This leaves us with rigid, flexible and fluid links. A *rigid link* is a link which does not undergo

any deformation while transmitting motion, e.g. a connection rod. *Flexible links* on the other hand are allowed to partly deform. However, the deformation occurs in a way that does not affect the motion being transmitted, like chains and wires. Lastly, *fluid links* are links which are formed by confining fluid in a receptacle in order to transmit motion through the fluid with pressure and compression, as seen in hydraulic presses.

The number of joints incident on a link (how many links to which it is connected) determines its degree. A link which connects two other links is known as a *binary link*. In Figure 2.3a the links 1, 2, 3 and 4 are all binary links as each link has two joints. Analogously, if a link is connected to three or four links it is known as a *ternary* or *quaternary link* [2]. Figure 2.3c shows a linkage of both binary, ternary and quaternary links where the latter is connected to links 2, 4, 6 and 8.

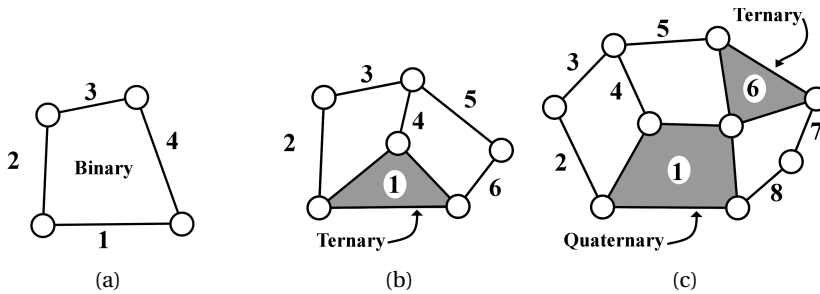


Figure 2.3: Binary, ternary and quaternary links

### 2.3.2 Joints

The kinematic pairs, hereby referred to as *joints*, can be classified according to the type of contact between the links. When two links of a pair have surface contact as relative motion takes place, meaning one of the surfaces slides over the other, the pair formed is known as a *lower pair*. If the two links have a line or point contact when relative motion takes place, the pair is known as a *higher pair*. The lower and higher pairs can be further categorized with respect to their *degrees of freedom* (DOFs). The DOFs of a joint, or object in general, is defined as the number of independent relative motions [37], thus a joint can have six DOFs. As a joint constraints the relative motion between two links it is not a separate physical entity by itself, but the interface composed of the contact

surfaces of the two links [39]. The two contact surfaces, when considered separately, are each referred to as a *joint element*. From the nature of the joint geometries it is apparent to refer to one the elements in a pair as male, or solid, and the other as female, or hollow. In each element there is placed a coordinate system. The relative motion permitted between the elements are assigned parameters referred to as *joint variables*.

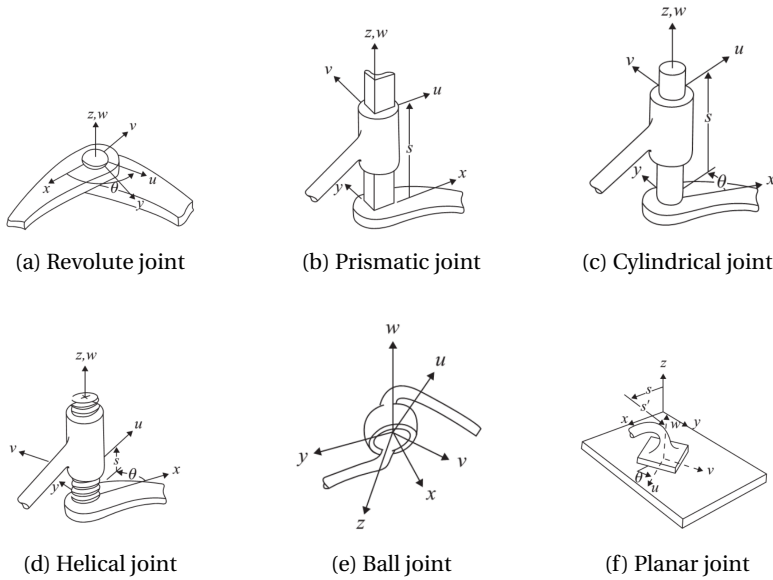


Figure 2.4: The six lower pairs [36]

The mechanisms presented in this thesis makes only use of lower pair joints, thus these will be the main focus of discussion. The six lower pairs, seen in Figure 2.4, are described with their joint axis as being the axis of rotation, translation or spindle, denoted as the z-axis.

A *revolute joint* constrains the relative motion of a pair to a single axis of rotation, allowing only for independent rotational motion around the joint axis. The revolute joint has only one DOF with the joint variable being the relative rotation ( $\Delta\theta$ ) between the links.

A *prismatic joint* constrains the relative motion of a pair to a single axis of translation, allowing only for independent translational motion along the joint axis. The prismatic joint has only one DOF with the joint variable being the relative translation ( $\Delta s$ ) between the links.

A *cylindrical joint* is equivalent to the revolute joint in series with the prismatic joint as it constrains the relative motion to a coaxial joint axis, allowing both independent rotation and translation. The cylindrical joint has two DOFs with both  $\Delta\theta$  and  $\Delta s$  as joint variables.

A *helical joint* is equivalent to the cylindrical joint as it allows both rotation and translation along the joint axis. However, since the links are constrained to screw motion, as a result of the threads, the helical joint has only one DOF and either  $\Delta\theta$  or  $\Delta s$  may be used as the joint variable.

A *ball joint*, also known as a spherical joint, is equivalent to a series of three revolute joints intersecting at a central point. This allows for free rotation around all the axis. The spherical joint has three DOFs and the three angles  $\Delta\theta$ ,  $\Delta\theta'$  and  $\Delta\theta''$  as joint variables.

A *planar joint* constrains two planar surfaces together allowing them to freely translate and rotate whilst remaining in contact with each-other. The planar joint has two translational DOFs and one rotational DOF about the joint axis normal. Hence, the three joint variables can be chosen as  $\Delta s$ ,  $\Delta s'$  and  $\Delta\theta$ .

In addition to these six lower pairs, we have taken advantage of defining a *free joint*. A free joint is initially defined to have six unconstrained DOFs, resulting in full freedom of motion between the links. The purpose of this joint is to facilitate introduction of constraints, prescribed motions and spring/damper properties [14]. All six DOFs are potential joint variables allowing for both no relative motion (resulting in a *rigid joint*) as well as relative translation and/or rotation.

The geometry of joints should be designed in order to minimize the stress concentration present under load. As a general rule, forces should be transmitted throughout the

mechanism as smoothly as possible. The transmission path of the force can be considered as a *force flow*. Thus, sharp transitions in the direction of the flow should be removed by smooth and rounded contours [53]. A typical sharp transition can be seen between the joint and member geometries. By blending these geometries together (with a fillet), one can obtain a smoother path for the force to flow. Additionally, the joints have to be properly dimensioned in order to withstand the required stresses. One way of determining the dimensions of a joint is by looking at the stress concentration factor (SCF). A look through typical joint dimensions calls for the ratio  $D/d$  to be between 1.2 and 1.5, where  $D$  is the member width and  $d$  is the bore diameter [7]. The scale-factor,  $r/d$ , for fillets typically range between 0.02 and 0.06, where  $r$  represents the fillet radius.

### 2.3.3 Degrees of Freedom of Planar Mechanisms

When designing mechanisms one of the most important concerns is the number of DOFs, hereby referred to as the *mobility*, of a mechanism. The mobility can also be defined as the minimum number of independent parameters required to specify the location of every link in a mechanism [37]. In order to determine the mobility of a planar mechanism one can apply the Kutzbach criterion. The *Kutzbach criterion* determines the mobility directly from the number of links and the number and types of joints which it includes given as [37]:

$$D = 3(n - 1) - 2p - h \quad (2.1)$$

where

$$D := \text{DOFs}$$

$n :=$  total number of links in a mechanism where one is fixed

$n - 1 :=$  number of movable links

$p :=$  number of lower pair joints (has one DOF)

$h :=$  number of higher pair joints (has two DOFs)

When two links are connected via a revolute joint, two DOFs are lost. Hence for  $p$  number of joints the DOFs lost are  $2p$ . Therefore, when a mechanism consists of different

types of links the total number of lower pairs are:

$$p = (1/2)[2n_2 + 3n_3 + 4n_4 + \dots] \quad (2.2)$$

where

$n_2$  := number of binary links

$n_3$  := number of ternary links, and so forth

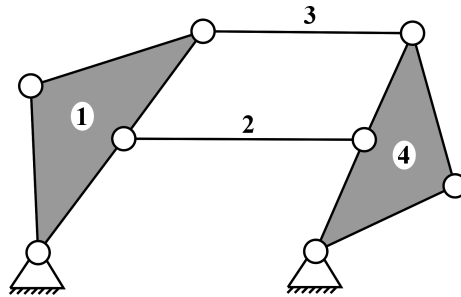


Figure 2.5: Redundant links

Before the correct number of DOFs of a mechanism can be determined, one may have to consider the presence of redundant links. If a link can be moved without introducing new or unexpected motion to the mechanism the link is said to have a redundant DOF ( $D_r$ ) [37]. As seen in Figure 2.5, link 2 and 3 are parallel and since they produce no extra constraint they can be considered redundant. By removing one of them the motion remains the same.

Therefore, Equation 2.1 can be modified to:

$$D = 3(n - n_r - 1) - 2(p - p_r) - h - D_r \quad (2.3)$$

A four-bar mechanism, as shown in Figure 2.6, consists of four binary links, each with two revolute joints. The dotted line illustrates the 'ground link'. As the mechanism has no higher pair joints, the mobility of the system is 1 (1 DOF).

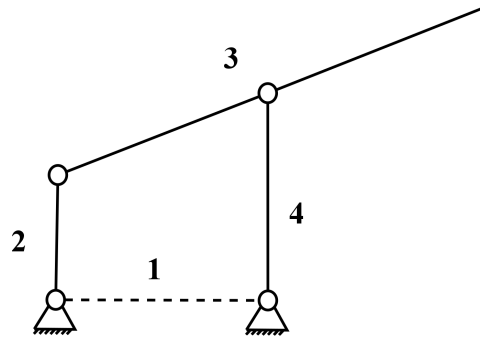
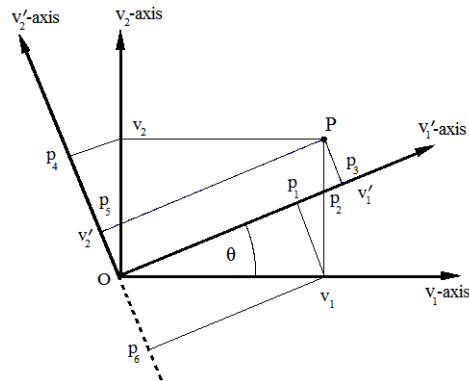


Figure 2.6: Four-bar mechanism

### 2.3.4 Transformation and Rotations

When describing the position of an object in space it is imperative to know what coordinate system the position is relative to. In mechanism modeling there might be many different coordinate systems present, e.g. one at the system, or assembly, level and one for each substructure. The relations between the systems can be described by *coordinate transformations*. The principle of coordinate transformation will be demonstrated by transforming a point  $P$  from one coordinate system to another (two-dimensional):

Figure 2.7: The position of a point  $P$  can be represented in two coordinate systems

Consider the point  $P$  with the coordinates  $(v_1, v_2)$ , as seen in Figure 2.7, in a given coordinate system. Next consider another coordinate system that can be obtained by rotating the original coordinate system by a degree of  $\theta$ . The new coordinates of point  $P$  will then be given as  $(v'_1, v'_2)$ . By geometric inspection we can see that  $v'_1$  becomes

the sum of the projecting of  $v_1$  and  $v_2$  onto the  $v'_1$ -axis (the line  $\overline{Op_1}$  and  $\overline{p_1p_2} + \overline{p_2p_3}$ ). Analogously, the coordinate  $v'_2$  is the projection of  $v_1$  and  $v_2$  onto the  $v'_2$ -axis (the line  $\overline{Op_4} - \overline{p_4p_5}$ ). Furthermore, since  $\overline{p_4p_5}$  is equal to  $\overline{Op_6}$  we get the following:

$$\mathbf{v}' = \begin{bmatrix} v'_1 \\ v'_2 \end{bmatrix} = \begin{bmatrix} v_1 \cos(\theta) + v_2 \sin(\theta) \\ -v_1 \sin(\theta) + v_2 \cos(\theta) \end{bmatrix} = \begin{bmatrix} v_1 \cos(\theta) + v_2 \sin(\frac{\pi}{2} - \theta) \\ v_1 \cos(\frac{\pi}{2} + \theta) + v_2 \cos(\theta) \end{bmatrix} \quad (2.4)$$

By denoting  $(v'_i, v_j)$  as the angle between the  $v'_i$ -axis and the  $v_1$ -axis, and the  $v'_2$ -axis and the  $v_2$ -axis, we can define the set  $\lambda_{ij} \equiv \cos(v'_i, v_j)$  which gives:

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{bmatrix} = \begin{bmatrix} \cos(v'_1, v_1) & \cos(v'_1, v_2) \\ \cos(v'_2, v_1) & \cos(v'_2, v_2) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \quad (2.5)$$

Then the Equation 2.4 becomes:

$$\mathbf{v}' = \begin{bmatrix} v'_1 \\ v'_2 \end{bmatrix} = \begin{bmatrix} v_1 \cos(v'_1, v_1) + v_2 \cos(v'_1, v_2) \\ v_1 \cos(v'_2, v_1) + v_2 \cos(v'_2, v_2) \end{bmatrix} = \begin{bmatrix} \lambda_{11} v_1 + \lambda_{12} v_2 \\ \lambda_{21} v_1 + \lambda_{22} v_2 \end{bmatrix} \quad (2.6)$$

Or, for three dimensions:

$$\mathbf{v}' = \begin{bmatrix} v'_1 \\ v'_2 \\ v'_3 \end{bmatrix} = \begin{bmatrix} \lambda_{11} v_1 + \lambda_{12} v_2 + \lambda_{13} v_3 \\ \lambda_{21} v_1 + \lambda_{22} v_2 + \lambda_{23} v_3 \\ \lambda_{31} v_1 + \lambda_{32} v_2 + \lambda_{33} v_3 \end{bmatrix} \quad (2.7)$$

Thus, in general, the transformation in summation notation is:

$$\boxed{v'_i = \sum_{j=1}^3 \lambda_{ij} v_j, i = 1, 2, 3} \quad (2.8)$$

$\lambda_{ij}$  is known as the *direction cosine* of the  $v'_i$ -axis relative to the  $v_j$ -axis. When expressed in matrix notation,  $\boldsymbol{\lambda}$ , it is called a *transformation* or *rotation matrix*. In further notion the transformation and rotation matrix is denoted as  $\mathbf{T}$  and  $\mathbf{R}$ .



### 2.3.5 Kinematic Modeling

The kinematic modeling convention used by Skaare, in the pilot implementation, is based on the Sheth-Uicker (SU) convention. As kinematic modeling has not been the main focus of our study we present only the main parameters and theory behind the SU convention. In order to illustrate the modeling of a mechanism we have adopted some examples from Bongardt's journal entry [6]. The problem of kinematic modeling can be defined as following:

*Given an arbitrary, physical mechanism, create a specification which gives both the physical copy and software model of the mechanism the same kinematic properties. This must be done without knowing the original mechanism and should be human-readable, compact and reflect both the topology and geometry of the mechanism.*

As of yet, the problem of kinematic modeling has not been solved [6]. A lot of the challenges boils down the specification and displacements of coordinate systems, hereby referred to as frames. Frames are invariable of time and do not move. The SU convention defines an augmented two-frame method which separates the displacements in links and joints, for then to decompose them into three axial screw displacements. This is done by having four frames at each joint, thus for a link-joint pair there is exactly two frames (one 'ordinary' and one 'augmented'). In matrix notation, a frame is defined as:

$$\mathbf{F} = \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.9)$$

The z-axis is the major axis of  $\mathbf{F}$  as it indicates the *dominant* direction. When  $\mathbf{F}$  is attached to a joint, the z-axis coincides with the joint direction. The x-axis is the minor axis of  $\mathbf{F}$  as it indicates the *secondary* direction and coincides with the link direction. The y-axis is the redundant axis of  $\mathbf{F}$ , thus follows from  $y = z \times x$ . Lastly,  $\mathbf{p}$  indicates the location of  $\mathbf{F}$  and can be determined by  $\mathbf{p} = x \cap y \cap z$ .

A mechanism  $\mathcal{M}$  is defined as the tuple  $\mathcal{M} = (\mathcal{L}, \mathcal{J})$ , where  $\mathcal{L}$  and  $\mathcal{J}$  denotes the sets of links and joints:  $\mathcal{L} = [L_1, L_2, \dots, L_n]$  and  $\mathcal{J} = [J_{i_1, j_1}, J_{i_2, j_2}, \dots, J_{i_m, j_m}]$ . For a given 'posture'

or frame specification the mechanism is given as  $\mathcal{M} = (\mathcal{L}, \mathcal{J}, \mathcal{F})$ . Links are enumerated with simple indices, whilst joints, together with their axes, are enumerated with double indices. This is done in order to reflect the topology of the mechanism. Frames are enumerated with triple indices to also reflect the links 'shared' between frames.

Having four frames per joint might seem redundant at first, however the two extra augmented, or transformed, frames ensures correct spatial displacement of the joints. As the joint center between two joints rarely coincide (due to geometry of the joints) one have to correct for an arbitrary displacement between the two elements of a joint. To exemplify, consider the chain of links illustrated in Figure 2.8. This configuration gives the frames  $F_{(ij)i}, F_{(ij)j}, F_{(ij\hat{k})}, F_{(jk)j}, F_{(i\hat{j}k)}$  and  $F_{(jk)k}$ <sup>1</sup>. Then, for frames attached to link  $L_j$  we define:

$$F_{D_j} := F_{(ij)j} \quad F_{C_j} := F_{(ij\hat{k})} \quad F_{A_j} := F_{(jk)j} \quad F_{B_j} := F_{(i\hat{j}k)}$$

Where  $F_{C_j}$  and  $F_{B_j}$  are the augmented frames.

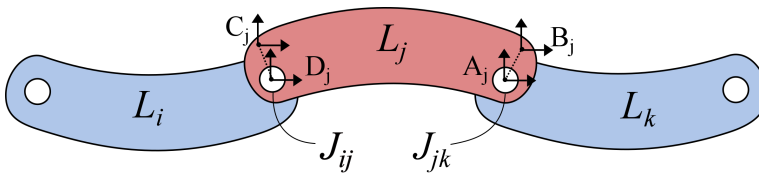


Figure 2.8: Arbitrary configuration of links  $L_i, L_j, L_k$

Rotations and displacements are represented with a homogeneous matrix  $\mathbf{M}$ . It incorporates linear rotation, via the rotation matrix  $\mathbf{R}$ , and linear translation, via the translation vector  $\mathbf{t}$ . This gives the following matrix:

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \quad (2.10)$$

<sup>1</sup>The 'hat' symbol is read as 'not' and denotes e.g. that frame  $F_{(ij\hat{k})}$  belongs to the link tuple  $(L_i, L_j)$ . Thus, it is the closest frame to the tuple which does not lie on the link containing the index  $k$ .

Then for spatial displacements (rigid body transformation) between the two frames  $F_D$  and  $F_A$  we get:

$$\mathbf{M}_{(D,A)} = (\mathbf{F}_D)^{-1} \cdot \mathbf{F}_A \quad (2.11)$$

Consider an arbitrary joint-link displacement  $D_{ijk}$  that maps from  $F_{(ij)_j}$  to  $F_{(jk)_j}$ . When decomposing it into joint the displacement  $D_{ij}$  and the link displacement  $D_{ijk}^*$ <sup>2</sup>, we get:

$$D_{ijk} = D_{ij} \circ D_{ijk}^* \quad (2.12)$$

Further, when decomposing the link displacement into screw displacement for the three twists  $(\mathcal{S}_{\tilde{c}_{ijk}}, \mathcal{S}_{\tilde{b}_{ijk}}, \mathcal{S}_{\tilde{a}_{ijk}})$ <sup>3</sup>, the displacement can be written as:

$$D_{ijk}^* = \mathcal{S}_{\tilde{c}_{ijk}} \circ \mathcal{S}_{\tilde{b}_{ijk}} \circ \mathcal{S}_{\tilde{a}_{ijk}} \quad (2.13)$$

Each of the screws, as illustrated in Figure 2.9, is made up of two parameters giving a total of six major parameters  $((\gamma, c), (\beta, b), (\alpha, a))$ , described in Table 2.1.

Table 2.1: Geometric meaning of the six parameters of the SU convention [6]

| Screw           | Axis                     | Parameter | Geometric description  | Alignment                 |
|-----------------|--------------------------|-----------|--|---------------------------|
| $\mathcal{S}_c$ | $\mathbf{z}_{ij}\hat{k}$ | $\gamma$  | Constant angular distance of $\mathbf{x}_{D_j}$ and $\mathbf{x}_{C_j}$ | Around $\mathbf{z}_{D_j}$ |
|                 |                          | $c$       | Constant linear distance of $\mathbf{x}_{D_j}$ and $\mathbf{x}_{C_j}$  | Along $\mathbf{z}_{D_j}$  |
| $\mathcal{S}_b$ | $\mathbf{z}_{ij}\hat{k}$ | $\beta$   | Constant angular distance of $\mathbf{x}_{C_j}$ and $\mathbf{x}_{B_j}$ | Around $\mathbf{z}_{C_j}$ |
|                 |                          | $b$       | Constant linear distance of $\mathbf{x}_{C_j}$ and $\mathbf{x}_{B_j}$  | Along $\mathbf{z}_{C_j}$  |
| $\mathcal{S}_a$ | $\mathbf{z}_{ij}\hat{k}$ | $\alpha$  | Constant angular distance of $\mathbf{x}_{B_j}$ and $\mathbf{x}_{A_j}$ | Around $\mathbf{z}_{B_j}$ |
|                 |                          | $a$       | Constant linear distance of $\mathbf{x}_{B_j}$ and $\mathbf{x}_{A_j}$  | Along $\mathbf{z}_{B_j}$  |

<sup>2</sup>The 'star' symbol indicates a dual space

<sup>3</sup>The 'tilde' symbol indicates a dual entity.

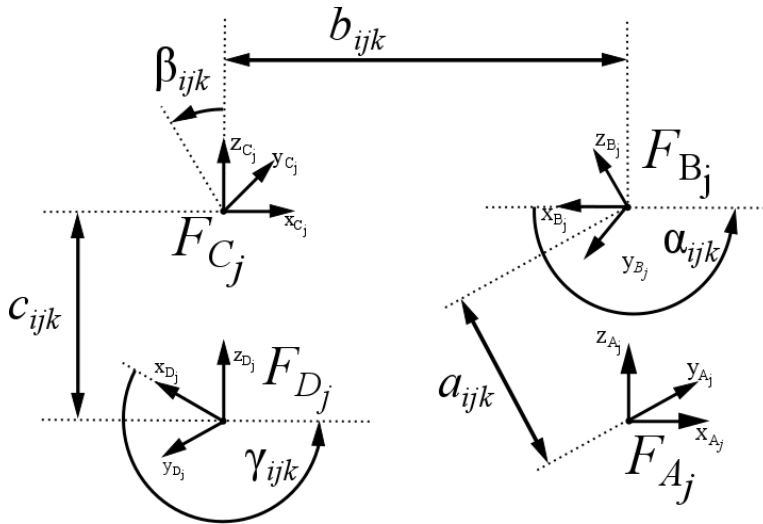


Figure 2.9: Example frame placement and SU parameters

## 2.4 Finite Element Analysis

In the context of structural mechanical analyses and simulations involving deformations and internal stresses, modeling problems can become quite complex. The problem might be defined in such a way that finding exact solutions cannot be done using the governing partial differential equations (PDE). Finite Element Analysis (FEA) is a numerical method for approximating solutions for these kind of problems, among others. FEA involves breaking the modeling domain down in a finite number of elements (meshing), to represent model behavior by approximating solutions for each element. A set of boundary conditions are applied to relate the elements to the environment and to each other.

### 2.4.1 Meshing

Model behavior is represented by dividing the model into a finite number of elements, thereby creating a mesh. Elements in the mesh are connected to each other through nodes, common edges and common surfaces, depending on the mesh dimensions. Figure 2.10, shows different types of elements, with a varying number of nodes in each element.

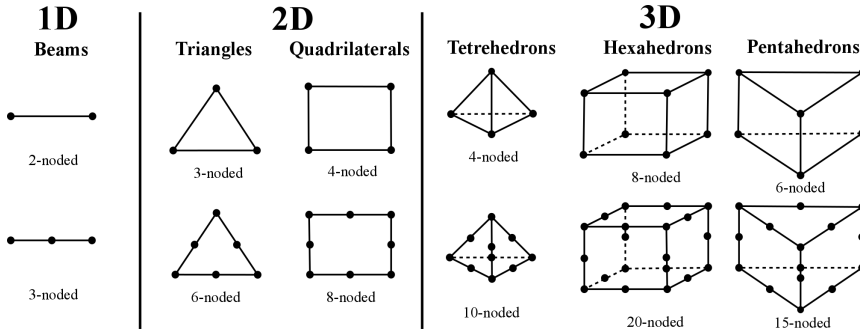


Figure 2.10: Example models and different mesh representations [27]

To calculate the element responses, e.g. average forces or stresses, the deformation of each element must be known. These are calculated through a series of *unit deformations*,  $\mathbf{u}$ , at each connecting node in the mesh. In short, this involves calculating an element's DOFs at a set of node points. When these equations, constituting the *stiffness matrix*,  $\mathbf{K}$ , are solved, the maximum stresses in the model can be calculated, given a set of *forces*,  $\mathbf{F}$ , and *constraints*, also known as boundary conditions. Equation 2.14 shows this fundamental relation [16].

$$\mathbf{K}\mathbf{u} = \mathbf{F} \quad (2.14)$$

## 2.4.2 Element Dimensions

The mesh types in Figure 2.10 depicts some of the ways to partition a mesh. Choosing the right type of elements, as well as their sizes, can be crucial in order to model a structure in the best way possible. There are several aspects to consider, some of them being the element's type, shape, and size.

The element type will either create structured or unstructured meshes. A structured mesh will consist of quadrilaterals in surface meshes and hexahedrons in volume meshes, while unstructured will consist of other polyhedron shapes, most commonly triangles in surface meshes and tetrahedrons in volume meshes. For models with a complex geometry, generating an unstructured mesh is usually much faster than generating a

structured one [10]. However, performing calculations on a structured mesh is usually faster. According to Felippa [16], structured meshes should always be preferred, and in volume meshes, tetrahedrons should only be used if there are no other options. Since the accuracy of mesh types might vary from a model to model, choosing which type to use might be a delicate process, depending on the situation. One solution can be to use a hybrid mesh, constituted of unstructured elements in areas with more complex geometry, and structured in more uniform areas. Figure 2.11 shows an example of such a hybrid mesh.

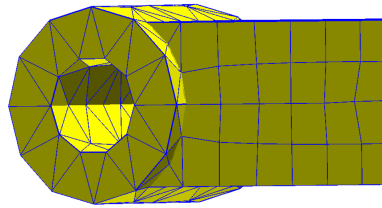


Figure 2.11: Link with hybrid mesh

The aspect ratio of the element shapes should also be considered when evaluating a mesh. In general, elements where one or more of the side lengths are much longer than any of the other, should be avoided. According to [16], if one of these ratios exceeds 3, the element should be viewed with caution, and those exceeding 10 with alarm. Figure 2.12 illustrates preferred and non-preferred elements as a result of a bad aspect ratio.

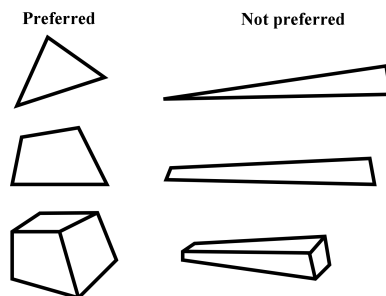


Figure 2.12: Element aspect ratios

Determining the element sizes proves to be an impossible task due to the character of FEA [20]. However, since FEA is an approximation to the real solution, it is possible to compare error estimates for different sizes. Take the element in Figure 2.13 as an

example. If we want to determine how much the relative approximation error decreases with the element size, we can start by looking at the element's translational function at its origin, which can be described by the Taylor series in Equation 2.15, where the element's origin is illustrated by the subscript  $0$ .

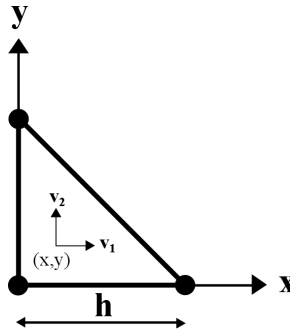


Figure 2.13: Example element

$$v_1(x, y) = v_{1,0} + \left(\frac{\delta v_1}{\delta x}\right)_0 x + \left(\frac{\delta v_1}{\delta y}\right)_0 y + \left(\frac{\delta^2 v_1}{\delta x^2}\right)_0 \frac{x^2}{2!} + 2\left(\frac{\delta^2 v_1}{\delta x \delta y}\right)_0 \frac{xy}{2!} + \left(\frac{\delta^2 v_1}{\delta y^2}\right)_0 \frac{y^2}{2!} + \dots \quad (2.15)$$

It can be shown that an interpolation polynomial of degree  $p$ , will give an error estimate,  $\Delta v_1$ , in the order of

$$\Delta v_1 = O(h^{p+1}) \quad (2.16)$$

where  $h$  is an element length, and  $O$  is the *big O* notation. A reduction of the element size, from  $h_1$  to  $h_2$  yields the following equation:

$$\frac{\Delta v_{1,2}}{\Delta v_{1,1}} = \left(\frac{h_2}{h_1}\right)^{p+1} \quad (2.17)$$

The element shown in Figure 2.13 has three nodes and its interpolation polynomial is linear ( $p = 1$ ). Reducing this element's size by a factor of 2, leads to Equation 2.18.

$$\frac{\Delta v_{1,2}}{\Delta v_{1,1}} = \left(\frac{h_2}{h_1}\right)^2 = \left(\frac{1}{2}\right)^2 = \frac{1}{4} \quad (2.18)$$

Consequently, a halving of the element size in a three-noded triangle element, reduces the relative translational error by a factor of 4. Calculating the same function for an element with a quadratic interpolation polynomial, for instance a six-noded triangle element, will reduce the error by a factor of 8.

Since stresses can be obtained through strains, which are the derivatives of the translations, the stress reduction ratios will be one degree lower than the interpolation polynomials for the translations. The stress error estimate will then be in the order of

$$\Delta \sigma = O(h^p) \quad (2.19)$$

In turn, this means that the relative stress error will converge to the real solution by one degree slower than the translations, when reducing the element size. In theory, this indicates that in areas where stress gradients are high, lowering the element sizes in the mesh, will reduce errors. In general, high stress gradients can be found around sharp corners, holes, cracks and in contact areas where load transfers are expected, illustrated in Figure 2.14.

From Equation 2.16, it is apparent that the error will converge to zero in two ways. The first way is by lowering the value of  $h$ , like the example in Equation 2.18. This is called *h-refinement*.  $h$ -refinement can be further categorized in *isotropic* and *anisotropic* refinement. In isotropic refinement, the element is divided into multiple elements by adding node points in both the  $x$ - and  $y$ -direction (for a surface mesh). In anisotropic refinement, new nodes are only added in one of the directions. The second way is by increasing the polynomial (increasing the number of nodes in the element), which is called *p-refinement*. If there is a desire to keep the number of elements and nodes in the mesh fixed, *r-refinement* can be applied. This involves resizing elements relative to one another, so that elements in high stress areas are smaller, while the rest are larger. Combinations of all three are also possible. Since the accuracy of a mesh cannot be



given before the problem is solved, a recursive process can be made for acquiring the desired mesh accuracy [20]:

1. An initial mesh and basis functions are chosen.
2. The problem is discretized and solved, and local error estimates are made.
3. If error is small enough, stop.
4. Perform h-, r- and/or p-refinement. Go to step 2.

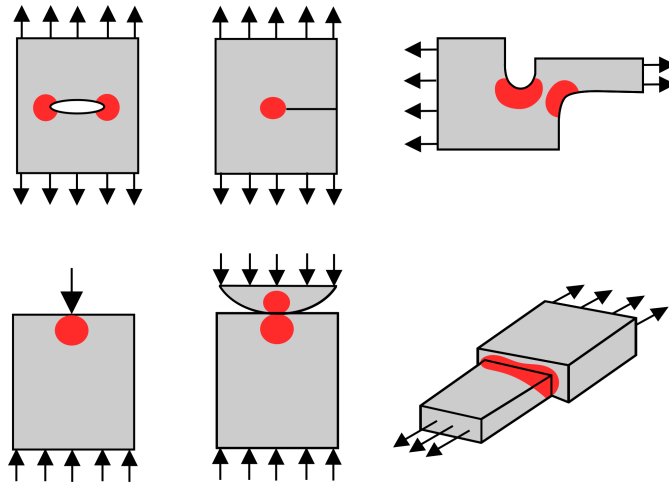


Figure 2.14: Typical high stress areas

### 2.4.3 Boundary Conditions in Structural Mechanisms

There are two types of boundary conditions, *essential* and *natural* [16]. Essential boundary conditions are conditions that directly influence an element's DOFs, while natural boundary conditions will not. In the domain of structural mechanisms, natural boundary conditions comprise the set of applied forces (the right-hand side of Equation 2.14). The essential boundary conditions concern displacements, and can further be divided into *symmetry conditions*, *ignorable freedoms*, *ground constraints* and *connection constraints*, where the two latter are the constraints of importance in this thesis. Ground constraints are defined to restrain rigid body motion, and can be thought of as bearings. Connection constraints, can be thought of as connection points between structures, which define how the structures' DOFs relate to one another, creating a rigid body

element (RBE). RBEs can either be a single point constraint (SPC) or a multi point constraint (MPC). The difference between SPCs and MPCs is that SPCs enforce a prescribed value for a single nodal displacement component, while MPCs enforce a prescribed value for two or more nodal displacement components. The prescribed value in question, is the relation between the DOFs of the two structures. A simple example can be a joint connecting two structures (links). Depending on the type and nature of the joint, it will relate the two links' nodal displacements (DOFs) to each other, thereby creating a rigid body. An SPC will only have one node dependent on the constraint imposed by the joint, but an MPC will have two or more [25].

There are two types of MPCs of interest in this thesis, RBE Type 2 (RBE2) and RBE Type 3 (RBE3)<sup>4</sup>. To start with, the RBE2 will define a master-slave relationship between its nodes [14], where the master node, also known as the independent node, will dictate the relative movement of the slave nodes, also known as the dependent nodes. As illustrated in Figure 2.15, four slave nodes are dependent on one independent node. The DOFs of the dependent nodes are solely dependent on the DOFs of the independent node. The independent node will always have six DOFs, but the dependent nodes can either have six DOFs, simulating a weld, or three DOFs, simulating a bolt. The dependent nodes can not move relative to one another, thereby adding stiffness to the mesh model, endorsing the rigid body motion.

RBE3 works a little differently. Despite its name, the RBE3 does not create a fully rigid element, like the RBE2. The motion and DOFs at an RBE3 dependent node, is the weighted average of a set of independent nodes, illustrated in Figure 2.15. This means that a dependent node will have different responses for each of its independent nodes. Likewise, a force applied to the dependent node, will be distributed among the independent nodes, with a weighted factor. In that sense, the RBE3 can be thought of as a force distributor in a free body [42]. Each weighting factor has to be defined when the RBE3 connection is set up, either specified directly, or by an algorithm.

---

<sup>4</sup>RBE2 and RBE3 are NX Nastran names for these MPCs. IN FEDEM, they are called RGD and WAVGM, respectively.

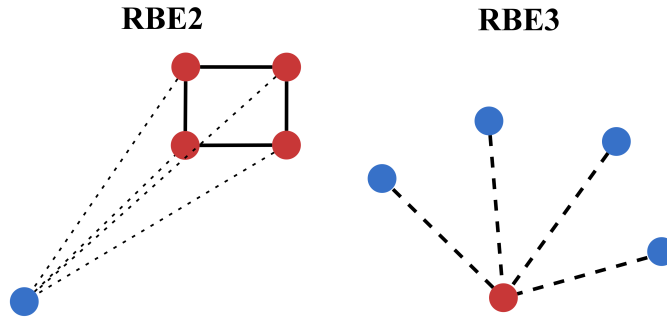


Figure 2.15: RBE2 and RBE3

## 2.5 Control Systems

A control system is a device or process that regulates the behavior of another device or process with regard to a particular condition [4]. Control systems are separated into open- and closed-loop systems. In open-loop systems the output of the system has no effect on the control action, as seen in Figure 2.16a. Open-loop systems are also called calibrated or unmonitored systems as the output is neither measured or fed back to the system for control. An example of an open-loop system is a washing machine. The soaking, washing and rinsing operates solely on the measurement of time, not the cleanliness of clothes. Closed-loop, or monitored, systems on the other hand, feeds back portions of the output and reuse this information in coherence with the input, as illustrated in Figure 2.16b. An example is a thermostatically controlled water heater, where the temperature of the water is measured and kept above and or under a given temperature.

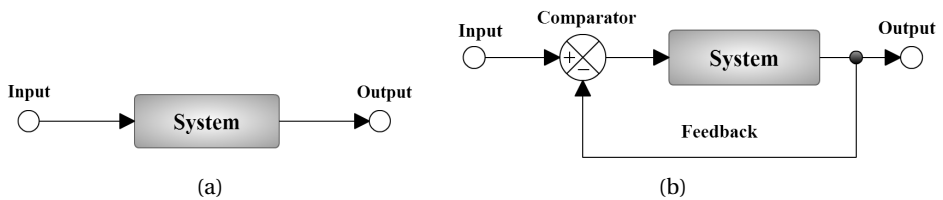


Figure 2.16: Open-loop and closed-loop system

## 2.6 Design Optimization

Section 2.1 describes how the designer can take a step back in the design process, if the design fails to satisfy the design specifications, creating a feedback loop. The feedback, generally a response from an analysis, needs to be evaluated in order to determine which parameters to modify. Experienced designers may be able to know which parameters to modify solely based on their skills and intuition, which can be a preferred approach when it comes to designing simple systems. However, it becomes very difficult, or even impossible, for complex systems. An alternative approach is to include design optimization methods. By incorporating optimization algorithms as a computational tool, the computer can give exact suggestions as of how to better the design. These suggestions can both guide the designer to make better designs as well as reduce the time-to-market since the designer will be relieved from repetitive, time-consuming calculations.

In short, design optimization is the act of finding the best performance of a system under given constraints. The formulation can be divided into three steps, the first being to identify a set of variables that describe the system - *design variables* [11]. By altering the numerical values of these variables, alternate designs of the system can be obtained. The second step is to specify requirements on the design and performance which the system must satisfy. All requirements imposed are called *constraints*. The final step is to specify a criterion to judge whether a given design is better than another. This criterion is called the *objective function* and is a performance index on which the system's performance can be measured.

### 2.6.1 Design Problem Formulation

To put it in mathematical terms, the design problem can be defined as finding exact values for a set of design variables,  $\{x_1, \dots, x_n\} = \mathbf{x}$ , which minimize the objective function:

$$f(\mathbf{x}) \tag{2.20}$$

that satisfy a set of inequality and equality constraints:

$$g_j(\mathbf{x}) \leq 0 \quad j = 1 \dots m \quad (2.21a)$$

$$h_j(\mathbf{x}) = 0 \quad j = 1 \dots p \quad (2.21b)$$

and a set of move limits:

$$x_i^u \geq x_i \geq x_i^l \quad i = 1 \dots n \quad (2.22)$$

A collection of  $n$  independent design variables will create an  $n$ -dimensional space. The subset of this space, constrained by the design variables' upper and lower bounds ( $x_i^u$  and  $x_i^l$ ) and the inequality and equality constraints, forms what is commonly referred to as the *design space*. Thus, the design space is the space containing all allowable configurations of the design variables, according to the problem formulation.

It is conventional to normalize the constraints, to ensure equal order of magnitude. This normalization helps avoiding numerical difficulties, by making each constraint lie in the interval  $[0, 1]$ . For example, a stress constraint,  $\sigma(\mathbf{x})$ , could have an upper limit of  $\sigma^u$ . To make the constraint value fall within the given interval, the normalization would become:

$$g(\mathbf{x}) = \frac{\sigma(\mathbf{x})}{\sigma^u} - 1 \leq 0 \quad (2.23)$$

which also is used in the optimization example in Chapter 5.

Design optimization of mechanisms often includes a response vector,  $\mathbf{s}$ , acquired through dynamical simulations. The response vector is usually a time history describing the system's responses at a given time,  $t$ , but does not have to be. Time histories typically describe displacements, velocities or accelerations, and they are also functions of the design variables, such that  $\mathbf{s} = \mathbf{s}(\mathbf{x}, t)$ . This response will impose further constraints on the system, which in turn alters the constraint functions:

$$g_j(\mathbf{x}, \mathbf{s}(\mathbf{x}, t)) \leq 0 \quad j = 1 \dots m \quad (2.24a)$$

$$h_j(\mathbf{x}, \mathbf{s}(\mathbf{x}, t)) = 0 \quad j = 1 \dots p \quad (2.24b)$$

Like mentioned, there are many different types of design variables. These types can be categorized in the following categories [44]:

- **Substructure:** Design variables that affect the inherent properties of a system substructure, i.e sizing variables, material properties and shape variables.
- **System:** Design variables that only affect input data for an assembled mechanism. Can be further categorized into:
  - **Structural:** Typically input parameters, like load parameters, coordinates of attachment points (of springs, dampers or hinges), values or lumped nodal masses, and initial velocity values.
  - **Control:** External parameters for a general control element, like gain parameters, coefficients for comparator-, summer-, integrator- and derivator elements, sample and delay time, and limits and slopes of piecewise continuous elements.

Constraints can also be categorized, depending on the values of the design variables. The design variable values that leaves the constraint function in an infeasible area, i.e.  $g(\mathbf{x}) > 0$ , means that the constraint is violated. If  $g(\mathbf{x}) < 0$ , the constraint is passive, and if  $g(\mathbf{x}) = 0$ , then the constraint is said to be active (but still in the feasible area). These categories are illustrated in Figure 2.17, where an objective function is subject to a single constraint, and the line 'A' is drawn at the value of design variable,  $x_1$ .

Given the problem formulation, there are many different optimization methods that can be utilized, where some might be more applicable than others. Some formulations might for instance be unconstrained, some might need gradient information of the design variables to be effective, and some might include multiple objectives. However, all methods described in this section have in common that they are numerical, i.e. they are iterative processes applied to converge as best as possible to an optimal solution.

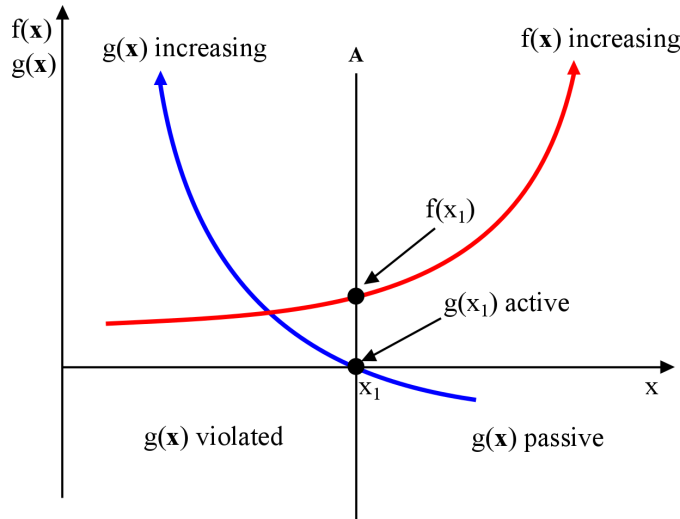


Figure 2.17: Constraint areas

## 2.6.2 Unconstrained Methods

Unconstrained optimization methods are methods where there are no restrictions on the design space except those imposed by the move limits of the design variables. As such, they form the basis of other methods used in optimization systems. In general, these are direct search methods, and can be categorized in relation to the differential order used in the algorithm.

Starting with *zeroth* order algorithms, these methods have no requirements on the differentiability of the objective function, and will only evaluate function values. Only two zeroth-order algorithms are considered useful [19], the *Nelder-Mead method*, also known as the *sequential simplex method*, and *Powell's conjugate direction method*. Both of these methods are part of the AMOpt library in Section 3.3.5, so they will be described in short terms. The Nelder-Mead method involves searching an  $n$ -dimensional space using a simplex, which is an  $n$ -dimensional object with  $n+1$  vertices. The idea is to move the point of the simplex where the objective function gain the highest value, through the opposite face of the simplex, thereby making it lie in an area that has a lower value, until it approximates a local minimum. Powell's method is a method that utilizes conjugate directions to quadratically converge to a minimum. This is a search that moves along a set of directions that is conjugate to the objective function. It requires at most

n iterations. Both of these methods will usually converge to a minimum, either local or global, which means that the objective function must only have one minimum value (unimodal) for these methods to be applicable. According to Press [31], Powell's method will almost surely propagate faster than the Nelder-Mead method.

First and second order algorithms are more frequently used. These utilize gradients and the function value in order to calculate the direction of which to move in the next iteration step. An example of a first order method is the *steepest descent method*, and some second order method examples are *Newton's method*, *quasi-Newton methods* and *conjugate gradient methods*. As these methods are not utilized in this thesis, they will not be described further.

It is worth noting that according to Sobieszczanski-Sobieski [40], unconstrained direct search methods have little relevance for standard optimization problems. However, constrained problem definitions can be transformed to the aforementioned unconstrained ones, by augmenting the objective function. These methods are called Sequential Unconstrained Minimization Techniques (SUMT), and are often used to solve nonlinear constrained optimization problems. Generally speaking, SUMT incorporates constraint functions in the objective function by utilizing *penalty functions*. Penalty functions are just a way to control the value of the objective function when the design variables move into previously constrained areas. These functions form a self-correction, but if the constraint violation is too large, this process may not work, causing the next iteration to start too far away from a feasible design area. As claimed by Specht [43], using SUMT for solving structural optimization problems, will not provide any computational savings, compared to solving the problems directly.

### 2.6.3 Constrained Methods

Many of the unconstrained optimization methods mentioned can also be applied in constrained optimization. In these cases, constraint methods are explicitly incorporated into the solution process. The optimization is now done by projecting the unconstrained direction onto active constraints. Finding the optimum of the objective function is no longer the same as finding its minimum value, but rather a point where



it is impossible to move without violating a constraint or making the objective function larger.

Constrained optimization algorithms have in common that the objective function can be approximated by linear or quadratic functions, and the constraint functions either are linear already, or can be approximated by linear methods. An important note is that the approximations are calculated *at the current iteration point*, in sequence. Thus, the case that approximates linear functions for both objective and constraint functions, is called Sequential Linear Programming (SLP) [49]. SLP problems can be efficiently solved, even without gradient methods, by for instance the Nelder-Mead method.

The case that approximates the objective function by a quadratic function, and the constraint functions by linear functions, is called Sequential Quadratic Programming (SQP). SQP methods have good local convergence properties, but will often require that the search for an optimum starts at a feasible design point. Hence, they are the standard of solving smooth nonlinear optimization problems. However, they can experience difficulties converging to a global optimum, if the problem involves many design variables and constraints. An example SQP method is Newton's method in combination with a method that transforms the problem formulation into an equality constrained problem.

#### 2.6.4 Multi-Objective Optimization Methods

In multidisciplinary design problems, there are often conflicting interests, like for instance minimizing costs while maximizing performance, and so on. Optimization of such methods might therefore seem incompatible with the methods previously described, but in general there are two options. The first is to combine all objectives into one function, where each objective might have a unique weight to distinguish it by. Setting these weights accurately can be rather difficult, so the second option might be preferable, which is to create a set of objective functions. Optimization methods with such problem formulation are called *Multi-Objective Optimization Methods* (MOOM). Possible implementations for MOOM is the family of *Genetic Algorithms* (GA).

## Genetic Algorithms

GA are based on the theory of evolution, and are typically used to solve general optimization problems. As such, they differ from the direct search methods because they do not iteratively search from one design point to the next. Instead, they progress iteratively from a set of design points, called a *population*, to another [40]. Each population set forms a *generation*, and the idea is to improve the objective function from generation to generation. This is done by evaluating the design points in each population, through a *fitness function*. The two design points with the best fitness evaluation will be chosen as 'parents' for the next generation. The design points, or 'offspring', will be based on their parents by using a *crossover operator*. Then, each design point have a predefined probability of *mutation*, where a mutator function alters its values. The offspring that pass a feasibility evaluation are passed to the next generation, until the population limit is reached. When either the maximum number of generations is reached, or a stopping criterion is satisfied, the design point that best satisfies the objective function is chosen.

The reason that GA are well suited for MOOM is that they are able of searching different regions of the design space, simultaneously. This makes it possible to find a diversified set of solutions for problems that numerous local minima. In addition, a Multi-Objective GA (MOGA), do not require the user to set weights for the objective functions.

### 2.6.5 Structural Optimization

In structural optimization problems, there is a desire to reduce the number of structural analyses and retrieve feasible designs at each iteration step. *Convex approximation methods* is a suitable answer to this desire.

#### Convex Approximation Methods

A function is convex if a straight line connecting two arbitrary points on the function graph, lies nowhere below the graph [49]. Thus, they guarantee that any any minimum point is a global minimum point. Also, an iteration leaving the objective function smaller than in the previous iteration, makes it possible to stop the iteration process at

any time, when a suitable design has been reached.

The general idea of convex approximation methods is to approximate the objective and constraint functions as convex functions, and then apply an optimization method, for instance an SQP method, for each step. Exactly how this is done lies a bit outside the scope of this thesis and will not be explained further.

### **Sensitivity Analysis**

Another feature to incorporate in a structural optimization process is *sensitivity analysis*. Section 2.6.1 described the use of a response vector,  $\mathbf{s}(\mathbf{x}, t)$ . The results obtained in this vector give little information of the design variables that will give the most positive feedback for the objective, when perturbed. However, this information can be obtained through the gradients of the response, with respect to the design variables, and is called sensitivity analysis. Mainly, it is used to guide the optimization process, but given enough design variables, calculation of gradients can become quite time-costly, and should be evaluated before applied.

### **2.6.6 Final Notes**

When incorporating design optimization in a KBE environment, KBE becomes a supporting technology that can bring together all aspects of the optimization process in a software package [40]. Much of the KBE philosophy is to automate repetitive design tasks, and the process of design optimization can in many cases be seen as highly repetitive. According to Etman [11], a successful optimization requires the designer to not be completely left out of the optimization loop. Some amount of user-interaction is required in order to control the process in the best possible way. A semi-automatic optimization process can therefore be considered optimal in many cases.

## 2.7 Software Development

Software development is the process in computer programming that results in a software product. The software development methodologies used in this thesis is object-oriented development and Scrum.

The software development process can be split into distinct stages with activities, depending on where in the life cycle the software is. These stages typically reduce down to the following: requirement specification, system analysis, system design, implementation, testing, deployment and maintenance. Our thesis address only the first four stages, thus these will be the main focus of discussion.

### 2.7.1 Object-Oriented Development

Object-oriented development decomposes a system based upon the concept of an object [5]. Object-oriented development is typically divided into three main activities: analysis (OOA), design (OOD) and programming (OOP). OOA and OOD is the process of analyzing, designing and separating a system into meaningful classes and relations. OOP is the process of implementing this design as a set of objects.

Some key concepts of the object-oriented approach are: class, object, messages, encapsulation and inheritance. A class can be thought of as a template, or abstract data type, as its purpose is to declare the actions (methods) and data fields (attributes) of objects. Thus, an object is really just an instance of a class, characterized by its set of methods and attributes. An object is self-contained and designed to be easily used and understood. Attributes and methods are imposed with restrictions and encapsulated, and the objects interact with each other through, what is generally called messages.

Encapsulation is the process of grouping related information and protect it from the outside world [29]. Appropriately grouping and hiding implementation details of an object reduces the overall system complexity. Inheritance allows one class to inherit characteristics from another, thus models the 'is an extension of' relationship.

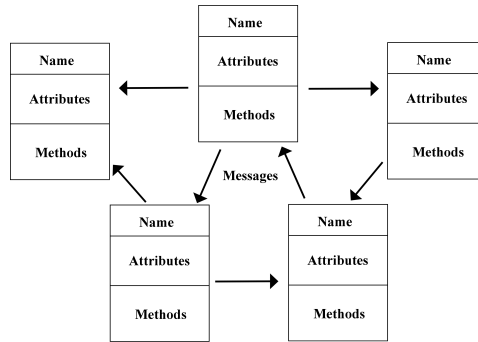


Figure 2.18: An object-oriented system consists of many well-encapsulated objects, interacting with one another by sending messages

### 2.7.2 Scrum

Scrum is an iterative, agile framework which defines a holistic strategy where the development team works as a unit to reach a common goal [45, 35]. It emphasizes communication, collaboration and team flexibility. Scrum accepts problems that can not easily be understood and focuses instead the team's ability on fast deliveries. This enables the team to better adapt to changing requirements and technologies.

Scrum introduces the concepts of sprints to help guide the team's workflow. A *sprint* defines a period of time, typically about one to four weeks, where a development team is set to carry out a given amount of work. Work is pulled from a product backlog, which is made beforehand, and put into the sprint backlog, as seen in Figure 2.19. Daily scrum meetings are encouraged in order to keep track of the team's progress and to promote collaboration. Another widely used concept, not only in Scrum but software development in general, is pair programming. Pair programming is an agile software development technique in which two programmers work together at different levels of abstraction. Typically there is one 'driver', which writes the code, while the other reviews each line that is written as the 'navigator' [3].

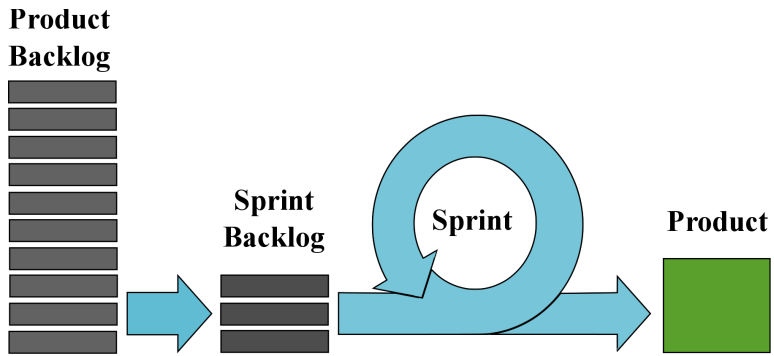


Figure 2.19: The scrum process

# Chapter 3

## Methodology

### 3.1 Runtime Environment

Development was conducted on 64-bit computers running Microsoft Windows 10 Home. The following software was used: AML, MSC Nastran, FEDEM and Notepad++. Four AML modules was also used: AMSketcher, *aml-analysis-module-pack-type-3-01-06*, *aml-analysis-module-pack-type-3\_ui* and AMOpt. In addition, the mechanism system requires a series of prerequisites in order to run. These are described in full under installation details, Appendix A.

### 3.2 Development Infrastructure

Software development between multiple developers and computers can make backup, merging and sharing code challenging, especially since developers might work on different versions of the system. In order to enhance collaboration and ensure proper storing of versions, both now and in the future, we set up a development infrastructure in GitHub. GitHub is a stand-alone version control system. It hosts a web-based repository with distributed revision control and source code management [18]. This enables multiple developers to work freely on different and/or common parts of the system without having to worry about losing code.

Along with the repository, a private 'NTNU-IPM' user was created to ensure that the

proper administrator rights of the repository were retained by the IPM department. The repository, containing all of the source code and mechanism library can be viewed by visiting '<http://ntnu-ipm.github.io/KBE/>'.

In addition, a backlog, containing the work that was needed to be done has been set up. The setup is controlled using an online planning tool, called Pivotal Tracker [30]. Pivotal Tracker lets users plan work ahead, by adding definable units of work, called stories, to a project backlog. This planning tool is used in line with the Scrum methodology, and served as a great tool for controlling the project process.

### 3.3 Adaptive Modeling Language

AML is a modeling language for concurrent engineering created by TechnoSoft Inc. [47]. It is based on the List Processor (LISP) programming language with an underlying object-oriented nature. AML enables multidisciplinary modeling and integration of entire products and processes throughout the whole development cycle [46]. Computations in AML are done in a demand-driven fashion, by using automatic dependency tracking between objects and properties, in order to trigger calculations only when they are required.

#### 3.3.1 Framework

The AML framework, as seen in Figure 3.1, consists of several modules, each with different functionality for different knowledge domains. This enables the developer to 'adapt' AML to his/her specific needs. E.g. if there is no need for graphics or geometry, then this module can be removed - allowing for just the kernel to be loaded. The AML kernel provides the language constructs for defining classes, methods etc. [46].

#### 3.3.2 Editor

AML includes and encourages the use of XEmacs, which is a graphical- and console-based text editor [52]. However, due to structuring of the source and general efficiency reasons, we decided to write and edit the code in Notepad++, and then compile the



code in the XEmacs console. Notepad++ is a text and source code editor which supports a wide range of programming languages, and allows a high level of user customization [26].

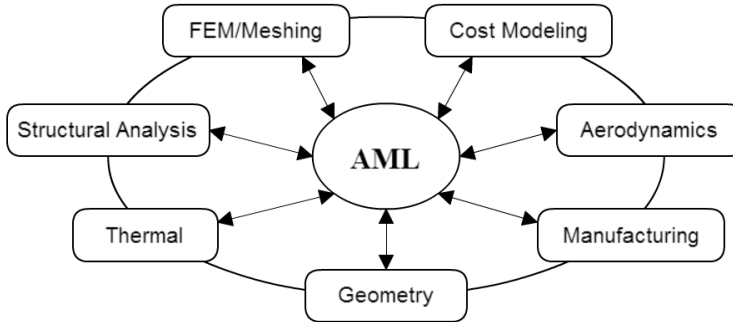


Figure 3.1: The AML framework

### 3.3.3 Source Code Management

In order for the source code to be easily loaded, compiled and reused, TechnoSoft recommends the use of *systems*. In AML, a system groups all of the source code files together so they can be treated as a single module, hence loaded, compiled and archived as a single entity. When the system compiles, binaries for multiple platforms are also created - allowing different platforms to operate within the same system version. In order to compile and load the system, a *logical-path* has to be created. A *logical-path* locates a resource in the file system by mapping variables to the path-reference of the resource.

The system is defined in a *system.def* file. The *logical-path* of the system has to point to the location of the *system.def* file. The *system.def* consists of the *define-system* construct, which is the main mechanism for creating systems. This is where the name of the system is defined along with the source code files that comprise the system.

### 3.3.4 AML Modeling Forms

The models and geometries created from the source code can be viewed and edited in a modeling form. AML comes with a couple of options depending on the need of interactively editing the geometry. The most standard form is the *Main Modeling Form*,

where objects can be drawn and inspected, and custom made GUI frames are shown. A more advanced form is the *AMSketcher* module, which bears more resemblance to a CAD modeling form. In addition to the functionality from the Main Modeling Form, *AMSketcher* provides menus for directly working with the models, where different geometries can be made from a set of primitive, or basic, geometries. For example, typical operations like extrusion, sweeping and blending are possible. The new and altered geometries will then be added to the model as objects. All objects are shown in a *Model tree*, and can be edited in the *Inspect form*. The *Inspect form* shows all object properties, as well as the underlying formulas for the properties. These formulas can be edited in run-time, thereby removing the need to recompile when testing small changes.

### 3.3.5 AMOpt

AMOpt is a module containing basic tools for design optimization and design studies. It is integrated with AML, thus it can be used from the modeling interfaces. Its functionality is mainly described through its library of optimization methods:

- Multi-Objective Genetic Algorithm
- Powell's Method
- Nelder-Mead Simplex Method
- Design of Experiments
- Monte Carlo Simulation

The three first methods are described in Chapter 2.6, but the last two are not, as they fall more under the probabilistic design studies category, than the design optimization category.

According to TechnoSoft [46], two other optimization libraries can also be integrated, *Design Optimization Tools* (DOT) and NPSOL. Both of these provide optimization methods not available in AMOpt, like for instance SLP and SQP methods [51, 41]. However, these libraries are license-based, and have not been tested.

Through the AMOpt interface, optimization problems can be set up and solved through any of the available methods. However, an optimization case needs to be programmed manually, as objective functions, constraint functions and design variables often are not simple properties of the mechanism model.

The methods will run an automatic, non-interruptible optimization before they conclude with an optimal design, given the problem formulation. No source code or documentation of the implementation of the AMOpt interface is provided, and therefore it can be treated as a black-box application. Unfortunately, this means that the final optimized result can not be obtained programmatically, only visually, which is a problem for the designer. However, workarounds are described in Chapter 5.

### 3.4 FEDEM

FEDEM is a general multi-purpose software package for analysis and virtual testing of structural systems. The name FEDEM is an acronym for Finite Element Dynamics in Elastic Mechanisms. FEDEM is based on: (i) a non-linear finite element (FE) formulation, (ii) a master-slave joint formulation, and (iii) a component mode synthesis (CMS) reduction [13].

Mechanisms in FEDEM are represented at two levels, the *substructure level* and the *system level*. The substructure level represents the FE part, while the system level represents the mechanism, or multi-body system, part. The FE model of a substructure contains a number of nodes. These nodes are divided into *external* and *internal* nodes, with corresponding DOFs. The external nodes typically represent points of interest for joints, springs, dampers, loads, sensors etc. During the model reduction of a substructure to a *super element* the external nodes are retained as *super nodes*. A super element is simply a term used for a link when it is represented by a substructure with a reduced number of DOFs. A *system model* is created at the system level by connecting joints, springs, dampers etc. to the super nodes [14].

The master-slave joint formulation express the kinematic relations between links by defining slave and master nodes. The joint type is then defined by how these nodes can

move relative to one another. Joints can, in turn, be combined to form transmissions.

Dynamic analyses are, in general, more expensive than static analyses. However, the CMS model reduction and the use of super elements reduce the cost of dynamic computations. This is achieved by reducing the number of DOFs by assuming that the low-frequency modes of vibration are the most important. The CMS reduction replaces the internal DOFs with a smaller number of modal DOFs. These DOFs represent the possible vibration modes of the substructure, typically those with lowest eigenfrequency, and are used as system DOFs in the system model. If no modes are included the reduction becomes a *static condensation*. The model reduction is done prior to the assembly of the system model [14].

During a dynamic analysis, the super nodes might move due to rigid and elastic body motion of the super elements. Deformation of super elements is calculated by looking at the updated position, local coordinate system, of the super nodes. The responses from the dynamic analysis can be divided into substructure responses and system responses. The responses are calculated by numerical integration (the Newmark integration algorithm) of the equations of motions [49].

Note that solver processes in FEDEM can either be executed directly from the application GUI, or by creating batches from a command-line prompt [15]. The latter is particularly beneficial when working on several versions of a model, or as a part of an automatic analysis. Section 6.2.7 describes how this has been used.

### 3.5 Modeling of the Mechanism System

Software engineering systems often become quite complex, for example due to large and intricate code bases, and multiple connected environments. Thus, they are inherently difficult to comprehend, thus need to be documented, both descriptively and visually. Unified Modeling Language (UML) is a convention created by the Object Management Group (OMG) [28], used for this exact purpose, as it is a standard for visualizing the design of a system. Through a combination of notations from object-oriented design, object modeling techniques and object-oriented software engineering, it presents a consistent methodology that is both visual and easy to use. UML is mostly used to determine system requirements and details regarding system implementation.

One of the UML constructs is the *class diagram*. Class diagrams represent a static view of the building blocks that make up an object-oriented system. A class diagram depicts the different relationships between classes and interfaces. Inheritance, composition, aggregation and usage are all given different syntax to indicate the behavior of and messages between classes.

AML can be characterized as a frame based system (FBS), not to be confused with coordinate systems (Section 2.3.5). According to La Rocca [34], a frame is *“a collection of slots to describe attributes and/or operations featured by a given object. Each slot may contain a default value, a pointer to another frame, or a set of rules (including IF-THEN rules) and procedures by which the slot value is obtained.”* Generic frames are called class frames and their instances are called object frames. Adopted from this frame definition, we have introduced a custom convention where both objects and classes are depicted in the same diagram. The reason for this is twofold: (i) subobjects in AML are defined directly in the class definition, thereby amplifying the importance of the object instance, and (ii) having both objects and classes in the same diagram has proven vital in order to keep track of the flow of information in the mechanism system. The basis for the extended convention can be seen in Figure 3.2, and will be used for documentation in Chapter 6.

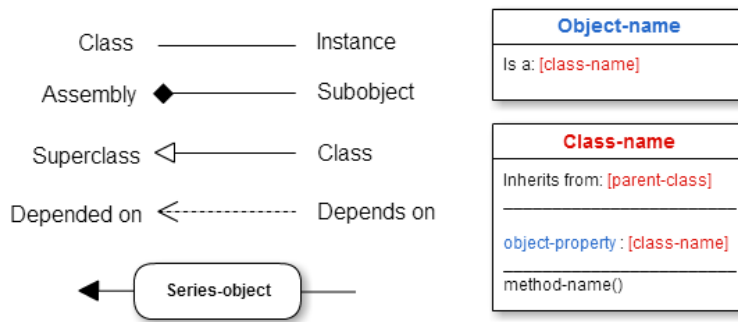


Figure 3.2: Class diagram extensions

To easily distinguish between classes and objects, this convention uses red characters for classes and blue for objects. All the relations in Figure 3.2 are commonly used in UML, with the exception of the 'Series-object' relation. This indicates that the class at the start of the arrow, creates the class that is pointed to, using the *series-object* class in AML (described in Chapter 6).

# Chapter 4

## The Mechanism System

The pilot implementation of the mechanism system comprised a general system synthesis for mechanism design. The system included fundamental link and joint definitions, as well as a kinematic model for placement of frames. The system was able to create and mesh generic mechanisms, consisting of different links, joints and surfaces between straight members. Additionally, an initial standalone framework for RBEs were commenced. However, the system was not stable and struggled with lacks in both the kinematic model and the joint definition. In the kinematic model, certain joint orientations could cause the incident members to get undesirable twists and bends, as a result of incorrect augmentation of sub-frames. Joints was defined to always have two interconnected, visible links. This yielded several errors and null geometries<sup>1</sup> when a joint was connected to a 'ground link' and not an actual geometry. The only mesh available was a triangular surface mesh with a constant mesh size.

This chapter address the main components and functionality of the mechanism system, encompassing both improvements and additions.

### 4.1 Application Input

The mechanism system uses a total of six different input files. These files form the basis of how a mechanism is represented in the system, and are as such, the most important

---

<sup>1</sup>The null geometry error occur when AML tries to draw impossible geometry.

tool a designer has for modeling a mechanism. The structure of the files have been iteratively improved in order to make them as human-readable and intuitive as possible. Note that the files have two things in common: Firstly, the row values are tab separated, meaning that each new value must be separated by one or more tabs; Secondly, the first row in each file will be ignored, as it only contains headers. The files will be described in the following subsections.

### 4.1.1 Node Positions

The mechanism system utilizes nodes for defining key positions in a mechanism. The different nodes include: joint positions, connecting points for springs and dampers, point of attack for a force or torque and design points for link geometries. These nodes represent the external nodes of a mechanism (when not considering control points). The input file is called *coordinates.txt* and contains information about the node names and their respective coordinates. An example input file is shown in Table 4.1.

Table 4.1: Example coordinates input file

| Index | Name                       | X-pos | Y-pos | Z-pos |
|-------|----------------------------|-------|-------|-------|
| 0     | "Revolute-joint"           | 0.0   | 0.0   | 0.0   |
| 1     | "Ball-joint"               | 0.25  | 0.0   | 1.0   |
| 2     | "Free-joint-1"             | 0.1   | 0.5   | 0.0   |
| 3     | "Rigid-joint"              | 1.25  | 0.5   | 1.0   |
| 4     | "Free-joint-2"             | 1.25  | -0.5  | 1.0   |
| 5     | "Control-point-1"          | 1.75  | 0.25  | 0.0   |
| 6     | "Control-point-2"          | 1.3   | 0.5   | 0.0   |
| 7     | "Spring-damper-connection" | 0.5   | 0.0   | -1.0  |
| 8     | "Spring-damper-ground"     | 1.5   | 0.0   | 1.0   |

The reason for the explicit statement of indices is just to make the files more human-readable, e.g. when referring to a node from other input files.

### 4.1.2 Constraints

The joints are defined in the input file called *constraints.txt*. The file specifies joint type, which links the joint is incident on, the direction, i.e. the orientation of the joint and the joint's DOFs. An example input file is shown in Table 4.2.



Table 4.2: Example constraints input file

| Point | Type       | Link-incidence | Joint-direction | Fixed-DOFs    |
|-------|------------|----------------|-----------------|---------------|
| 0     | "revolute" | (nil 0)        | (-1 0 0)        |               |
| 1     | "ball"     | (0 2)          | (0.1 0 1)       |               |
| 2     | "free"     | (0 nil)        | (0 0 1)         |               |
| 3     | "free"     | (1 0)          | (0 1 0)         | (1 2 3 4 5 6) |
| 4     | "free"     | (2 nil)        | (0 -1 0)        | (4 5 6)       |

To start with, the numbers in the 'Point' column refer to the corresponding node in *coordinates.txt*. The 'Type' column indicates the joint type, annotated in quotation marks. 'Link-incidence' defines the two links that the joint constrains. For example, on the third line, (0 2) means that link 0 and link 2 are constrained by a ball joint. Furthermore, it is also implied that the first element (0) is the solid, *male*, part of the joint, while the second (2) is the hollow, *female*, part, as explained in Section 2.3.2. One of these values can also be *nil*. In AML, *nil* is a placeholder for an empty set, so *nil* in this context simply means that the given element should not exist. Taking the first input line as an example, (nil 0) means that link 0 should have a female revolute element, located at point 0, and not connected to any male element. In the mechanism system, a joint having a *nil* element is a direct indication that it is supposed to be connected to ground. Ground constraints are described in Section 2.4.3.

Free joints have no joint geometry and are initially defined to have six unconstrained DOFs, as explained in Section 2.3.2. These DOFs are specified in the 'Fixed-DOFs' column. For instance, (4 5 6) means that the joint is free to translate along the x, y and z direction but cannot rotate about any axis. Analogously, (1 2 3) means that the joint is free to rotate about its x, y and z axis but cannot translate in any direction.

Finally, notice that 'Fixed-DOFs' is left blank for every other joint type than free. This is because a joint's DOFs is explicitly given by its type, except for free joints.

### 4.1.3 Link Shapes

The general appearance of the links are defined in *shapes.txt*. Example entries are listed in Table 4.3.

Table 4.3: Example shapes input file

| Name            | Link | Member | Cross-section | Dimensions        | Points-list | Weights-list |
|-----------------|------|--------|---------------|-------------------|-------------|--------------|
| "Knuckle-m-0"   | 0    | 0      | "rectangular" | (0.1 0.05)        |             |              |
| "Knuckle-m-1"   | 0    | 1      | "nil"         |                   |             |              |
| "Knuckle-m-2"   | 0    | 2      | "hexagonal"   | (0.1 0.05)        | (5 6)       | (0.4 1)      |
| "Steering-link" | 1    | 0      | "i-beam"      | (0.6 0.1 0.2 0.1) |             |              |
| "Upper-wish"    | 2    | 0      | "h-beam"      | (0.6 0.1 0.2 0.1) |             |              |

Here, each link member has specified its type of cross section and its dimensions. The dimensions are defined either as (*height width*) or (*height-start width-start height-end width-end*). Height and width refers to the y and z direction, respectively, in the cross section's local coordinate system. The '-start' and '-end' postfixes defines the same values, only for the start and end cross sections of the link member.

The values under 'Points-list' and 'Weights-list' allows for the user to specify control, or design, points for a member. The values listed in 'Points-list' refer to the corresponding nodes in *coordinates.txt*, and control the sweep of a member. The sweep follows a NURBS curve from these nodes. The values under 'Weights-list' specify the weighting of each point. If the weight is equal to 1, the curve is simply a B-spline.

Note that each member of the links is specified in Table 4.3. This does not always have to be the case, as the system provides automatic generation of link geometry. By introducing a 'default' value instead of a link and/or a member number, the system can use these values for all the corresponding geometries, as seen in Table 4.4. The first line in this example means that all members on link 0 will have (0.2 0.1) as dimensions, while the second line means that all members on all remaining links will have (0.1 0.05) as dimensions.

Table 4.4: Example shapes input file with default

| Name          | Link    | Member  | Cross-section | Dimensions | Points-list | Weights-list |
|---------------|---------|---------|---------------|------------|-------------|--------------|
| "Knuckle"     | 0       | default | "rectangular" | (0.2 0.1)  |             |              |
| "Other-links" | default | default | "rectangular" | (0.1 0.05) |             |              |

#### 4.1.4 Springs and Dampers

A mechanism might include springs or dampers. These are defined in the *spring-damper.txt* file. An example is illustrated in Table 4.5.

Table 4.5: Example spring-damper input file

| Type     | Point-start | Point-end | Incident-links | Stiffness/Damping |
|----------|-------------|-----------|----------------|-------------------|
| "Spring" | 7           | 8         | (2 1)          | 750000            |
| "Damper" | 7           | 8         | (2 nil)        | 3000              |

As we can see, springs and dampers are treated relatively equal. So, to differentiate between the two, the type has to be specified in the 'Type' column. To define the start and end points of the springs and dampers, the nodes in *coordinates.txt* are again used as references. The links that the spring or damper is connected to, also has to be defined, in the 'Incident-links' column. The reason for requiring the incident links is explained in Section 4.5.

#### 4.1.5 Loads

For analysis and simulation purposes, loads can also be applied to a mechanism. The loads are defined in *loads.txt*, as seen in Table 4.6.

Table 4.6: Example loads input file

| Type   | Point | Direction     | Magnitude | Loaded-link |
|--------|-------|---------------|-----------|-------------|
| Force  | 4     | (0.0 0.0 1.0) | 50        | 1           |
| Torque | 0     | (0.0 0.0 1.0) | scale30   | 0           |

There are two main types of loads that can be added, namely forces and torques. As springs and dampers, these are also added to a node point as well as defined on a link, to guide the system.

One can choose between two types of magnitudes, a constant and a scale. The constant will instantly apply a force of the given magnitude, whilst the scale is a simple linear function with the specified magnitude as the slope.

### 4.1.6 Design Optimization

In order to allow for automatic initiation of design optimization, a final *optimization.txt* is used. An example file is shown in Table 4.7.

Table 4.7: Example optimizations input file

| Optimization Type | Affected links | Init-values | Constraints | Max deformation |
|-------------------|----------------|-------------|-------------|-----------------|
| cross-section     | (0 1 2)        | (0.1 0.1)   | stress      | 0.038           |

'Optimization Type' indicates what the design optimization should be optimized with regard to. This example illustrates a minimization of cross sections. 'Affected links' indicates that link 0, 1 and 2 will have their cross sections minimized at the same time. 'Init-values' are the start width and height for the given cross sections. If needed, they can be changed in the GUI at a later stage. 'Constraints' refer to constraint functions, (see Section 2.6), and in this case, it indicates that the stress should not exceed the yield stress for the given link material. Finally, 'Max deformation' is an optional parameter, which represents a max deformational value (used in comparison with responses from FEDEM). For further elaboration see Section 2.6.

### 4.1.7 Mechanism Library

All the mechanisms created is stored in individual folders under library, each with a copy of the input files presented in Section 4.1. Each mechanism folder has subsequent folders for versions. It is inside these version folders that the input files are stored. This construction of the mechanism library allows users to switch between, and edit, different mechanisms and versions without having to recompile the system.

## 4.2 Initial Frame Placement

Section 2.3.5 explains the main concepts behind the SU kinematic modeling convention. With the use of an augmented two-frame method the joint and link displacements are decomposed into three axial screw displacements. The augmented two-frame method states that every link-joint pair should have exactly two frames, note that this refers to a single joint element (male or female). This is done in order to correct for

potential spatial displacements between the joint centers. Figure 4.1 illustrates how the 'ordinary' and 'augmented' frames, denoted as main- and sub-frame respectively, are placed for the two elements of a revolute joint. The sub- and main-frame of a link corresponds to  $F_{D_j}$  and  $F_{C_j}$ , or  $F_{A_j}$  and  $F_{B_j}$  (as seen in Figure 2.9) respectively, depending on whether it is the start or the end of a link.

In the pilot implementation, certain joint orientations could cause the incident members to get undesirable twists and bends. Twists occurred for instance for some members between a revolute and ball joint. This was corrected by adding a condition for linear independence between joint directions. Bends occurred for instance for members between a free and revolute joint, due to their relative spatial displacement. This was corrected by augmenting the sub-frame for a free-joint relative to interconnected joints.

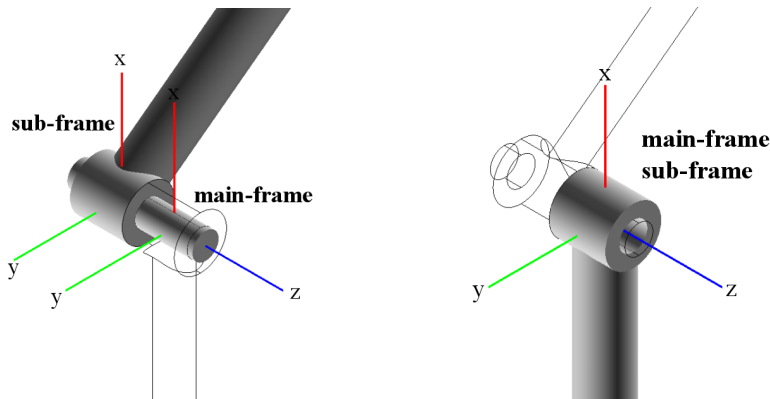


Figure 4.1: Frame placements for the male and female elements of a revolute joint

### 4.3 Links

Recall from the Section on Application Input, 4.1, that the start and end point of a link member is defined, implicitly, by a joint's link-incidence. This means that the position of links is governed by the joints' positions. As described in the Section 2.3.1, links can be defined as binary, ternary or quaternary (or more) links. The degree of a link governs the number of possible connections (members) that can be made. For a binary link,

there is only one member between the two joints. For a quaternary link, there is six possible members. Thus, the number of members possible for a link is given by  $\frac{n}{2}(n-1)$ , where  $n$  denotes the number of incident joints. By default, all possible members of a link are drawn. However, if the cross section type is set to 'nil' (as seen in 4.3), the member is neglected.

Members of a link are generated by sweeping a given cross section from one joint to another. The sweep follows a spline curve from one sub-frame to another, thus the start and end points are given by  $F_{D_j}$  and  $F_{A_j}$ . The orientation of the frames follows the directions specified in the SU convention: z-axis as the joint direction, x-axis as the member-direction (direction towards the next joint) and y-axis as the redundant axis. Splines are generated by interpolating control points to obtain a smooth continuous function. The control points used for interpolation is either obtained from the input files or automatically generated. This, combined with the ability to use default values for joints and links, allows for a great degree of flexibility. Figure 4.2 illustrates a smooth NURBS curve that has been automatically generated between two joints.

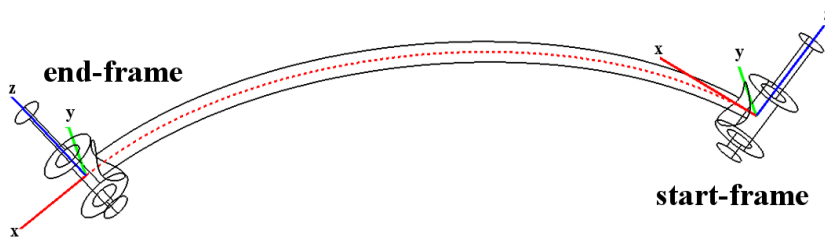


Figure 4.2: Smooth NURBS curve (illustrated with red dots) generated automatically from joint-directions

The members are rigidly fastened and can not move relative to one another. This allows a solid to be created in-between them, called a *link surface*. Examples of these surfaces are shown in red in Figure 4.3. Each surface is constructed in the following way: First, a sheet is made from three curves and then the surface is thickened to a size dependent on the surrounding link members. In the case of quaternary links, there are actually two (triangular) surfaces made, that are fused together to produce the total surface, as seen in Figure 4.3b. The number of possible surfaces are given by  $\frac{n}{6}(n-2)(n-1)$ , where

$n$  is the number of incident joints.

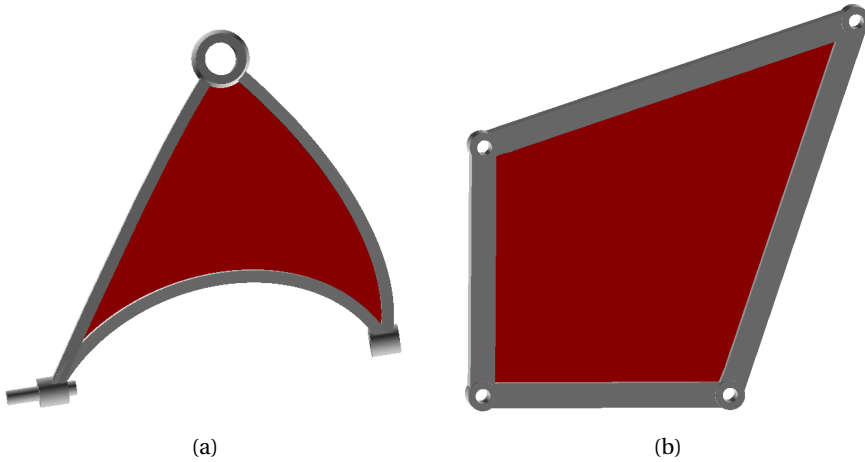


Figure 4.3: Ternary and quaternary links with surface

## 4.4 Joints

As explained in Section 2.3.2, a joint consists of a male and female element. Each element has a main- and sub-frame, oriented with respect to the SU convention. Since the two main-frames of a joint coincides, this position is used as the joint center. The sub-frames do not coincide since they are positioned according to the incident links. A joint is defined as a connection between two links. In the pilot implementation both these links had to be an actual link. When trying to define a joint to be connected to ground (a necessity for analysis) this definition yielded several errors and required extensive workarounds. By enabling a joint to be connected to 'nil' links, as described in Section 4.1.2, we managed to maintain the kinematic model whilst integrating grounded constraints for analysis.

The geometry of the joints are designed in order to minimize the stress concentration present under load. By blending sharp edges, a smoother path for the force to flow is obtained. Blends, or rather fillets, are created in the intersection between a joint element and incident members. The geometry of joints and fillets are dimensioned with respect to the incident members. Typical ratios, as discussed in Section 2.3.2, calls

for the scale-factors  $D/d$  and  $r/d$  to be between 1.2 and 1.5, and 0.02 and 0.06. As a trade-off between the weight and a minimized SCF, 1.2 and 0.02 is used as the initial scale-factors.

More precise scale-factors could have been obtained by calculating the SCF with regard to the nominal and max stresses present. However, since this calls for separate formulas for each joint type as well as the consideration of dimensional changes and discontinuities, it is better fit to let the user adjust the scale-factors further.

The process of automatically determining which edges in the geometry that should have a fillet is rather cumbersome. A custom procedure was developed which searches for all intersecting edges between incident members and the 'joint geometry' itself. If any edges are found, the method will apply a fillet with the given scale factor of 0.02. Note that some geometries might have incompatible edges with this initial scale-factor. In these cases, choosing a smaller scale factor for the specific edges usually solves the issue.

## 4.5 Springs and Dampers

In order to control dynamical responses in the mechanism, springs and dampers may be added. Even though both springs and dampers can be nonlinear, only linear versions are considered in the mechanism system. Thus, only basic parameters, such as the spring and damping coefficients, have been implemented.

The input files (Section 4.1.4) determines which links the springs and dampers are connected to. This could have been derived from the individual node positions, however, we have chosen to make this information mandatory. The reason behind requiring the incident links is twofold: (i) computational efficiency, and (ii) uniform definition of how parts are connected to ground. When a spring or damper is connected to a part, an external node is connected to an internal mesh node at its position. Obviously, the mechanism needs to be meshed before the internal node can be found, so specifying the connecting link(s) prohibits *all* links in the mechanism to be unnecessarily



meshed, when retrieving the node. In addition, the coordinates of the user specified external node connection is not guaranteed to exactly correspond to those of an internal node. Therefore, a search algorithm is used to find the closest internal node, which then is used for the connection. This node can either be a regular mesh node or an RBE2 independent node, if this is the closest one.

## 4.6 Loads

Applying external loads is a central part of any static or dynamic simulation. The mechanism system allows two types of loads to be added, namely forces and torques. These loads are defined by a separate input file, as described in Section 4.1.5. Each load type corresponds to an external node, thus they are point loads. As with springs and dampers, loads must be defined on a link, for the same reasons explained Section 4.5.

## 4.7 Mechanism Assembly

Every joint element, link member and surface geometry is initially constructed as separate entities. At this stage they do not conform with the kinematic model since parts that are supposed to be rigidly fastened together are not yet 'united'. To ensure correct kinematic properties, the mechanism is assembled in the following way: First, each member of a link is sewn together as they should not move relative to one another. Second, for ternary and quaternary links a surface is also generated (this is optional and can be removed by the user). Lastly, the joint geometry is sewn together with the link geometry, since a joint is not considered a separate physical entity by itself (Section 2.3.2). The end result is one link consisting of joints, members and a surface.

## 4.8 Meshing

When the final geometry for the mechanism is ready, it needs to be meshed in order to be prepared for analysis. An important aspect of meshing is setting the mesh size. In AML this is done through a feature called *attribute tagging*, which mainly is used to set a specific mesh size on an object in the system. It can also be used to 'tag' an object, so

it can be queried to retrieve specific 'mesh parts' from a larger geometry later. The specific mesh parts include nodes, edges, surfaces and solids. An example usage is to tag all one-dimensional entities, so that all nodes in a given mesh object can be retrieved. In the mechanism system, each separate part of a link is tagged with a mesh size. These parts are members, joint elements, surfaces and blends, and they are applied unique mesh sizes primarily to match their individual dimensions. For a member, the size is set to a quarter of its shortest cross section dimension, for joints, it is an eighth, and for surfaces the size is equal to the surface thickness. Blends are always located in typical high-stress areas, so their relative mesh sizes are set to half the size of the joint's size. Due to these different mesh sizes, the transition between them can be rather poor if no refinement methods are applied. Poor transitions typically involves elements with bad aspect ratios, like described in Section 2.4.2. Luckily, the AML mesher provides methods for h-refinement, where an isotropic refinement is utilized. This refinement gradually reduces the size of the larger elements, so that they transition smoothly towards the smaller elements. In addition, a curvature refinement, i.e. making a finer mesh where there is curvature, is applied to all links. This means that, for example a link with a circular cross section, will get a smaller mesh size than it initially had, due to its curvature.

Each link geometry is meshed separately, initially with four-noded triangular surface elements. The element type can be changed manually in the GUI later, but changing the number of nodes for the elements is not possible. The triangular elements are three-noded, and the quadrilateral are four-noded. The reason for creating an initial surface mesh is because the mesher requires an existing surface mesh in order to generate a volume mesh. Furthermore, the volume mesher can only generate a tetrahedron mesh. Therefore, it *has* to use a triangular mesh as a basis. However, it is possible to change the surface element type to either a quadrilateral or a hybrid mesh, but no volume mesh can be made from these types. All mesh elements and entities for a link are stored in a mesh database, which can be stored and retrieved.

### 4.8.1 Boundary Conditions

In order to simulate boundary conditions between links, RBE2 connections are made for each joint element. The node used for the independent node (see Section 2.4.3) will always be located in the joint's center (position of the main-frame). Since this node does not exist in the mesh, it has to be explicitly created and added to the node list. Note that the independent node for both the male and the female element will be located at the same coordinates. The forces acting between the links will then be transferred through the two independent nodes.

Unlike the independent node, the dependent nodes are not explicitly created. These are retrieved from the mesh via the tagging feature. In general, for male elements, the dependent nodes will lie on the surface, around the joint center. The nodes for the female elements will in general lie on the inside of the element, also around the joint center. All dependent nodes have 6 DOFs by default. This is because they should all move equally, relative to the independent node, and not relative to each other.

An RBE2 are also created for all free joints. This is beneficial for three reasons: *(i)* The RBE2 can be used as a connection for springs, dampers and loads, to distribute forces over multiple nodes. *(ii)* A free joint can be fixed in one or more directions, thus it needs an MPC to avoid singularities. For example defining a free joint to simulate bearings. *(iii)* The RBE2 is placed at the end of the cross section a given member, and can be used as a basis for other joint types, such as the cylindrical and prismatic joints. The current implementation satisfies the minimum requirements to create both a cylindrical and prismatic joint in FEDEM.

## 4.9 Analysis

AML comes with an analysis module that requires the integration of MSC Nastran. The module is complemented by a single example for static analyses that is not working, nor further documented. The only part working is writing Nastran Bulk Data Files, (*.bdf*), which contains all FE nodes and elements. In the absence of a working AML analysis module, as well as the need of dynamic analyses, FEDEM is used as the analysis module

instead. As there are no existing frameworks or interfaces to aid in the integration between AML classes and FEDEM models, a set of custom functions have been developed to facilitate this integration, and finalize the pre-processing before analysis. In general, the functions convert the internal structures in the mechanism system to a format understood by FEDEM. Recall from Section 3.4 that mechanisms in FEDEM are represented on a substructure and system level. The substructures represent the actual FE parts, which is what the *.bdf* files are used for. A single *.bdf* is created for each link in the mechanism and assembled at the system level. The mesh written to *.bdf* consist of either shell or solid elements, which is specified by the user in the GUI. The system level is represented by a Fedem Mechanism Model (*.fmm*) file, containing a complete description of the mechanism. The *.fmm* file is automatically generated by the mechanism system, and connects all the corresponding substructures, joints, springs, dampers and loads. This is done by looping through every entity in the mechanism model, and writing the corresponding FEDEM definitions to *.fmm*. Transformation and rotation matrices (presented in Section 2.3.4) are used to ensure that the kinematic model remains intact, e.g. correct orientation of revolute joints. An example model file generated by the system can be seen in Appendix D. In addition, generic functions for adding a control system are created. However, due to the number of input parameters possible, only an example to control the crank of a four-bar mechanism can be used.

The automatic generation of the *.fmm* file enables analyses to be run without any need of input or modification in FEDEM. This means that directly after the mechanism input files are created, the mechanism can be automatically exported and analysed, replacing the otherwise time-consuming process of manual pre-processing. The results from analysis can then be used to alter the design in any way, before running further analyses.

## 4.10 Results

The following results summarize our improvements and additions to the mechanism system. All depicted geometries are created in the mechanism system and retrieved from either the system itself or FEDEM. To visualize analyses in FEDEM, angular or

translational deformation are used for face contours.

Figure 4.4 illustrates issues resolved in the kinematic model by comparing the old and new system for identical configurations. Certain joint orientations could cause the incident members to get undesirable twist and blends, and in some cases leaving the whole member geometry undrawable from its connection line.

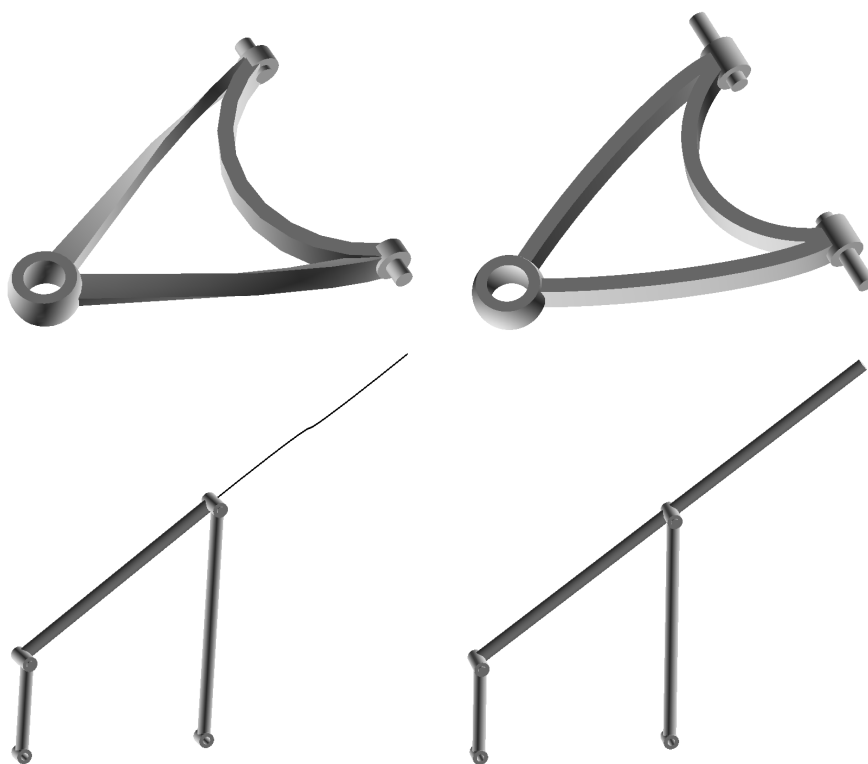


Figure 4.4: Fixed link twist and spatial joint displacement

Figure 4.5 shows unnecessary links and null geometries (with a black line for their connection) in the old system as a result of an incomplete joint definition. The revised joint definition allows joints to be connected to 'ground links' with no geometry, avoiding both subsequent unwanted links and null geometries.

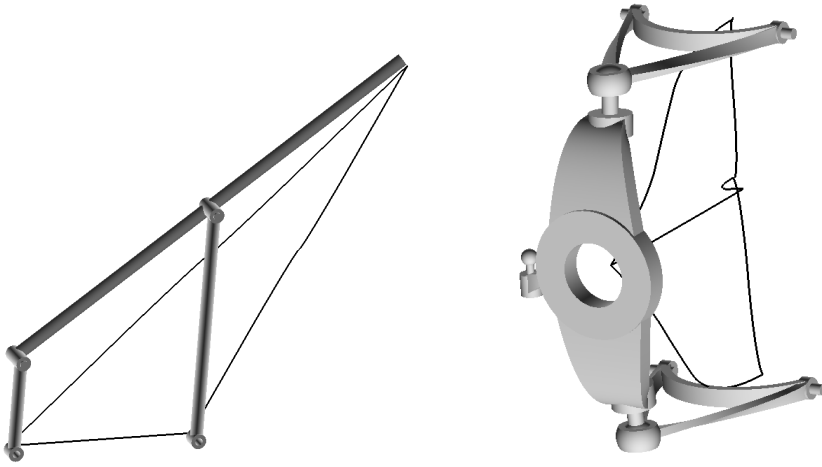


Figure 4.5: Unnecessary links and null geometries in the old system

An additional joint type has also been implemented. As seen in Figure 4.6, the single knuckle joint is a variation of the revolute joint where the point of rotation (joint center) is not displaced to the side. Alternatively, one can use the revised joint definition to introduce a double revolute joint. This is done by defining a joint, with its incident links, twice in the input files. For instance, if two revolute joints are defined for the same point and with the same link as the male element, but with different joint directions, the system will generate two pins (one for each other incident link). This configuration is used for the log grabber mechanism, Figure 4.21.

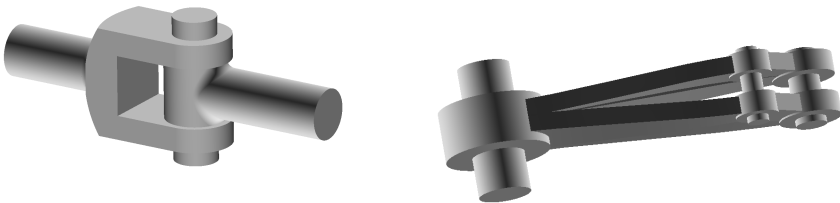


Figure 4.6: Single knuckle and double revolute joints

Figure 4.7 shows two examples of how several fillets are automatically applied to the transition of a member and joint geometry, for a revolute joint and ball joint. This process happens at the link assembly, and works for all types of cross sections and joints, as long as the geometry does not contain too small edges, relative to the fillet radius.

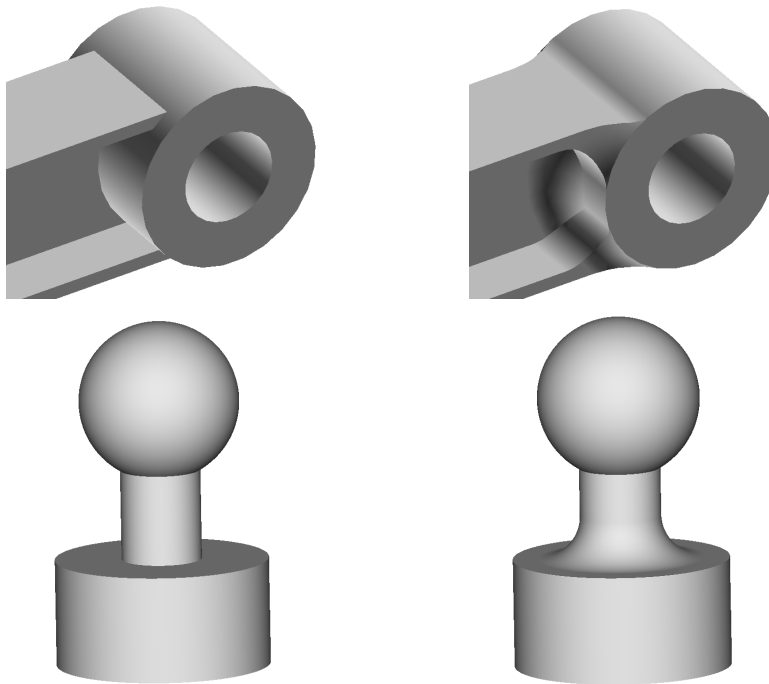


Figure 4.7: Revolute and ball joint before and after fillet

For each free joint, RBE2 connections are made at the end of the member cross section. The independent node is located at the center of the cross section, while the dependent nodes are found by gathering all internal mesh nodes at the end cross sections, illustrated in Figure 4.8. Free joints have also been extended to allow different fixed DOFs. This enables, for instance, the user to make a rigid joint, create 'custom' joints or simulate a ball/revolute joint without creating the subsequent geometry.

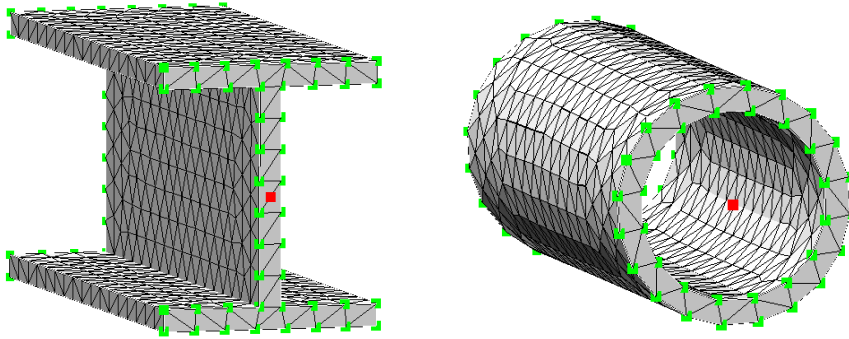


Figure 4.8: I-beam and thin-walled tube with RBE2 connections at their ends

Figure 4.9 shows how a surface can be created between curved members of a link. The surface is thickened and sewn together with the members and joints in the mechanism assembly process, creating one single component.

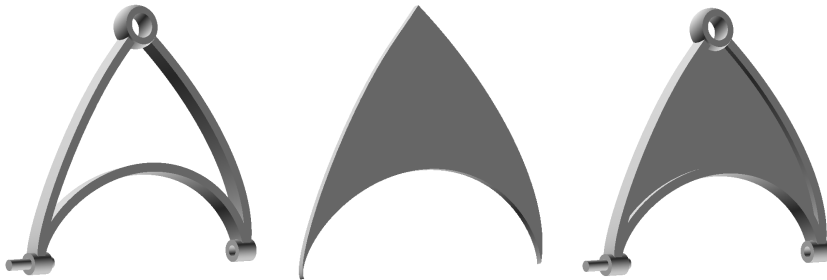


Figure 4.9: Thickened surface of a lower wishbone

Figure 4.11 show how springs and dampers are represented in the system. Their end points are specified in the input files, and will connect to the closest internal mesh node, shown in green.

Loads are applied in the same way as springs and dampers. For the case in Figure 4.11, both the force and torque will be attached to the independent RBE2 node of the female revolute joint. This torque configuration is used for the crank in Figure 4.15a.



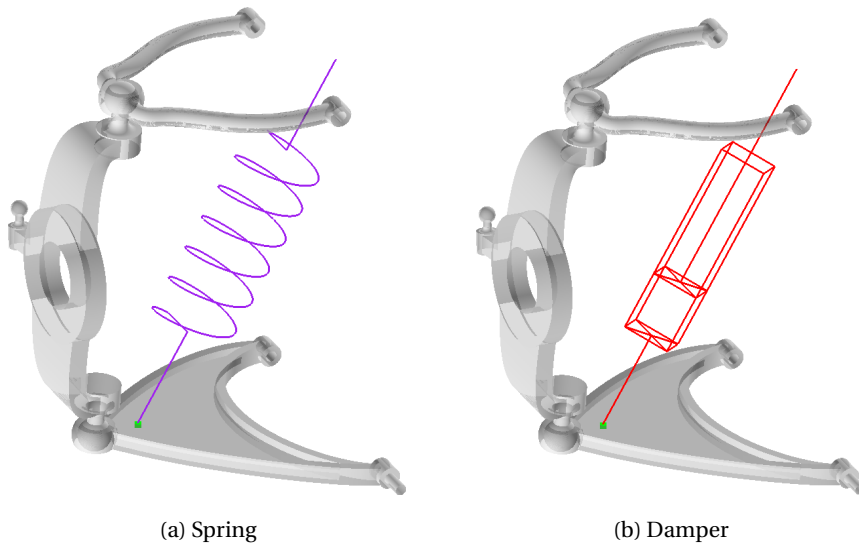


Figure 4.10: Spring and damper for a double wishbone suspension

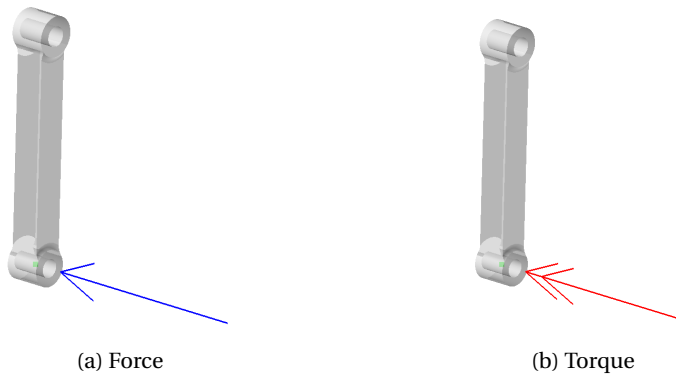


Figure 4.11: Force and torque on a simple I beam

Figure 4.12 depicts the different stages of the mesh refinement process. Initially, there is no refinement, and the transition between the member and the joint is non-filleted and contains elements with poor aspect ratios. Then, a fillet with subsequent small elements is applied, however this yields even more elements with poor aspect ratios. Finally, a smooth transition is shown in Figure 4.12c and Figure 4.12d, both with isotropic mesh refinement and curvature refinement applied. Smooth transitions and refine-

ment is also possible for volume mesh. Figure 4.13 shows a refined tetrahedron mesh.

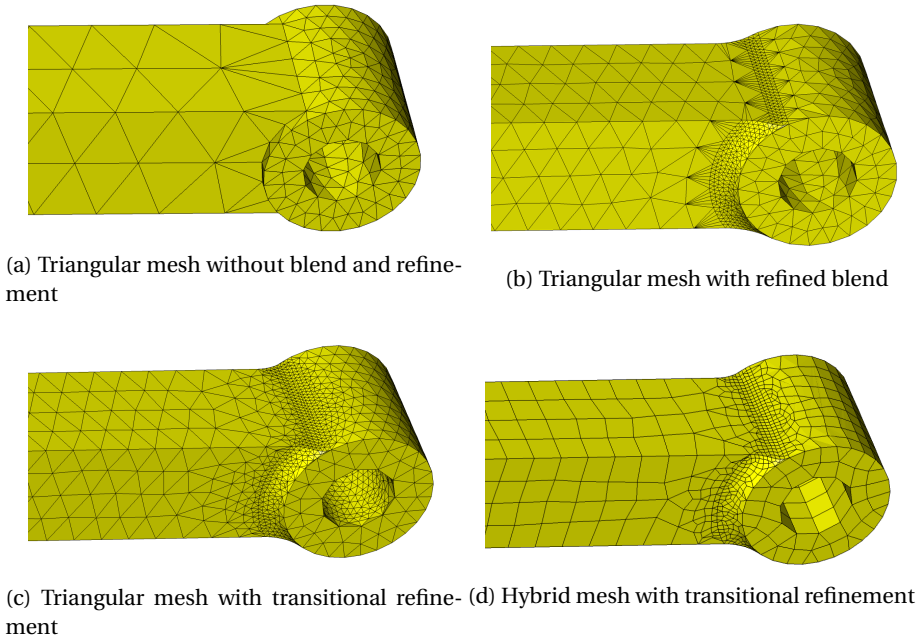


Figure 4.12: Different mesh configurations with isotropic curvature refinement

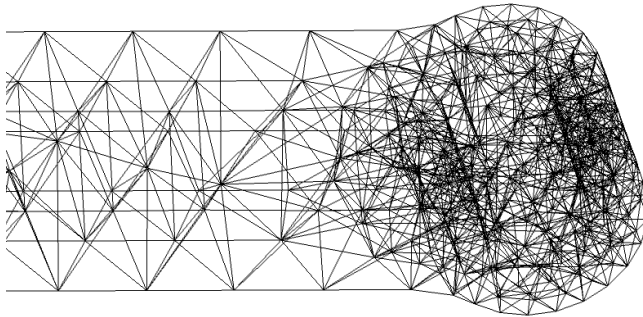


Figure 4.13: Refined tetrahedron mesh

Figure 4.14 shows how joints, loads, springs and dampers are automatically positioned and attached to RBE2 elements in the corresponding substructures in FEDEM. The subsequent model file can be seen in Appendix D.

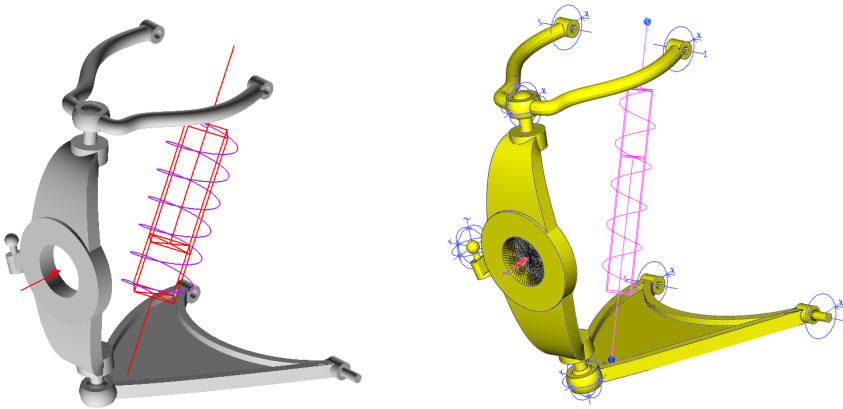


Figure 4.14: Double wishbone suspension represented in the mechanism system (left) and FEDEM (right)

Control systems can also be added from the mechanism system and automatically attached to the correct substructures in FEDEM. Figure 4.15 shows the first positions of a four-bar, 'straight line', mechanism, subsequent control system and resulting path of the coupler end.

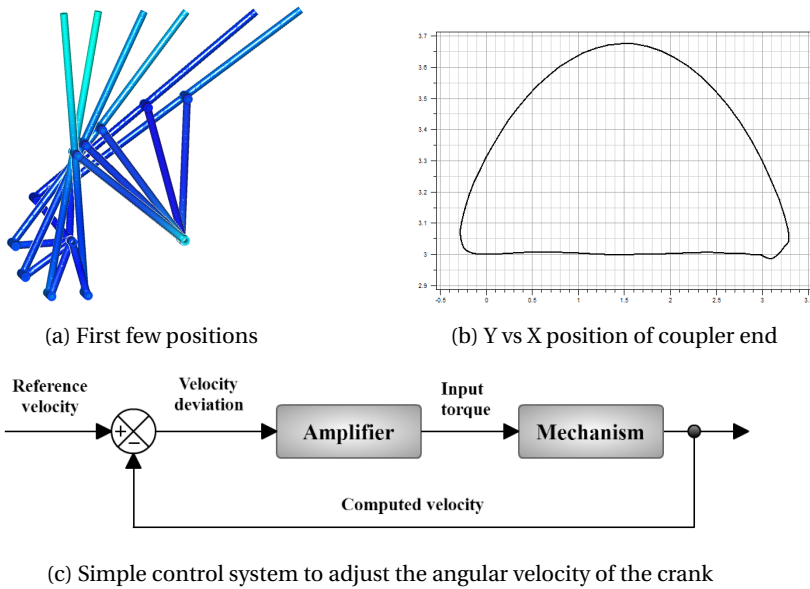
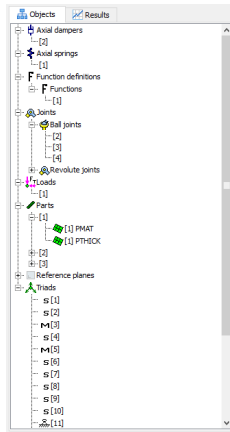
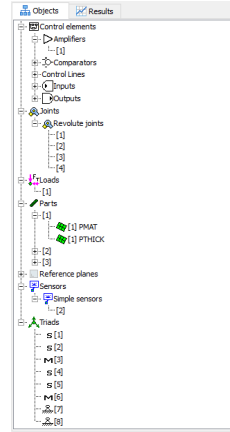


Figure 4.15: Four-bar mechanism with control system

Figure 4.16 shows the resulting object trees that is automatically generated in the export between the mechanism system and FEDEM.



(a) Double wishbone suspension (Figure 4.14)



(b) four-bar (Figure 4.15)

Figure 4.16: Resulting object tree in FEDEM

When running analyses in FEDEM, necessary design changes may come apparent, e.g. singularities in the spring-damper connection of a lower wishbone (as seen in Figure 4.14). The revised joint definition can also be used to create custom spring-damper connections, with subsequent RBEs. Figure 4.17 shows how an example of how this can be done. Here, a female revolute joint is created by defining additional members in the lower wishbone. This configuration is not fully automated yet and requires slight post-editing of the model in FEDEM (removal of redundant joints and RBEs between the members).

Figure 4.18 shows how the RBE2 connection for a ball joint is represented in the mechanism system, and then in FEDEM. The independent nodes are shown in red, and the dependent nodes in green.

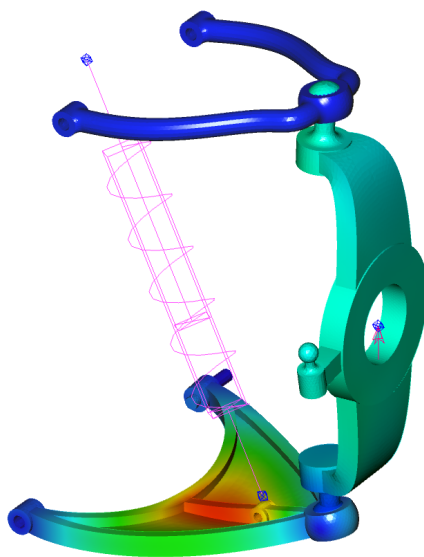
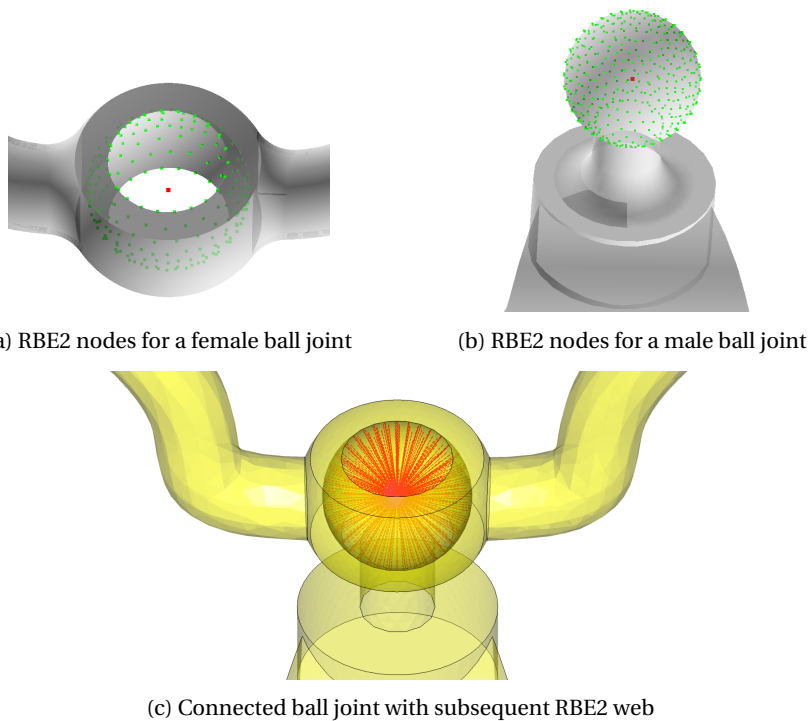


Figure 4.17: Double wishbone translational deformation



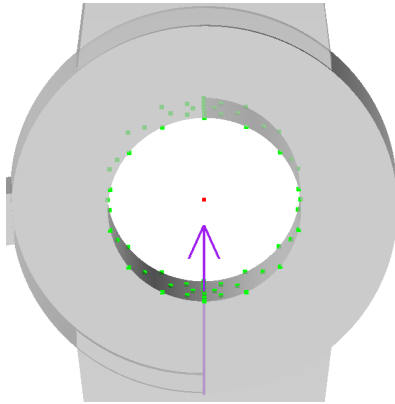
(a) RBE2 nodes for a female ball joint

(b) RBE2 nodes for a male ball joint

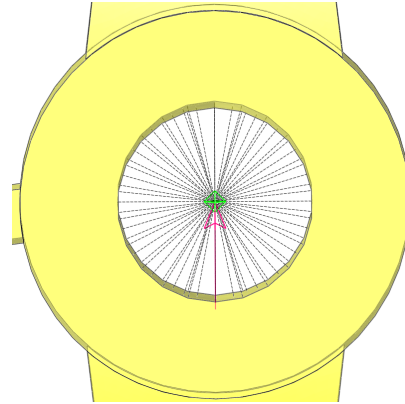
(c) Connected ball joint with subsequent RBE2 web

Figure 4.18: RBE2 for a ball joint in the mechanism system and FEDEM

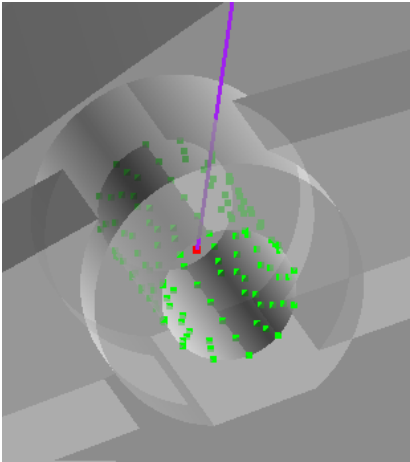
Similarly, loads, springs, and dampers are connected to independent nodes in the mechanism system and subsequent triads are created and connected in FEDEM. Figure 4.19 depicts how loads, springs and dampers are connected to the mechanism in the system and FEDEM.



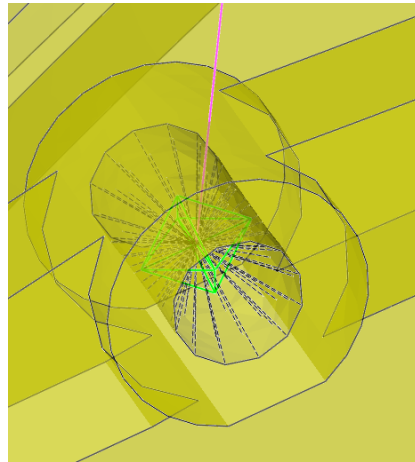
(a) Force displayed in the mechanism system



(b) Force displayed in FEDEM



(c) Spring and damper connection displayed in the mechanism system



(d) Spring and damper connection displayed in FEDEM

Figure 4.19: Load, spring and damper connection in the mechanism system and FEDEM

Figure 4.20 and 4.21 shows two additional mechanisms that has been added to the mechanism library, a steerig linkage and a log grabber. The steering mechanism is a rather radical design, consisting of 14 links. The mechanism use a six-bar chain to guide each wheel, resulting in a large translational shift. Note that both of these mechanisms require simple modifications in FEDEM before analysis can be run.

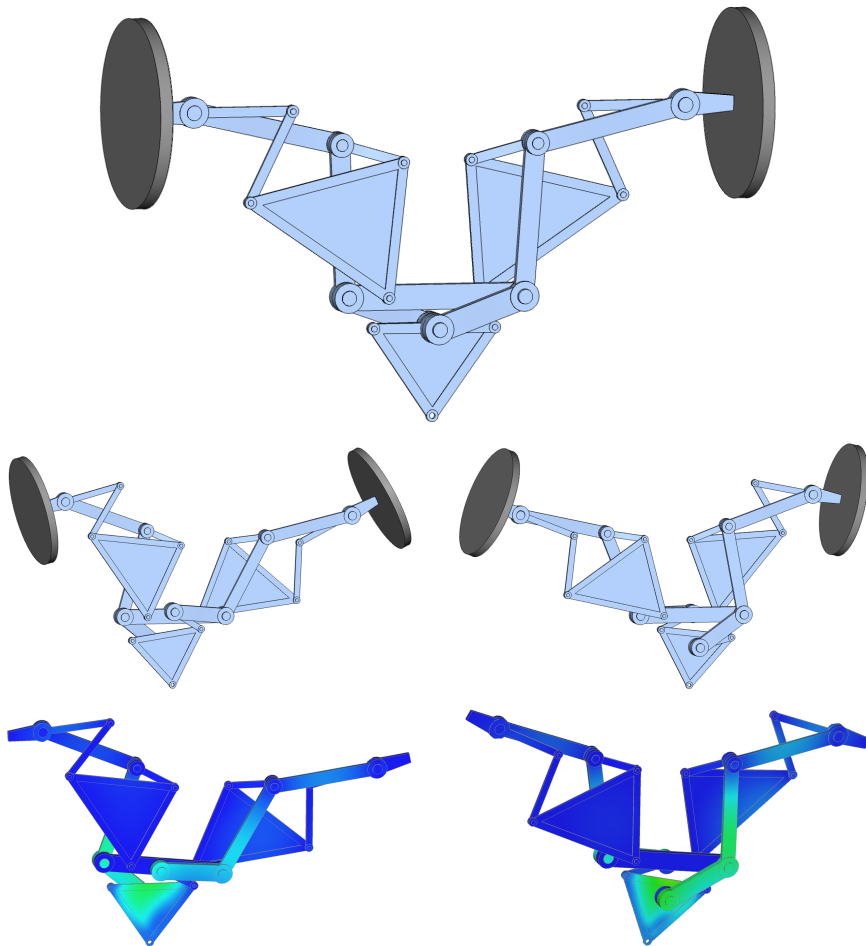


Figure 4.20: 14-bar steering linkage

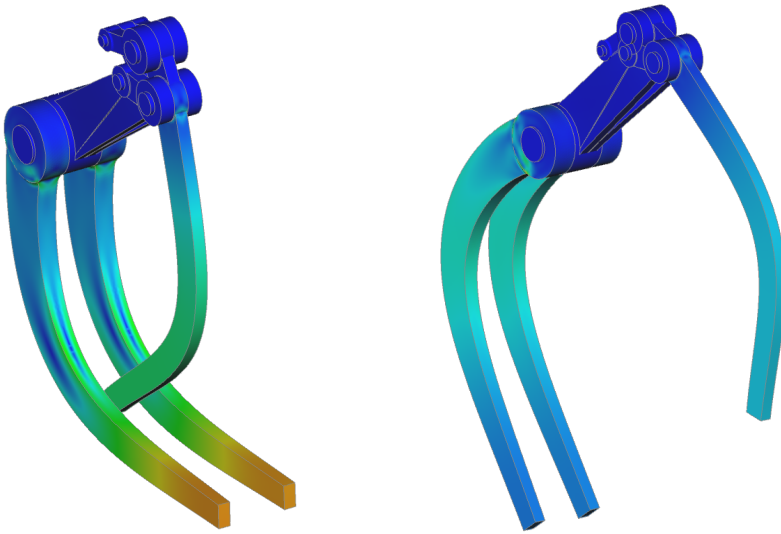


Figure 4.21: Log grabber mechanism

The revised definition of free joints and subsequent RBEs supply the basics needed to set up prismatic and cylindrical joints in FEDEM. This process is not yet automated, however with simple modifications in FEDEM both of these joints can be applied, as seen in Figure 4.22. Note that these would ideally have more than two masters (in future implementation).

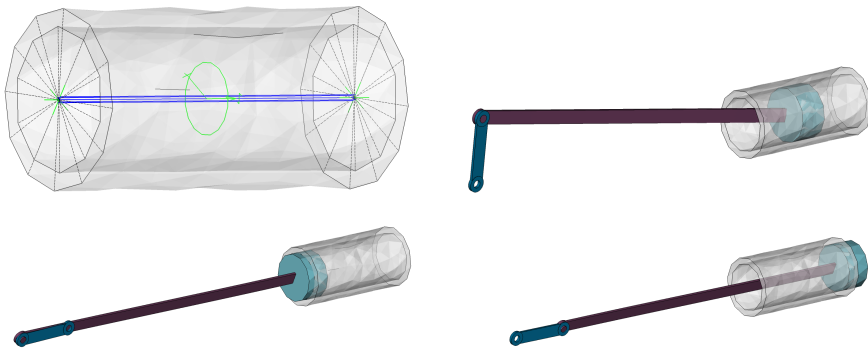


Figure 4.22: Cylindrical joint used for a slider crank mechanism



## 4.11 Discussion

The kinematic model and general link-joint synthesis in the pilot implementation has proved to be a good basis to create generic mechanisms. The 'old' system synthesis has remained intact, with the exception of the underlying joint definition and some frame augmentations in the kinematic model. The present kinematic model is stable for simple mechanisms, but a few inconsistencies can still be found. For instance, if no additional design points for a member is given, the SU parameters might not produce satisfactory NURBS, due to sharp angles between the member and incident joints. However, this is easily resolved by defining extra control points for the NURBS.

The revised joint definition has enabled a more versatile modeling process of mechanisms. By allowing the user to define the individual DOFs for a free joint, a wide variety of geometric constraints can be obtained. Figure 4.22 shows how free joints, with their RBE2s, can be used to create a slider-crank mechanism. Free joints can also be used to replace a ball or revolute joint, if their joint geometries are undesirable.

Links are created by assembling separate elements to form one body. This individual construction of geometries allows both tags and geometric primitives to be queried after the actual link has been created, e.g. creating fillets for intersecting edges. Individual mesh sizing enables the mesh to be refined in areas of high stress concentration. Both blends and joints are initially defined with quite small mesh sizes. As an additional curvature refinement is applied, elements in high-curvature areas (e.g. blends) become a lot smaller than their 'tagged' sizes. Recall from the theory on mesh sizes (Section 2.4.2) that calculating the best possible mesh size is an impossible task, and will also depend on the accuracy needed for the analysis results, while keeping the computational efficiency as high as possible. If the elements become too small, the designer could easily turn the refinement off or increase the size in the GUI.

Since the volume mesh uses a surface mesh as an input parameter, it will receive the same refinements applied to the surface mesh. As stated, the volume mesh consists solely of four-noded tetrahedron elements, which generally are regarded as poor elements, due to their stiffnesses. As a result of insufficient documentation, it is not clear

if other solid element types are available.

For all male and female joint elements, the RBE2 connection will choose all mesh nodes on the connecting surface area as dependent nodes. Since there is no relative motion between the dependent nodes, any stresses that would occur in the surface geometry, are therefore neglected, due to the stiffness. This feature is often desirable, if the stresses in the joints themselves are of no interest. Should they be of interest, a more realistic connection can be modeled by only using the nodes closest to the independent node as dependent nodes. An even better solution is to use an RBE3 connection, but this often requires a user input, and is difficult to implement universally.

Sections 4.5 and 4.6 mentioned that the coordinates of the connection points for springs, dampers and loads are not guaranteed to exactly correspond to those of an internal mesh node, so they simply connect to the closest node, which might be an RBE2 independent node. Of course, this solution is not optimal, as the connection should be on an MPC, to distribute the loads evenly. A possible solution is shown in Figure 4.17, where an extra revolute joint and two members are created on the link. Since the system automatically creates an RBE2 constraint in the joint, this can become the connection for the spring. If this becomes too complicated, or the extra geometry is redundant, the RBE2 can of course be created in FEDEM and applied as the spring connection. A final solution is to have the system automatically create an RBE2 if the closest node is an internal mesh node, but this feature is yet to be implemented.

With the existing load, spring and damper configuration, realistic results from analyses are obtainable. However, in order to simulate a wider range of mechanisms, more complex loads and spring/damper configurations should be included. The pre-processing could for instance be extended to include non-linear stiffness and damping functions, as well as time history input files.

The mechanism system has become a generic pre-processor, and due to the lack of an integrated working analysis module, the pre-processing is tailored for FEDEM. Automatically creating *.fmm* files has been one of the more difficult tasks in the implementation, largely due to retrieving the correct node numbers, joint handling (transformations), and configuring triad connections. The conversion from the AML object

format, to the FEDEM *.fmm* format requires an unambiguous definition of each part of the system, and a correct parametrization of mechanism models. The parametrization enables the user to make both topological and morphological changes, automatically generate a mesh and run subsequent analyses. More specifically, the user can move design points without having to manually generate new geometry, and quickly generate meshes while not having to make changes to the underlying FEA model. Accompanied, these features define a mechanism that is immediately ready for analysis, allowing fast analysis iterations, where the results can be used to guide the design right from the start. Additionally, the embedded library with predefined 'basic' mechanisms allows users to modify and add mechanisms together to quickly solve new design problems. However, no implementation of post-processing features, i.e. handling of the analysis results, is conducted, with the exception of a single example used in design optimization (Chapter 5). Post-processing functionality would make the system more well-rounded and be able to facilitate a design process where less time is spent on routine design tasks, and more time spent on innovative and creative tasks.



## Chapter 5

# Design Optimization

Chapter 4 described how the mechanism system established an interface with FEDEM, by automatically converting the mechanism structure in AML to match the FEDEM mechanism models. The resulting *.fmm* file made it possible to immediately run FEDEM analyses manually, for obtaining post-processing data. This chapter will show how analyses can be ran automatically, and integrated as a part of a design optimization.

Section 5.1 describes the ways AMOpt impacts the general design optimization, and subsequent decisions. Section 5.2 will go through how the iteration process is affected by using AMOpt, and how workarounds are facilitated. The workarounds dictate how the problem formulation is constructed and implemented, and is described in Section 5.3. Based on these sections, we will go through a specific design optimization case in Section 5.4, in order to illustrate what the system is capable of. Finally, Section 5.5 will evaluate the results from the case, and discuss advantages and disadvantages of the way design optimization is implemented.

## 5.1 AMOpt

Recall from Section 3.3.5 that all AMOpt methods run an automatic, non-interruptible process before concluding with an optimal design. The design variables, objective functions and constraint functions have to be programmed as properties in the system model, before they can be assigned to their roles in AMOpt. This means that they have to return real numbers, or AMOpt will fail. There is also no direct way of incorporating response vectors in the formulation. In addition, MOGA is the only method in AMOpt where a constraint function can be added. A structural optimization without a constraint function is highly unrealistic, and can even be worthless, so MOGA has to be chosen, even though it has some drawbacks.

## 5.2 The Iteration Process

Since the AMOpt methods controls the whole optimization process, it could seem like there is no way to run FEDEM analyses between the iterations. However, the AMOpt implementation is also done in AML, which can be 'exploited'. AMOpt is an iterative process which first uses a subroutine for calculating the next set design variables, based on the constraint function (if any), and then calculates the corresponding objective function. Since all of these parameters are controlled properties, they can be modified. As stated, they have to return real numbers, but just as return values for the expressions, which means that a number of operations can be executed in-between. One of these operations can for instance mesh the current mechanism, or call a FEDEM batch command to calculate stresses or deformations. The return value of the function will then have to wait until all operations are executed.

This workaround allows analyses to be included at each iteration. In other words, FEDEM can be integrated to perform an analysis each time a new set of design variables are created. The only problem is that since MOGA has to be used, it does not make sense to run an analysis at each iteration. Remember from Section 2.6.4 that a general GA creates sequential populations, constituted by a number of design points that slightly deviates from each other by a random mutation factor. Since an iteration means

evaluating the next design point in the population, running an analysis on this design means running an analysis on a slightly different, *random*, design point. This is clearly a waste of computational effort, as many design points are guaranteed to be relatively equal. A better approach is therefore applied, where a FEDEM analysis is ran at the end of each generation, when all the design points have been evaluated. Each design point, with corresponding values for the constraint and objective functions, are then compared, and the design point yielding the lowest objective function value, while still satisfying the constraint function, is sent to analysis. The result from the analysis is stored, and can be used for a second constraint. For instance, the result might be a time history of deformations, while the second constraint is a prescribed maximum deformation. The final result of the optimization is the design point with the lowest objective function value, that also satisfies this second constraint.

### 5.3 Problem Formulation

The thoughts presented in the previous sections define how design optimization is implemented. For instance, since all properties have to be programmed, they must be tailored for a specific type of design optimization. The optimization type implemented is a cross sectional design optimization. In short, the goal is to minimize up to several links' cross section area,  $A$ , while the mechanism undergoes a bending moment,  $M$ . The maximum stress of the links,  $\sigma_{max}$ , must not exceed the yield stress for the given link material, which is the first constraint function,  $g_1$ . To include FEDEM analysis, we also include a second constraint,  $g_2$ , that relies on the analysis results. These results,  $\mathbf{s}$ , form a time history of the deformation of a given point in the mechanism. The largest deformation in the time history,  $u_{max}$ , should not be larger than a prescribed value,  $u_p$ , specified in the input files. The cross section type is also limited to I-beams in this pilot implementation. The design variables are then chosen to be the flange width,  $x_1$ , flange thickness,  $x_2$ , web height,  $x_3$ , and web thickness,  $x_4$ , shown in Figure 5.1. Note that several links can be affected by the design variables, stated in the input files. In addition, the two flanges will always have equal widths and thicknesses.

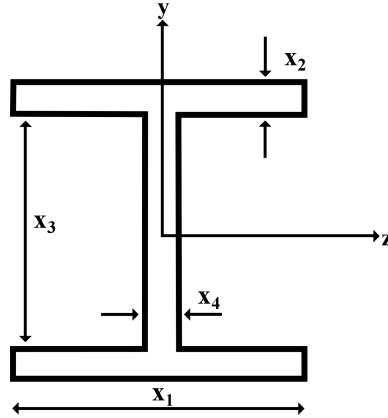


Figure 5.1: Design variables of an I-beam cross section

The problem formulation is as follows (based on the theory in Section 2.6): Given the design variables,  $\{x_1, x_2, x_3, x_4\} = \mathbf{x}$ :

$$\text{minimize} \quad f(\mathbf{x}) = A(\mathbf{x}) = 2x_1x_2 + x_3x_4 \quad (5.1)$$

$$\text{subject to} \quad g_1(\mathbf{x}) = \sigma_{max}(\mathbf{x}) - \sigma_y \leq 0 \quad (5.2)$$

$$\text{and} \quad g_2(\mathbf{s}) = u_{max}(\mathbf{s}) - u_p \leq 0 \quad (5.3)$$

$$\text{where} \quad \sigma_{max}(\mathbf{x}) = M \frac{y_{max}(\mathbf{x})}{I_z(\mathbf{x})}$$

The calculation of  $\sigma_{max}$  derives from simple beam theory, where  $M$  is the moment applied at the crank,  $I_z$  is the second moment of area about the  $z$ -axis, and  $y_{max}$  is the distance from the cross section's center of gravity to the point undergoing the most stress. Since the center of gravity always will be located at the mid of the cross section,  $y_{max}$  is the half of the total cross section height. Inserting expressions for  $I_z$  and  $y_{max}$ , as well as normalizing the total expression yields the following function:

$$\begin{aligned} g_1(\mathbf{x}) &= M \frac{x_2 + \frac{1}{2}x_3}{\frac{1}{6}x_1x_2^3 + \frac{1}{2}x_1x_2(x_3 + x_2)^2 + \frac{1}{12}x_4x_3^3} - \sigma_y \\ &= \frac{6M}{\sigma_y} \frac{2x_2 + x_3}{2x_1x_2^3 + 6x_1x_2x_3^2 + 12x_1x_2^2x_3 + 6x_1x_2^3 + x_4x_3^3} - 1 \leq 0 \end{aligned} \quad (5.4)$$



### 5.3.1 The Implementation

Figure 5.2 shows in detail how  $\mathbf{x}$ ,  $f(\mathbf{x})$ ,  $g_1(\mathbf{x})$ ,  $\mathbf{s}$  and  $g_2(\mathbf{s})$  are integrated in the iteration process. To begin with, the design from the input files,  $\mathbf{x}_0$ , is meshed and sent to FEDEM for analysis. The results are stored, and compared against the rest of the analysis results when the optimization process ends. Then, the AMOpt process kicks in, iteratively creating each design point in the initial population, and evaluating its objective and constraint functions. When the generation ends, i.e. when all design points have been evaluated, the best design point is meshed and analysed in FEDEM. The size of each population, as well as the number of generations are taken as user input in the AMOpt module. The process repeats itself until the result from the last generation is received, when the best result simply is chosen by comparison.

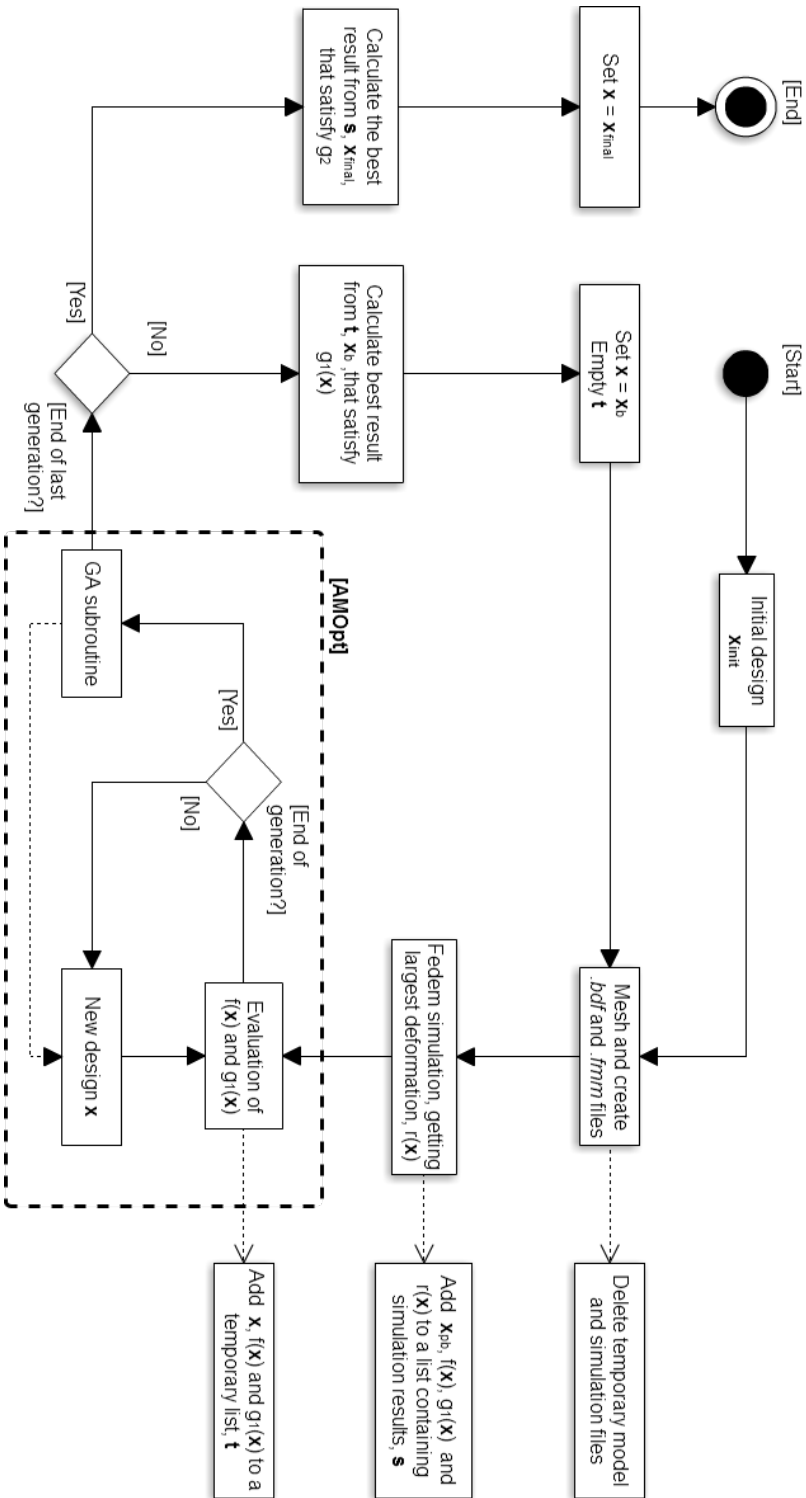


Figure 5.2: AMOpt loop

## 5.4 Results

The results of this pilot implementation are best illustrated through an example. The example is as follows: We wish to minimize all cross sections of a four-bar, while it undergoes a torque at the crank. This means that the design variables affect all the links simultaneously. Of course, a four-bar mechanism is supposed to rotate, but in this example, it is held in place, so that the simulation is static instead of dynamic. Dynamical movement of the mechanism is prohibited by stating that the end of the coupler link is a free joint, fixed in the global x- and z-axis, like Figure 5.3 shows. This example is chosen to illustrate the following key points:

- A mechanism can be automatically meshed after changing substructural parameters.
- Integration with FEDEM is successfully implemented, and can be automated as a part of a design optimization process.
- Design optimization works, and leads to a strictly better design than the preliminary, given the problem formulation.

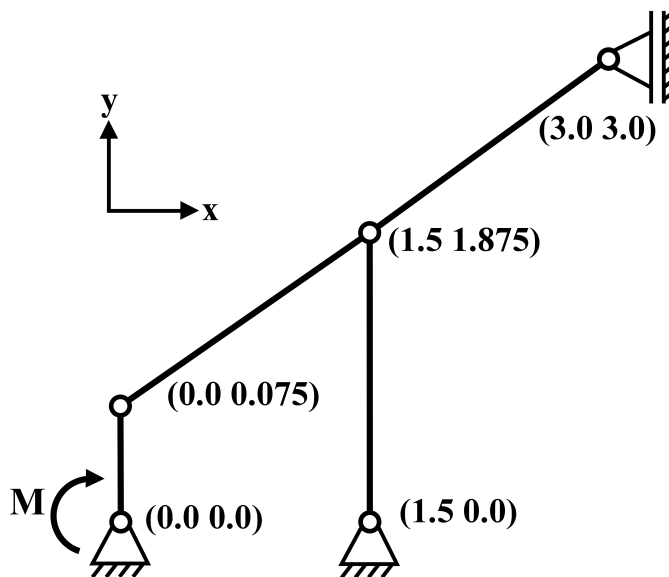


Figure 5.3: Initial design relative to the optimized design

The 'Init-values' from the optimization input file refer to the flange width and web height. Their thicknesses are automatically set by the system to be a tenth of their respective lengths, but can be changed in the GUI before optimization start, and are listed in Table 5.1. Note that all units are in meters, even though FEDEM is unitless. However, units are implicitly applied by the mechanism system, when calculating the various parameters. In addition, the mechanism is meshed with a three-noded surface triangular mesh, and have default mesh element sizes, as described in Section 4.8.

Table 5.1: Design variables with move limits

|                   | <b>Flange Width (<math>x_1</math>)</b> | <b>Flange Thickness (<math>x_2</math>)</b> | <b>Web Height (<math>x_3</math>)</b> | <b>Web Thickness (<math>x_4</math>)</b> |
|-------------------|--|--|--------------------------------------|---|
| <b>Init value</b> | 0.1                                    | 0.02                                       | 0.1                                  | 0.02                                    |
| <b>Min value</b>  | 0.05                                   | 0.005                                      | 0.05                                 | 0.005                                   |
| <b>Max value</b>  | 0.2                                    | 0.03                                       | 0.2                                  | 0.03                                    |

A link material and the magnitude of the torque must be chosen. AML provides a material catalog, where we choose steel for all links. The material properties<sup>1</sup> necessary for the analysis, are shown in Table 5.2. The torque magnitude is chosen to mimic an engine running at 30 Nm. Also note that the prescribed maximum translation,  $u_p$ , is a constraint for the point at the middle of the coupler link, at the revolute joint.

Table 5.2: Steel material properties and torque magnitude

| <b>E Modulus [GPa]</b> | <b>Yield Strength [GPa]</b> | <b>Torque [Nm]</b> | <b>Max Translation [m]</b> |
|------------------------|-----------------------------|--------------------|----------------------------|
| 199                    | 0.448                       | 30                 | 0.038                      |

Finally, the parameters needed for the MOGA are listed in Table 5.3. All values are chosen through empirical testing, including the penalty parameters. These parameters somehow form a penalty function to be applied constraint violations, but it is uncertain how this functions looks like, as it is not documented in AMOpt.

Table 5.3: Design optimization values for the GA method

| <b>Population Size</b> | <b>Number of Generations</b> | <b>Penalty Weight</b> | <b>Penalty Power</b> |
|------------------------|------------------------------|-----------------------|----------------------|
| 50                     | 20                           | 10                    | 2                    |

<sup>1</sup>Note that the E modulus is 199GPa. This is due a bug in AML that happens when writing to *.bdf*. Luckily, 199 GPa is an okay approximation for a steel E modulus, and does not need a workaround.

Given the values from Table 5.2, the general first constraint function can be reduced to:

$$g_1(\mathbf{x}) = \frac{3}{4600} \frac{2x_2 + x_3}{2x_1x_2^3 + 6x_1x_2x_3^2 + 12x_1x_2^2x_3 + 6x_1x_2^3 + x_4x_3^3} - 1 \leq 0 \quad (5.5)$$

Likewise, the second constraint function would be:

$$g_2(\mathbf{s}) = u_{max}(\mathbf{s}) - 0.038 \leq 0 \quad (5.6)$$

The values listed in the previous tables and equations are all the values needed to begin the optimization. However, running an analysis on the initial design parameters,  $\mathbf{x}_{init}$ , is beneficial, for comparison with the final results. The values for the initial design formulation functions are as follows:

$$\begin{aligned} f(\mathbf{x}_{init} = \{0.1, 0.02, 0.1, 0.02\}) &= 0.006 \\ g_1(\mathbf{x}_{init} = \{0.1, 0.02, 0.1, 0.02\}) &= -0.665 < 0 \\ g_2(\mathbf{s}_{init}) &= 0.0389 - 0.038 = 0.009 > 0 \end{aligned} \quad (5.7)$$

As we can see, the analysis results,  $\mathbf{s}_{init}$ , makes the second constraint function,  $g_2(\mathbf{s}_{init})$ , violated. The initial design is therefore in an infeasible area.

### The Final Optimization Result

After going through 20 generations of design points, the final result is shown in Table 5.4. The whole process took about 30 minutes, which means that it takes 1.5 minute to mesh the mechanism, and get results from a FEDEM analysis. Figure 5.4 shows the initial and the optimized cross sections, side by side.

Table 5.4: Final I-beam cross section values

| Flange width | Flange thickness | Web height | Web thickness |
|--------------|------------------|------------|---------------|
| 0.052        | 0.005            | 0.173      | 0.005         |

To show that the design optimization result leads to a better design, given the formulation, we can calculate the objective function and constraint functions for the design.

These values are:

$$f(\mathbf{x}_{final} = \{0.052, 0.005, 0.173, 0.005\}) = 0.001385$$

$$g_1(\mathbf{x}_{final} = \{0.052, 0.005, 0.173, 0.005\}) = -0.002186$$

$$g_2(\mathbf{s}_{final}) = 0.0322 - 0.038 = -0.0058$$

Obviously, this is a feasible design, since the values of both constraint functions are less than zero. In addition we see that the cross section area is  $0.001385 \text{ (m}^2\text{)}$ , when the initial had an area of  $0.006$ , which is around 4.3 times bigger.

An interesting result is that AMOpt concludes with a different solution than what our framework did, shown in Table 5.5. Of course, this is due to the extra translational constraint,  $g_2(\mathbf{s})$ , which is not a part of the AMOpt problem formulation.

Table 5.5: Final AMOpt I-beam cross section values

| Flange width | Flange thickness | Web height | Web thickness |
|--------------|------------------|------------|---------------|
| 0.050        | 0.005            | 0.172      | 0.005         |

Figure 5.5 and 5.6 plots respectively  $f(\mathbf{x})$  and  $g_1(\mathbf{x})$  against each design point, to show how they vary along the iteration process, retrieved from AMOpt.

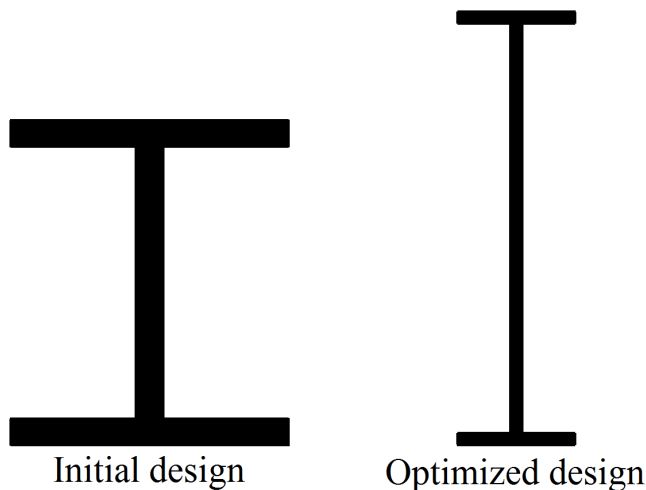


Figure 5.4: Initial design relative to the optimized design

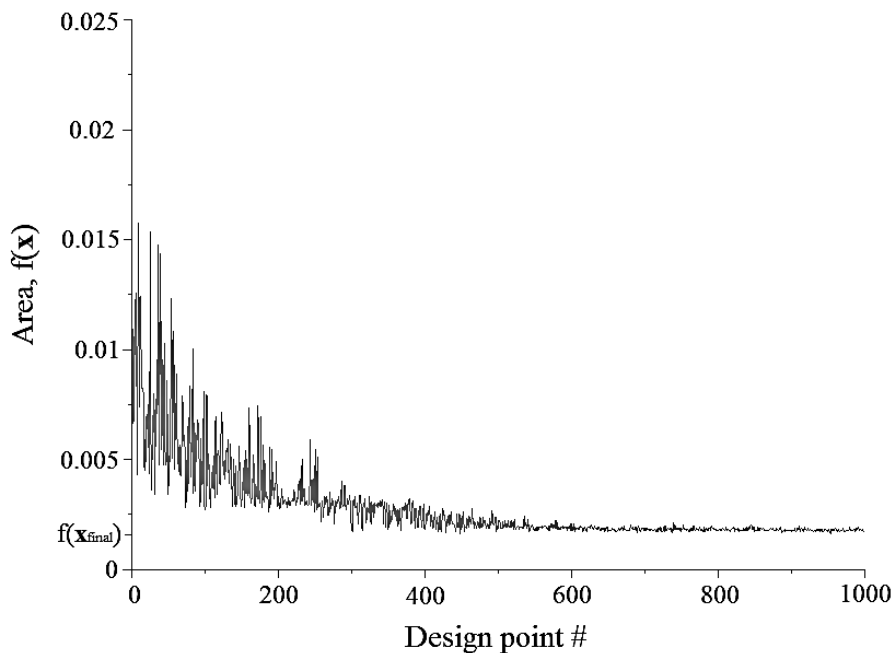


Figure 5.5: Area at each design point plot

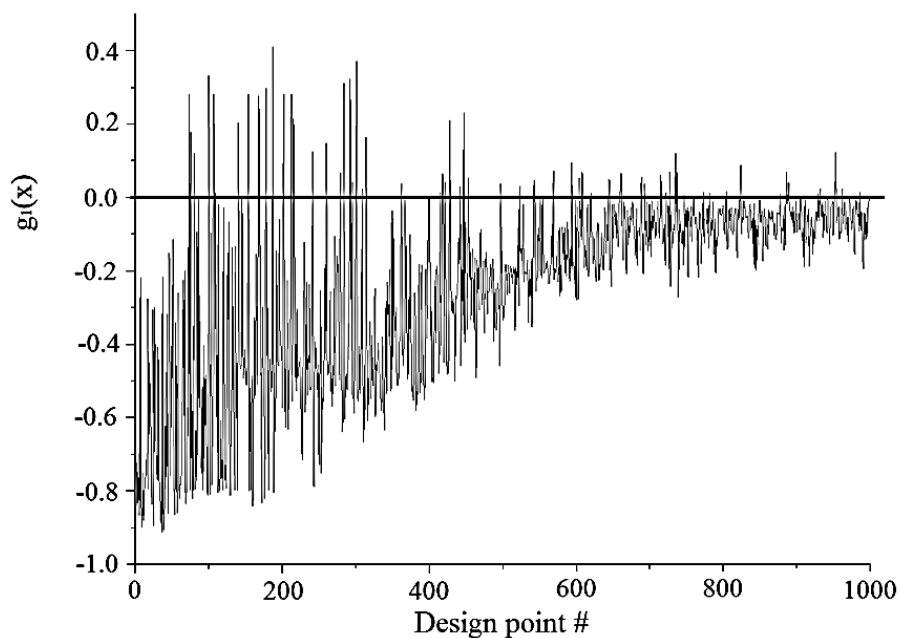


Figure 5.6: Constraint function at each design point plot

## 5.5 Discussion

Design optimization of four bar mechanisms is often a process to make the four bar-generate a line as straight as possible. However, it must still be able to withstand the forces applied at its crank, which the optimization example in this chapter deals with. The four-bar must tolerate the forces while still keeping its weight to a minimum, by reducing the area of all cross sections. Lower weight means lower material costs, which, of course, is highly sought-after for most companies. The straight line feature of the four-bar is also the reason that the second constraint,  $g_2(\mathbf{s})$ , is formulated as it is. The thought is that larger translations at the middle of the coupler link, will interfere more with the way the four bar moves. Of course, larger cross sections will give a more stable movement, but again, they will cost more, which means that a trade-off must occur.  $g_2(\mathbf{s})$  is an example of how to control this trade-off, although there was no calculation for how the maximum allowed translational value,  $u_p$ , was found. This value is set experimentally, and it seems to be good enough to illustrate the example.

Nevertheless, the example shows that that the system returns a strictly better design, given the problem formulation. The design is also well within the maximum translational bounds, which AMOpt's final result was not. We also notice that all design variables are within their move limits, although  $x_2$  and  $x_4$  took their respective minimum values, while  $x_1$  almost did. This makes it likely to believe that they would continue to decrease if  $g_2(\mathbf{s})$  was removed, and their lower bounds was lowered. This is mainly due to the simplicity of the problem formulation, where  $\sigma_{max}(\mathbf{x})$  is calculated without any concern of the relative thicknesses of the design variables, or other types of forces and stresses that will occur. Hence, without  $g_2(\mathbf{s})$  and the move limits, the formulation would probably make the cross section become longer and narrower as the iteration process goes on. However, some of the design variables will be more influential than the others, both in regards to lowering the objective function, but also the first constraint function. Obviously, a perturbation of  $x_1$  and  $x_2$  changes the objective function more than the others, because they dictate the dimensions of both flanges. But how they influence the  $g_1(\mathbf{x})$  is not that obvious. Looking at  $g_1(\mathbf{x})$  in Equation 5.5,  $x_2$  and  $x_3$  are the only variables of power 3, which makes them more influential to the function.



Deciding which variables to change more than others, through a sensitivity analysis, is a feature that could be desirable for the design optimization. According to Specht [44], these kind of substructural design variables also actually *require* a sensitivity analysis, which backs up this desire further.

If the optimized design from Figure 5.4 is considered too narrow, restrictions could be put on some of the move limits before running another optimization. Although, a better solution would probably be to add extra constraint functions, specifying relations between the design variables. For example, if we wanted the web height to always have a value of less than fifteen times larger than the web thickness, we could impose a third constraint saying that  $x_3 - 15x_4 \leq 0$ . But, since the point of this example was to show that an automated optimization loop has been made, rather than getting the actual best cross section, such constraints have not been made.

The plots from Figure 5.5 and Figure 5.6 illustrate why MOGA is not well suited for structural optimizations. First of all, it takes about 600 iterations before the area seems to stabilize, so running analyses on each design point is just too time costly. Many of them would also be redundant, as the design variables change by random factors in a population, which yields a lot of semi-equal designs. Secondly, since MOGA is a SUMT method, meaning that it incorporates the constraint functions in the objective function, many design points will be in an infeasible area. Figure 5.6 illustrates this issue, where all points above the black line at  $g_1(\mathbf{x}) = 0$  are violating the constraint. Therefore, other solutions might be more desirable to utilize in the future.

A similar solution to the one implemented could be to alter which designs that are analysed in FEDEM. Since the way FEDEM is included now might seem a bit forced and unnecessary, it would maybe make more sense to analyse the entire population of the last generation, instead of the best design of each generation. In that case, we need to compare  $g_1(\mathbf{x})$  and  $g_2(\mathbf{s})$ . If  $g_2(\mathbf{s})$  impose a 'stricter' constraint than  $g_1(\mathbf{x})$ , then all of the design points in the last generation might not satisfy the second constraint, which leaves the whole optimization without a result. If the constraints are equally strict, or  $g_1(\mathbf{x})$  is stricter, then this solution can possibly be better, although the process might take a while depending on the size of the population.

If we move away from the existing AMOpt methods, DOT or NPSOL could be investigated to map potential improved usages. Recall from Section 3.3.5 that these libraries include SLP and SQP methods, which are gradient-based and regarded as better methods than non-gradient-based methods, like the Nelder-Mead and Powell's method. If they in addition have a framework for incorporating constraint functions in the problem formulation, there is no doubt that these methods should be favored over the existing AMOpt methods. For example, the SQP methods could be baked into a convex approximation method, used to solve functionality optimizations, like the straight line functionality of a four-bar. In that case, the design variables would be the coordinates of the revolute joints, thereby moving away from a dimensional optimization process, into a functional one. Using a convex approximation method requires the implementation of custom approximation algorithms and gradient methods, which would have to be investigated further.

If DOT or NPSOL are not viable options and there still is a desire to utilize AMOpt, SUMT could be implemented. Then, the existing problem formulation would be augmented so that the objective function includes the constraints, and the Nelder-Mead or Powell could be applied. A drawback is that this setup requires penalty functions to be created, which are not configurable option in either of these methods in AMOpt, which means that some workarounds have to be created. For example, the infeasible designs could simply just not be sent to analysis.

# Chapter 6

## Implementation Details

Like discussed in Section 2.7, an object-oriented development process includes analyzing, designing and separating a system into meaningful classes and relations, and implementing the design as a set of objects. Based on this theory, we have derived a general development methodology, described in Section 6.1. Section 6.2 System Architecture will go through the mechanism system architecture, by describing class structures and purposes, and how they relate to each other. They will be illustrated using the modeling convention described in Section 3.5. Implementation details are also described. Larger images of the class diagrams can be found in Appendix B.

### 6.1 General Development Methodology

When constructing a system from scratch, or nearly from scratch, it is important to remember the object-oriented principles from Section 2.7. Not following the general guidelines of the principles may lead to undesirable aspects, like unwanted system behavior and poor maintainability of the system. The latter case was especially a problem, when we initially started working with the system. The problem became evident due to the lack of encapsulation, by allowing objects to directly change the state of other objects. Admittedly, there is no direct way of encapsulating object properties in AML, meaning that object access modifiers can not be set. If an object property is queried, it is always returned, which allows the object's state to be altered by any other object.

Regardless, accessor methods can be used to 'simulate' access modifiers. This is also regarded a better practice, because a class should not need to know or maintain knowledge of how another object is created, which the class does when altering the object's state directly. Rather, the class should only need to know the object's interface, through a set of *get* and *set* methods, along with a set of legal operation methods. Another advantage of working towards interfaces instead of modifying states directly, is that objects are more loosely coupled.

The system has been further developed with these thoughts in mind, while we continuously have been attempting to make it more loosely coupled. Note that the existing classes and relations by themselves, have been changed a little, but not as much as the tight coupling between them. The class diagrams in the following sections are created both for our own sake, as well as to make it less confusing for others to be introduced to the system. Additionally, if others are to continue working on the mechanism system, the GitHub repository is easily accessible, and contains the commit history, which is a documentation of the work process itself. All source code can either be viewed in the GitHub repository or in Appendix F.

## 6.2 System Architecture

### 6.2.1 Collections

All input parameters to the system are read from files, stored in a mechanism library, like portrayed in Section 4.1. For each file, there is a corresponding *collection-class* that is responsible for reading it and storing the data. Each line in a file generally represents an object, which is instantiated by a collection-class, through the use of the built-in AML functionality, *series-object*. Objects created as series-objects are added as subobjects for the object that created them. As such, the creator object can be viewed as an object constructor.

The *main-mechanism-class* is the main, or root class of the system, that the user should instantiate. Instantiation can be accomplished in one of the modeling forms, or through

the function (*create-model 'class-name*) in the AML console. All of the initial collection classes relate to the *main-mechanism-class*, either as properties or subobjects. For example, the *point-collection* reads all the coordinates from *coordinates.txt*, and instantiates a set of *point-data-models*, like shown in Figure 6.1. The other collection classes works analogously, except from *folder-collection*. This class instantiates a *folder-info-model* for each file existing under the *library* folder. For each *folder-info-model*, there is created a series of *mechanism-version-info* objects, representing a specific version of a mechanism. Originally, this class was thought to contain information about the different mechanism parameters during an optimization process, representing a history of the iteration process. However, now it represents more of a mechanism specification, while the *folder-info-model* represents a general mechanism type.

The *main-mechanism-class* is also a *series-object* itself. It will instantiate optimization objects based on the input data. Note that it is possible to have multiple optimization objects, if different kinds of optimizations is desirable.

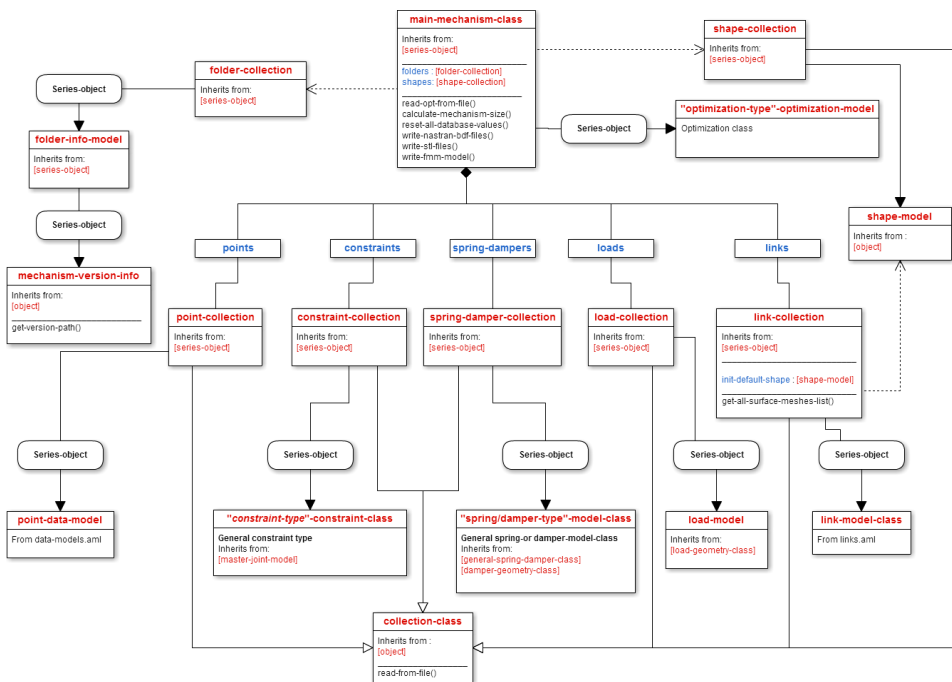


Figure 6.1: Class-object diagram of the initial collections

## 6.2.2 Data Models

AML encourages to use classes that contain domain specific knowledge, as superclasses for other classes. The domain specific knowledge is then inherited into subclasses, which can be mixed with their own knowledge. Three examples used in the mechanism system are shown in Figure 6.2. Classes inheriting from for instance *frame-data-model* will not only inherit its orientation and coordinate system properties, but also the definitions of how its GUI should look like.

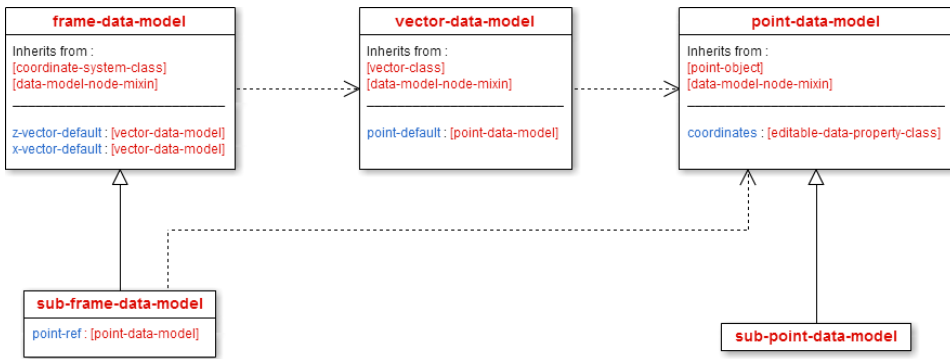


Figure 6.2: Class-object diagram of the initial collections

## 6.2.3 Joints

Joints are modeled in a fashion based on its two distinct elements, the female and the male. Since the concatenation of these form the entire joint, they are modeled as sub-objects for a *master-joint-model* holding the two. The *master-joint-model* functions as both a model holding the two joint elements, as well as a model containing shared joint meta data, like the common joint dimensions. Figure 6.3 shows this relation. It also shows how both joint elements are of the type *joint-element-model*.

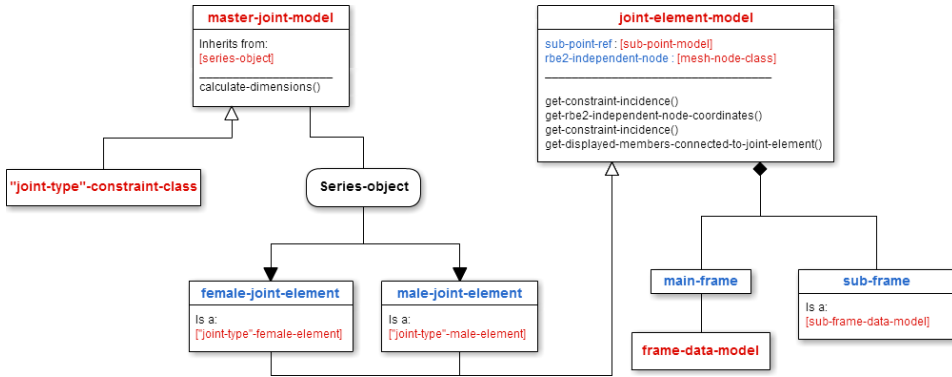


Figure 6.3: Class-object diagram of the general joint models

Figure 6.4 shows the different joint implementations of the *master-joint-model* and *joint-element-model*.

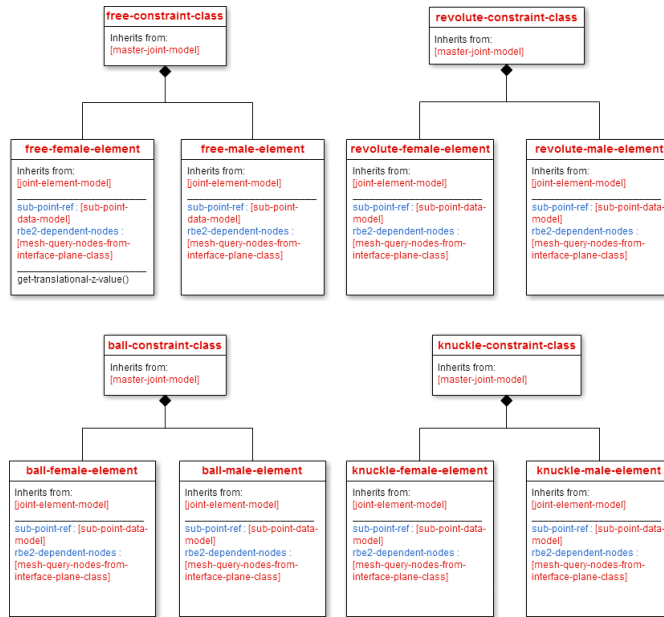


Figure 6.4: Class-object diagram of the joint types

## 6.2.4 Links

As explained in Section 4.7, joint geometry and link geometry are created separately, before assembled. The class that holds all information about the link geometry, i.e. the





All three models share the same class as subobjects, the *mesh-query-nodes-with-label-class*. In essence, this is just a query object, containing the load's, spring's or damper's connection node. As explained in Section 4.5, this node is obtained from an existing mesh. The way it is obtained is by searching through the corresponding link mesh, via a method implemented in *mesh-query-nodes-from-interface-class*, which the connecting node object inherits from.

The visual representation for loads, springs and dampers are implemented as 'geometry-classes' and are inherited into the models, respectively, in order to separate functionality.

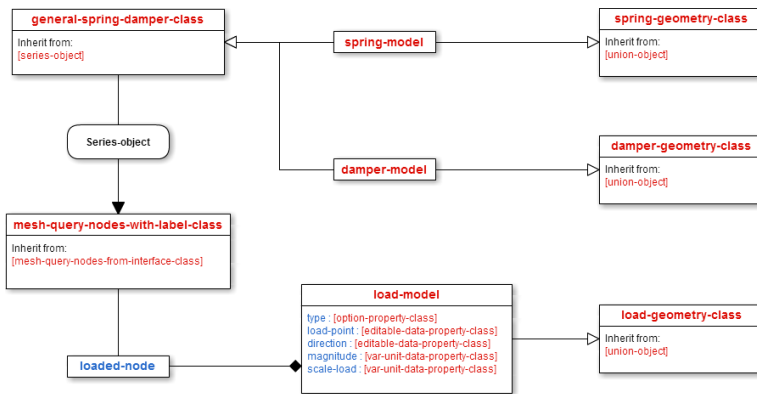


Figure 6.6: Class-object diagram of loads, springs and dampers

## 6.2.6 Meshing and Analysis

Surface meshing in the system is executed by a *paver-mesh-class*. This class takes an arbitrary geometry and generates a triangle, quadrilateral or hybrid mesh, depending on the input parameters. It is this class that controls the mesh refinement, described in Section 4.8, and it will also apply mesh sizes specified mesh sizes for any tagged objects. These tagged objects are *joint-element-model*, *member-solid-model*, *surface-model* and the *blend-object*, found in previous class diagrams. Finally, the *paver-mesh-class* requires an instance of a *mesh-database-class*, which is both used for storing and querying. This database object can be saved to file, so that it can be retrieved later. The *tet-mesh-class* takes a surface mesh as argument and creates a tetrahedron mesh, using the surface

elements. The solid mesh it generates, will have the same element edge lengths as the surface mesh.

The *analysis-link-model-class* is mainly used for pre-processing. For instance, all surface and volume mesh nodes elements are gathered and applied material properties. In addition, all RBE2 nodes are gathered from the joints. By themselves, these nodes are just a collection with no formal meaning. This meaning is provided to them by adding them as properties in the *analysis-rigid-body-element-type-1-class*<sup>1</sup>.

The rest of the *analysis-link-model-class* subobjects are instances of predefined AML classes. The *nastran-interface* is there to collect these objects and serves as an interface to NX Nastran and exporting the mesh and its properties to *.bdf* files.

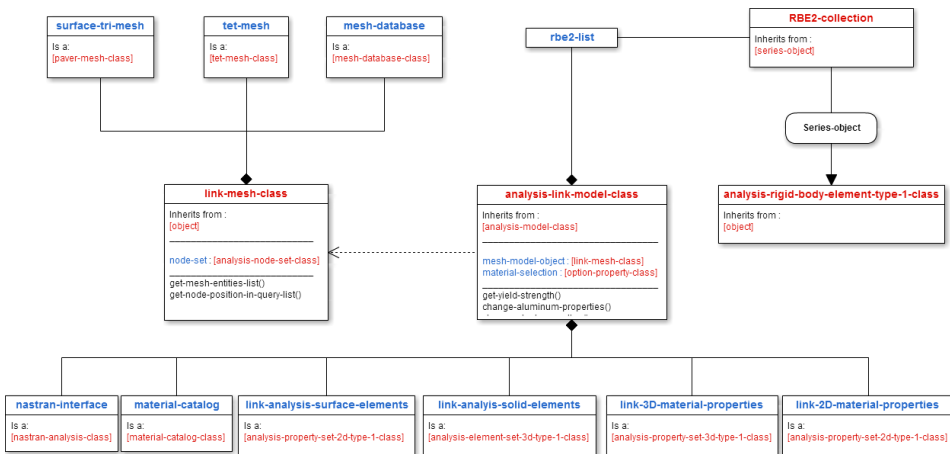


Figure 6.7: Class-object diagram of the meshing and analysis models

## 6.2.7 Design Optimization

Any type of a design optimization model is represented by a *general-optimization-class*. Specific optimization types will all inherit from this class, to ensure a good object-oriented design and reusability of code. As it stands, it is only possible to perform a structural optimization of cross sections. This is modeled in *cross-section-optimization-model*. It uses instances of *design-variable-classes* and a *constraint-class* and an objec-

<sup>1</sup>The name *analysis-rigid-body-element-type-1-class* is correct, even though it actually is an RBE2 class.

tive function property to model the formulation. Data from the optimization input file is stored in these instances.

The iterative optimization process is carried out by the AMOpt module, which is designed for a user to set up an optimization case interactively. It works by assigning properties from the object tree as design variables, objective functions and constraint functions. Manually setting every variable with upper and lower bounds, is a rather time consuming process, so an easier and semi-automatic process has been developed. As shown in Figure 6.8, the optimization model uses a set of AMOpt classes. When setting up a case interactively, these classes are created by AMOpt. Now, they are explicitly instantiated, through a custom made GUI, where upper and lower bounds for the design variables can be set. Not only is this setup more user friendly, as it saves time, but it also ensures a correct and controlled use of the model, since it connects the actual cross section parameters to refer to the design variables. This feature implies that a change in the design variables, also changes the cross section parameters, due to dependency backtracking.

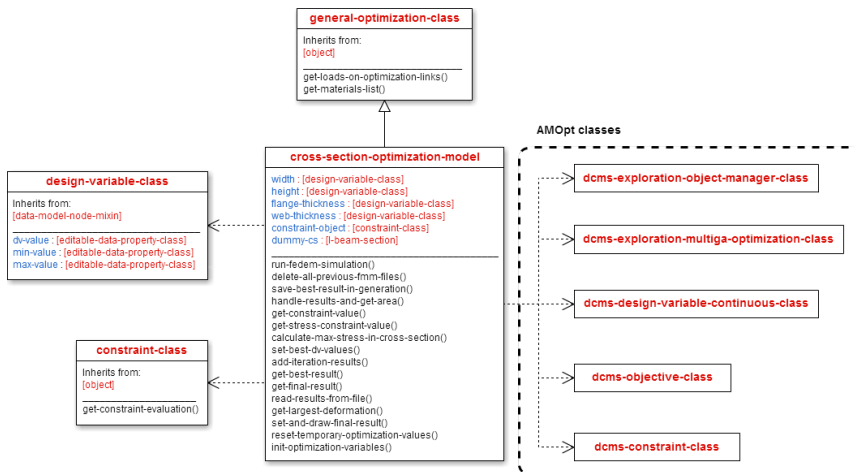


Figure 6.8: Class-object diagram of the optimization models

Section 5.2 mentioned running FEDEM analyses through batches. This feature is implemented in the *cross-section-optimization-model*, and is a twofold process. First, the model file is reduced and solved, using the command:

```
fedem -f 'model-file-path' -solve dynamics
```

Then, the appropriate results are retrieved using:

```
fedem_graphexp -curvePlotFile 'results-file-path' -curvePlotType 5  
               -modelfile 'model-file-path'
```

This creates a file located at 'results-file-path', which contains a time history of deformations at a given node position. The file is subsequently read to locate and store the largest deformation in the analysis.

### 6.3 Results and Discussion

Evaluating the way a system is implemented is a difficult task, due to the lack of specific metrics for measuring the implementation. However, after following the methodology from Section 6.1, we can identify some differences between the old and new version of the system. First of all, some data flow was quite confusing, as there was a lot of unnecessary data going back and forth when it did not need to. An example is the way the joint size was determined, by sending data back and forth between *master-joint-model* and *joint-element-model* (Figure 6.3). Without going into details, this process is now simplified, and hopefully less confusing. Secondly, some unneeded classes and constructs are removed. An example in this case is that both *master-joint-model* and *joint-element-model* inherited from an overlying *general-joint-model-class*. Defining both the components and the assembly to be of the same type, does not make sense in this case, so it was duly removed, thereby freeing up a layer of abstraction, and un-complicating the architecture.

Another aspect from the development methodology is the use of accessor methods. In the old state of the system, if a variable name was changed, or a class was modified, a set of objects directly depending on the variable would also have to be updated, which is a poor design for maintainability. New features of the system have now been implemented to avoid this kind of dependence. It is a small and simple measure, but with the access interfaces defined, the code is expectantly better for reuse and maintenance. However, the system has become quite large and complex, so there is still some work to be done in this area.

The class diagrams have proved to be great tools in the process of both learning and debugging the system. They provide a visual representation that are not easily obtainable when only viewing the source code itself, or by viewing the model tree in the modeling forms, described in Section 3.3.4. The model tree created when choosing a four bar with circular cross section is shown in Figure 6.9 (different objects are expanded in the two figures). We believe that using the inspect form in combination with the class diagrams, is probably the fastest way to gain a good understanding of the system, by visualizing how the system is built up. The only drawback of the inspect form is that it does not show class methods. This is an unfortunate factor that might lead to more methods being implemented as properties, which can facilitate a poorer architecture.

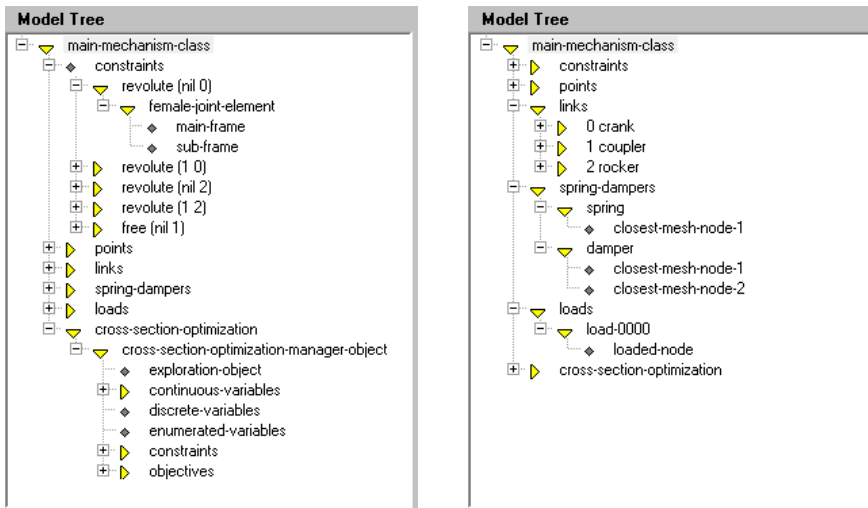


Figure 6.9: System model tree

Finally, the development infrastructure using a GitHub repository has proven a vital tool for collaboration and simultaneous editing of source code. The commit history and preceding work log are found in Appendix E.



## Chapter 7

# Final Discussion

The discussions of Chapters 4, 5 and 6 went into detail about the respective chapter's content. This discussion will address a more general view, drawing lines between the chapters, in the context of the research questions.

### **RQ1 How can a KBE system automate tasks in the mechanism design process?**

As described in Section 2.2 a goal for KBE applications is automating repetitive and non-creative design tasks, by means of capturing, storing and reusing domain knowledge. The mechanism system has been developed with these principles in mind. A parametrized model is used to maintain the connection between entities and their underlying abstractions. This enables knowledge to be captured, abstracted and subsequently used throughout different stages of the design process. A large variety of mechanisms can be stored in a central library, enabling the designer to reuse the knowledge obtained from previous design tasks.

Generally, the pre-processing is the most time-consuming process of undertaking an analysis. This thesis demonstrates how simple user input can be used to fully automate pre-processing, eliminating the need for repetitive user interactions. Designer interaction is still possible through a GUI, resulting in a high degree of flexibility. The thesis

also shows the staging of a common platform for mechanism design, by transforming the internal AML model to a FEDEM model file. This integration was facilitated by the parametrized model, in combination with the object-oriented structure of AML. Encapsulation of entities such as links and joints, containing their own domain knowledge, ensures the modularity necessary for handling the integration.

The mechanism system provides the initial framework for a complete KBE system for mechanism design. A designer can either create new mechanisms from scratch or edit the topology and geometry of predefined mechanisms in the library. In either case, both the input files and the GUI provides the tools needed to best solve the design task at hand. Considering a large number of designs at an early stage in the design process, is highly desirable. The mechanism system enables automatic modeling and subsequent pre-processing of mechanisms, allowing analyses to be run right away and even integrated in feasibility studies of preliminary designs.

### **RQ2 How can a KBE system support and enable the use of design optimization?**

Like discussed in Section 2.1, one of the aims of using KBE tools is to facilitate the connection between the design process and numerical analyses. Chapter 5 shows how this connection is integrated in the mechanism system, using AMOpt algorithms to drive the process. The design optimization has been greatly supported by the already parametrized mechanism model. Any number of link cross section parameters can point to overlying design variables. As a result of dependency backtracking in AML, all cross sections will change in accordance with the design variables, subsequently updating the mechanism model.

As shown in Chapter 5, the mechanism system provides the means to run a fully automated design optimization process. However, complete automation is in most cases not beneficial. As stated in Section 2.6.6, a successful optimization requires the designer to not be completely left out of the optimization loop. The designer should be able to guide the iteration process and evaluate intermediate results, thereby controlling the



progress. In the current state of the system, only the problem formulation is controlled, leaving the whole iteration process to the computer. However, by running short design optimizations, the designer is able to iteratively guide the design, using results from previous optimizations.

Integrating design optimization in the design process is no straightforward task. It requires a great degree of abstraction, both in the system and from the designer, who has to formulate the actual optimization problem. Even though design optimization can be beneficial in many design tasks, it might prove too extensive for most design processes. However, a possibility could be to create a library consisting of predefined problem formulations to assist the designer. In association with a mechanism library, this could help explore a larger design space, create smarter solutions and increase innovation.



# Chapter 8

## Conclusions

This thesis demonstrates how tasks in the design process can be automated, using principles of Knowledge-Based Engineering.

The mechanism system has been reviewed and further developed. The underlying kinematic model and joint definition have been altered to achieve a more stable mechanism representation. In addition, generic loads, springs and dampers, created through simple user input, have enabled a more versatile and thorough modeling of mechanisms. With the introduction of fillets, link geometries have become smoother and more realistic. Arguably, they have also improved the quality of finite element meshes, by allowing custom mesh refinements to be set at typical high stress areas. The mesh quality has been further improved by curvature and transitional refinements. An interface with AML and FEDEM has been implemented with an automated pre-processing of the mechanism model, reducing the need of repetitive user interactions.

A generic design optimization of I-beam cross sections is integrated with the use of the design optimization module, AMOpt. As an example, the cross sections of a four-bar mechanism have been optimized, leading to strictly better results, given the problem formulation. The example demonstrates how the mechanism model can be used in a fully automated design process, including structural analyses in FEDEM.

The mechanism system has been restructured in compliance with object-oriented design principles, achieving a higher degree of modularity and maintainability. A set of

class diagrams have been created, both to illustrate the system architecture and lower the threshold for further development.

# Chapter 9

## Further Work

This chapter suggests areas for further study and development.

Even though the input files for the system has been made more human readable, they rely on *.txt* files, which might be too basic. A possible solution is converting to a commonly used file structure, like XML, which both seems more professional and would probably make the files even more readable. However, this is not critical for the system itself.

A more critical task for the system is to implement more joint types. For example, parametrized models of cylindrical and prismatic joint types could be made based on the suggestions in section 4.8.1. Parametrization of higher joint types should also be looked into.

A feature to connect loads, springs and dampers to an MPC, if the closest mesh node is internal, should be implemented.

A broader variety of FE mesh types could be looked into, both to vary the number of nodes in an element, and to create other volume mesh elements than just the tetrahedron type.

FEDEM does not necessarily need to be the analysis module for the mechanism system, as built-in AML classes can be used to directly retrieve results from an integrated

NX Nastran module. We have not made this work, and it is uncertain if the module supports dynamical multi-body simulations, but it is worth looking into. If the use of FEDEM continues, a broader set of pre-processing procedures (constructing the *.fmm* file) should be implemented. Some examples include creating a generic control system (based on user input), non-linear springs and dampers, load types depending on mathematical functions, and the possibility for adding masses.

Regardless of the analysis module utilized, a post-processing framework should be developed, where the analysis results could be evaluated against a set of pre-defined criteria. This framework could also be integrated in the design optimization loop.

As of now, mechanisms are not ready for production. Determining and implementing features for extended detailing should be done.

Based on the suggestions from Section 5.5, other ways to use AMOpt, as well as integrating the external libraries DOT and NPSOL, could be looked into. If the design optimization is further implemented, the design variable models, should be made more generic. Now, they are sizing variables, and they should be able to be system variables, or other types of substructural variables, for the design optimization framework to be as complete as possible. A complete framework would also include more optimization cases, not just a cross section optimization, and it should allow the designer to be more involved in the iteration process.

# References

- [1] Arora, J.S. (1989). *Introduction to Optimum Design*. McGraw-Hill, New York.
- [2] Bansal, R.K., Brar, J.S. (2004), *Theory of Machines* Laxmi Publications LTD, New Delhi.
- [3] Beck, K. (2000). *Extreme Programming Explained: Embrace Change*, Addison-Wesley.
- [4] Bhattacharya, S. K. (2008). *Control Systems Engineering*. Pearson Education, New Delhi.
- [5] Booch, G. (1986). *Object-Oriented Development*. In: IEE Transactions on Software Engineering, Vol SE-12, pp 211-221.
- [6] Bongardt, B. (2013). *Sheth-Uicker Convention Revisited* Robotics Innovation Center, Bremen, pp 2-3.
- [7] Budynas, R.G., Nisbett, J.K. (2011). *Shigley's Mechanical Engineering Design*. McGraw-Hill, New York.
- [8] Chapman, C.B., Pinfold, M. (1999). *Design engineering – a need to rethink the solution using knowledge based engineering*. In: Knowledge-Based Systems 12, pp 257–267.
- [9] Chapman, C.B., Preston, S., Pinfold, M., Smith, G. (2007). *Utilising enterprise knowledge with knowledge-based engineering*. In: Int. J. Computer Applications in Technology 28, pp 2-3.

- [10] Darmofal D., *16.100 Aerodynamics, Fall 2005*. (Massachusetts Institute of Technology: MIT OpenCourseWare). Retrieved May 3, 2016, from: [http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-100-aerodynamics-fall-2005/lecture-notes/16100lectre48\\_cj.pdf](http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-100-aerodynamics-fall-2005/lecture-notes/16100lectre48_cj.pdf).
- [11] Etman, L.F.P. (1997). *Optimization of Multibody Systems using Approximation Concepts*. Doctoral Dissertation Eindhoven University of Technology, Eindhoven.
- [12] Etman, L.F.P, Van Campen, D.H, Schoofs, A.J.G. (1998). *Design Optimization of Multibody Systems by Sequential Approximation*. In: *Multibody System Dynamics 2*, pp 393-415.
- [13] Fedem Technology AS (2016). *FEDEM*. Retrieved May 13, 2016, from: <http://www.fedem.com/software/>.
- [14] Fedem Technology AS (2012). *Fedem Theory Guide, Release 7.0*
- [15] Fedem Technology AS (2013). *Fedem User's Guide, Release 7.0.3*
- [16] Felippa, C.A. (2004). *Introductions to Finite Element Methods*. University of Colorado, Boulder, Colorado,
- [17] Finger, S., Behrens, S. *Introduction to Mechanisms*. Retrieved April 28, 2016, from: <https://www.cs.cmu.edu/~rapidproto/mechanisms>
- [18] GitHub, Inc. *GitHub*. Website: <https://github.com/>.
- [19] Haftka, R. T., Gürdal, Z. (1992). *Elements of Structural Optimization*. KLUWER ACADEMIC PUBLISHERS, Dordrecht.
- [20] Heckbert, P. S. (1993). *Introduction to Finite Element Methods*. Global Illumination Course, Carnegie Mellon University.
- [21] Hughes, J. (2015). *Pair Programming*. Retrieved December 2, 2015, from: <http://cs.brown.edu/courses/csci0170/content/docs/pair-programming.pdf>
- [22] Hutcheson, M.L, (2003). *Software Testing Fundamentals, Methods and Metrics*. Wiley Publishing, Inc., Indianapolis.



- [23] Khurmi, R.S., Gupta, J.K. (2005). *Theory of Machines*. Eurasia Publishing House.
- [24] Kristoffersen, E., Kristiansen, A. (2015). *Knowledge Based Engineering in mechanism design, automating the design loop*. Project dissertation, Norwegian University of Science and Technology, Trondheim.
- [25] MSC Software. *RBES and MPCs in MSC.Nastran*. Retrieved May 2. 2016, from: [https://cdm.ing.unimo.it/files/progettazione\\_assistita/corso\\_2011\\_2012/2012\\_05\\_16\\_mer/RBES.ppt](https://cdm.ing.unimo.it/files/progettazione_assistita/corso_2011_2012/2012_05_16_mer/RBES.ppt)
- [26] Don Ho. *Notepad++*. Retrieved June 7. 2016, from: <https://notepad-plus-plus.org/>
- [27] NTNU, Department of Marine Technology. *The Finite Element Method - Theory*. Retrieved April 29. 2016, from: <http://illustrations.marin.ntnu.no/structures/analysis/FEM/theory/index.html>
- [28] Object Management Group Inc. *Unified Modeling Language (UML) Resource Page*. Retrieved December 1. 2015, from: <http://www.uml.org/>
- [29] Olagunju, A.O, Akpan, B. (2015). *The Benefits of Object-oriented Methodology for Software Development*. In: International Journal of Information and Computer Science, Vol 4, pp 39-46
- [30] Pivotal Labs. *Pivotal Tracker*. Retrieved December 3. 2015, from: <https://www.pivotaltracker.com/help/faq#whatispivotaltracker>
- [31] Press, W.H., Teukolsky, S.A., Vetterling W. T., Flannery B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*. Cambridge University Press, Cambridge, pp 408-416.
- [32] Ravi, V. (2011). *Theory of Machines (Kinematics)*. PHI Learning, New Delhi.
- [33] Rocca, G.L. (2001). *Knowledge based Engineering techniques to support aircraft design and optimization*. Doctoral dissertation, Faculty of Aerospace Engineering, TU Delft, Delft.
- [34] La Rocca, G. (2012). *Knowledge based engineering: Between AI and CAD. Review of a language based technology to support engineering design*. In: Advanced Engineering Informatics 26, pp 159-179.

- [35] *Scrum Methodology*. Retrieved December 2, 2015, from: <http://scrummethodology.com>
- [36] Sheth, P. (1972). *A Digital Computer Based Simulation Procedure for Multiple Degree of Freedom Mechanical*. University of Microfilms, Michigan.
- [37] Singh, S. (2011). *Theory of Machines*. Pearson, India.
- [38] Sivertsen, O.I. (2001). *Virtual testing of mechanical systems, theories and techniques*. Swets & Zeitlinger B.V., Lisse.
- [39] Skaare, R.K. (2015). *Mechanism parametrization, modeling and FE-meshing*. Master's thesis, Norwegian University of Science and Technology, Trondheim.
- [40] Sobieszczanski-Sobieski, J., Morris, A., van Tooren, M. (2015). *Multidisciplinary Design Optimization supported by Knowledge Based Engineering*. John Wiley & Sons, Ltd., New Jersey.
- [41] Stanford Business Software Inc. *NPSOL*. Retrieved May 11, 2016, from: [http://www.sbsi-sol-optimize.com/asp/sol\\_npsol5.0description.htm](http://www.sbsi-sol-optimize.com/asp/sol_npsol5.0description.htm)
- [42] Stress Ebook LLC. *RBE2 Vs RBE3*. Retrieved May 2, 2016, from: <http://www.stressebook.com/rbe2-vs-rbe3/>
- [43] Specht, B. (1992). *Optimization Theory and Software Requirement Document*. ESPIRIT 5524 MDS Technical Report, Dornier.
- [44] Specht, B. (1992). *Sensitivity Theory and Software Requirement Document*. ESPIRIT 5524 MDS Technical Report, Dornier.
- [45] Takeuchi, H., Ikujiro, N. (1986). *The New New Product Development Game*. In: Harvard Business Review 64, no. 1.
- [46] Technosoft Inc. *Adaptive Modeling Language*. Retrieved May 11, 2016, from: <http://www.technosoft.com>
- [47] TechnoSoft Inc. (2010). *AML Reference Manual 5.0B5*.
- [48] Thornton, S.T., Marion, J.B, *Classical Dynamics of Particles and Systems (Fifth Edition)* (2003), Brooks Cole

- [49] Trier, S.D. (2001). *Design Optimization of Flexible Multibody Systems*. Doctoral dissertation, Norwegian University of Science and Technology, Trondheim.
- [50] Verhagen, W.J.C., Bermell-Garcia, P, van Dijk, R.E.C., Curran, R. (2012). *A critical review of Knowledge-Based Engineering: An identification of reseach challenges*. In: *Advanced Engineering informatics* 26, pp 5-15
- [51] Vanderplaats Research & Development, Inc. *Design Optimization Tools*. Retrieved May 11. 2016, from: <http://www.vrand.com/sites/default/files/pub/DOT%20Brochure.pdf>
- [52] Xemacs community. *What is XEmacs?*. Retrieved December 1. 2015, from: <http://www.xemacs.org/>
- [53] Young, W.C., Budynas, R.G. (2002). *Roark's Formulas for Stress and Strain*. McGraw-Hill, New York.

# Appendix A

## Installation Details

The following instructions is valid for the Windows operating system only. In order to run the mechanism system the following software has to be installed: AML, MSC Nastran (included in Siemens PLM Software NX) and FEDEM. In addition, four AML modules is also required: aml-analysis-module-pack-type-3, aml-analysis-module-pack-type-3\_ui, amsketcher-module and AMOpt.

The files 'logical.pth' and 'aml-init.tsi' is found in the folder where AML is installed, e.g. 'C:/Program Files/Technosoft/AML/AML6.31\_x64'.

- Add a path in 'logical.pth' pointing to the location of the mechanism system, e.g. ':mechanism-system "C:/Users/User/mechanism-system/"'. This is required in order to compile the mechanism system.
- If the paths from ':tmp' and ':temp' in 'logical.pth' does not exist on the computer, change the paths or create the subsequent folder.
- Add a path in 'logical.pth' pointing to the location of 'nastran.exe', e.g. ':nastran-path "C:/Program Files/Siemens/NX 10.0/NXNASTRAN/bin/"'. This is required in order to export to .bdf files.
- Add a path in 'logical.pth' pointing to where nastran files should be saved, e.g. ':nastran-data "C:/Users/User/mechanism-system/nastran-data/"'. This is required in order to export to .bdf files.

- Load the modules. The modules are added in 'aml-init.tsi', e.g. '(load-module "aml-analysis-module-pack-type-3" :path "C:/Program Files/Technosoft/AML /AML6.31\_x64/modules/)". This is required in order to run AMSketcher, optimization, and perform meshing and analysis.
- For automatic compilation of the system on startup of AML, add 'compile-system :mechanism-system' in 'aml-init.tsi'. This is optional.
- When running design optimization one might have to start Emacs in administrator mode. This is situational due to different user setting for running batch commands.

# **Appendix B**

## **Class Diagrams**

This appendix contains large images of each class diagram presented in Chapter 6.

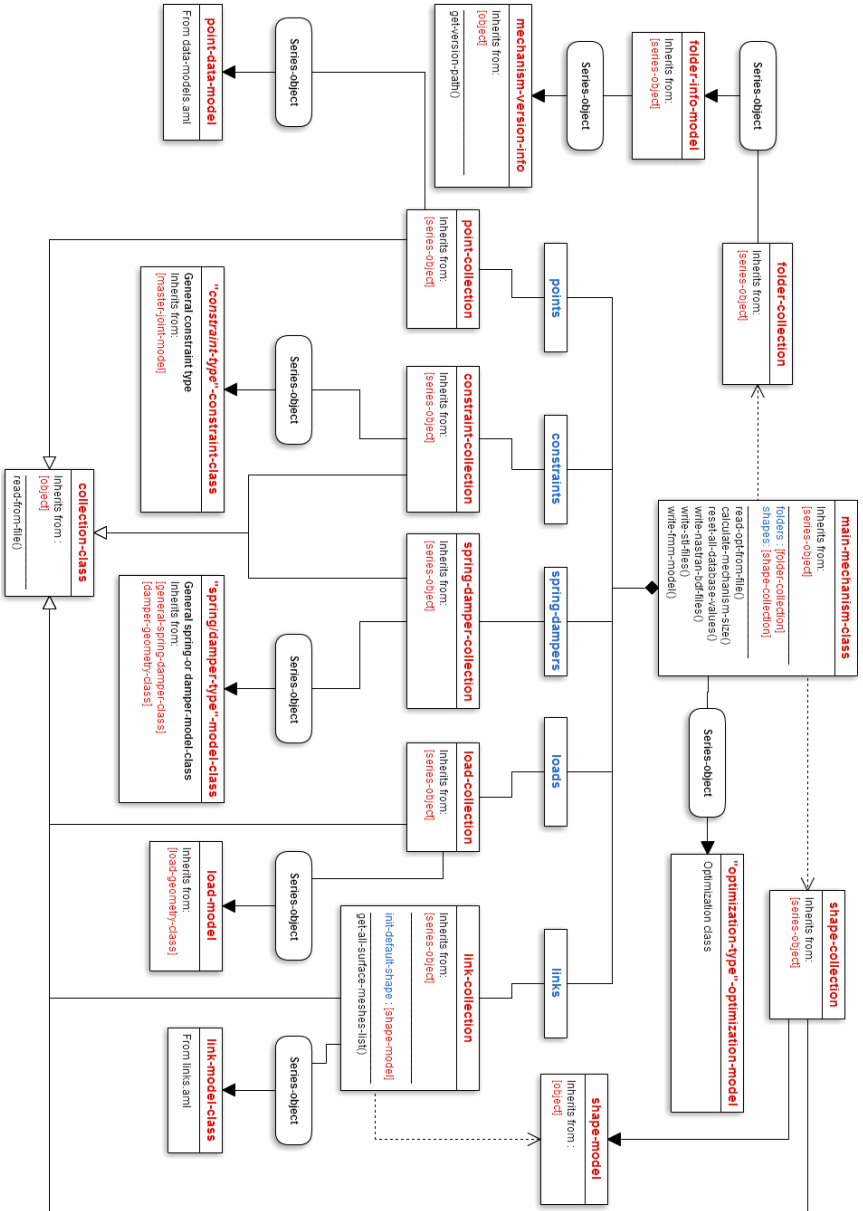


Figure B.1: Class-object diagram of the initial collections

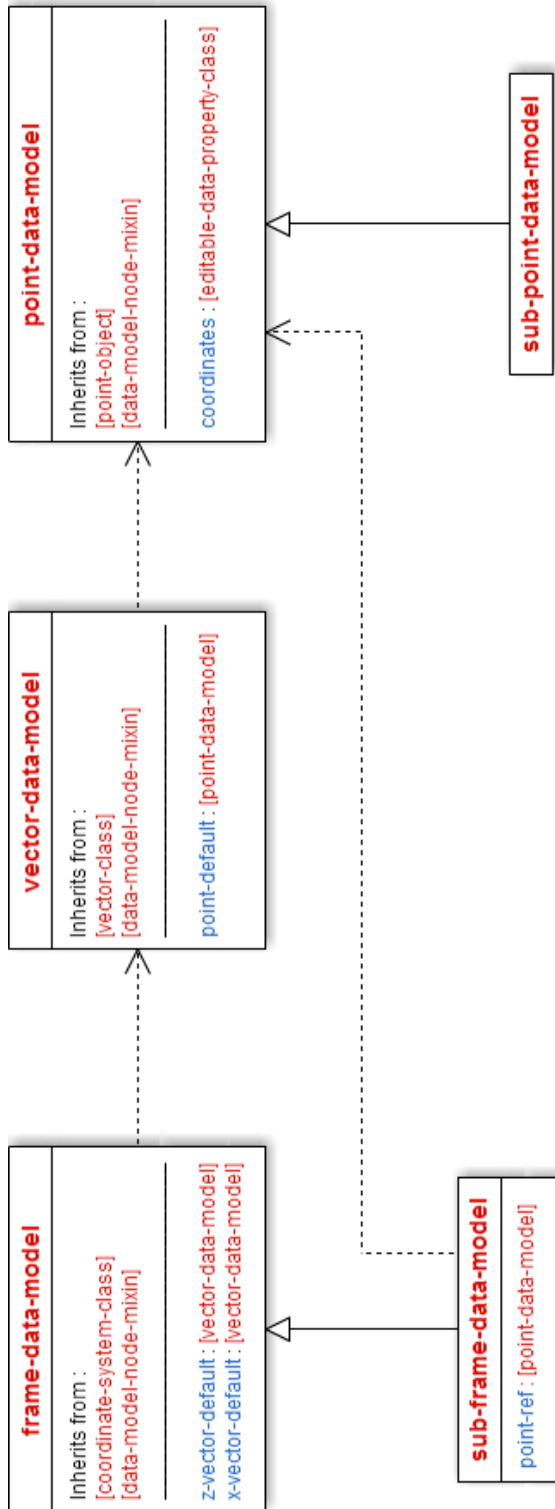


Figure B.2: Class-object diagram of the initial collections



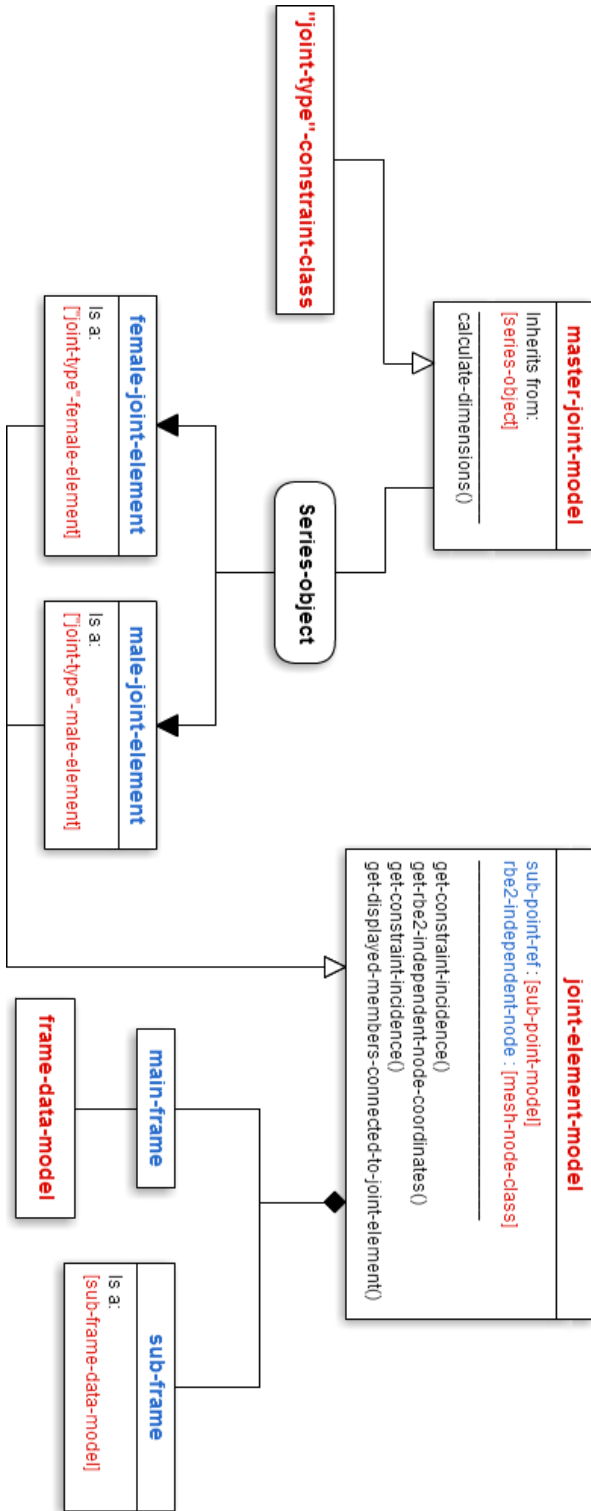


Figure B.3: Class-object diagram of the general joint models

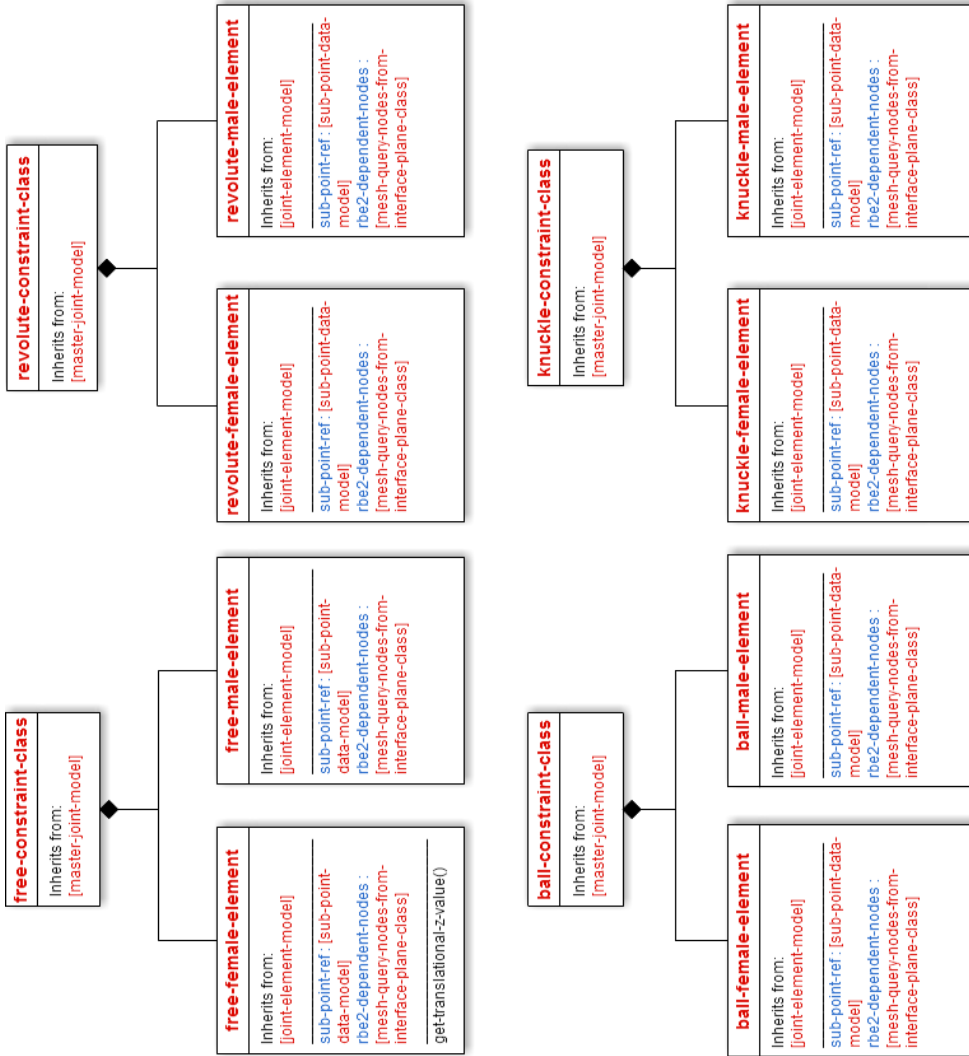


Figure B.4: Class-object diagram of the joint types

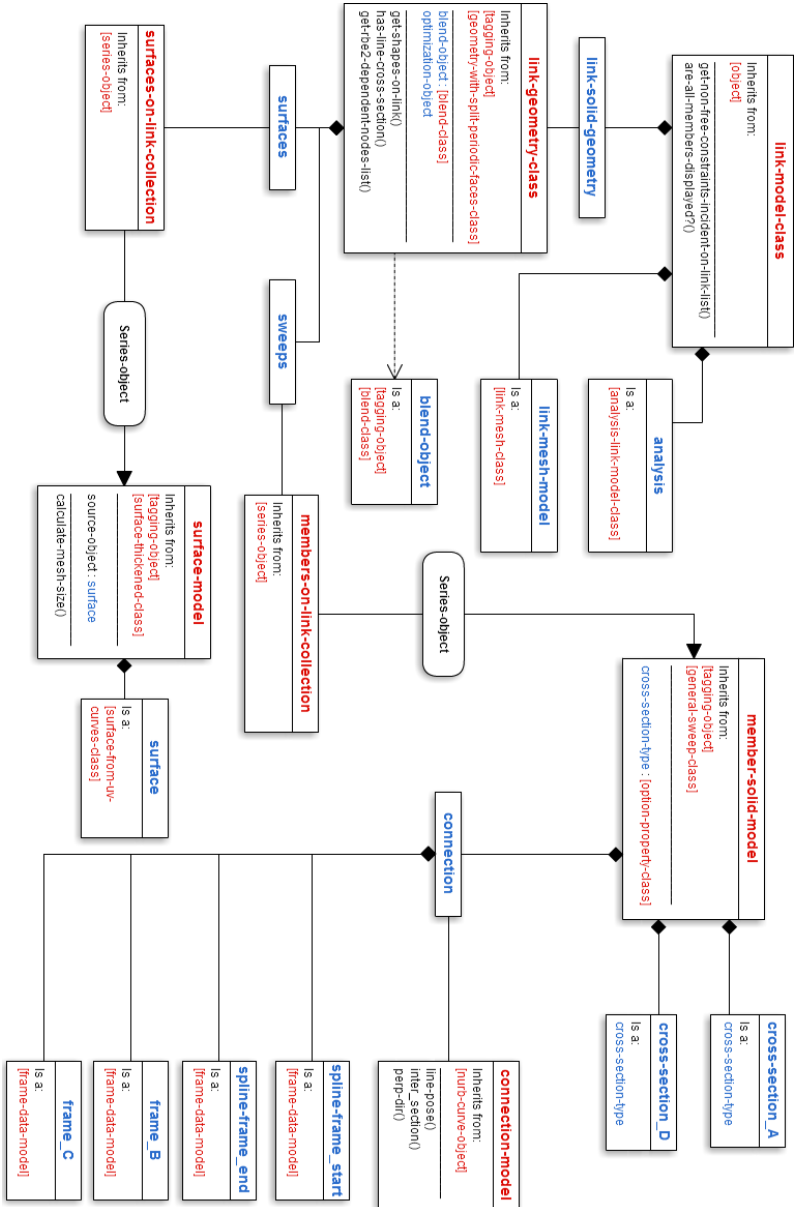


Figure B.5: Class-object diagram of the link structure

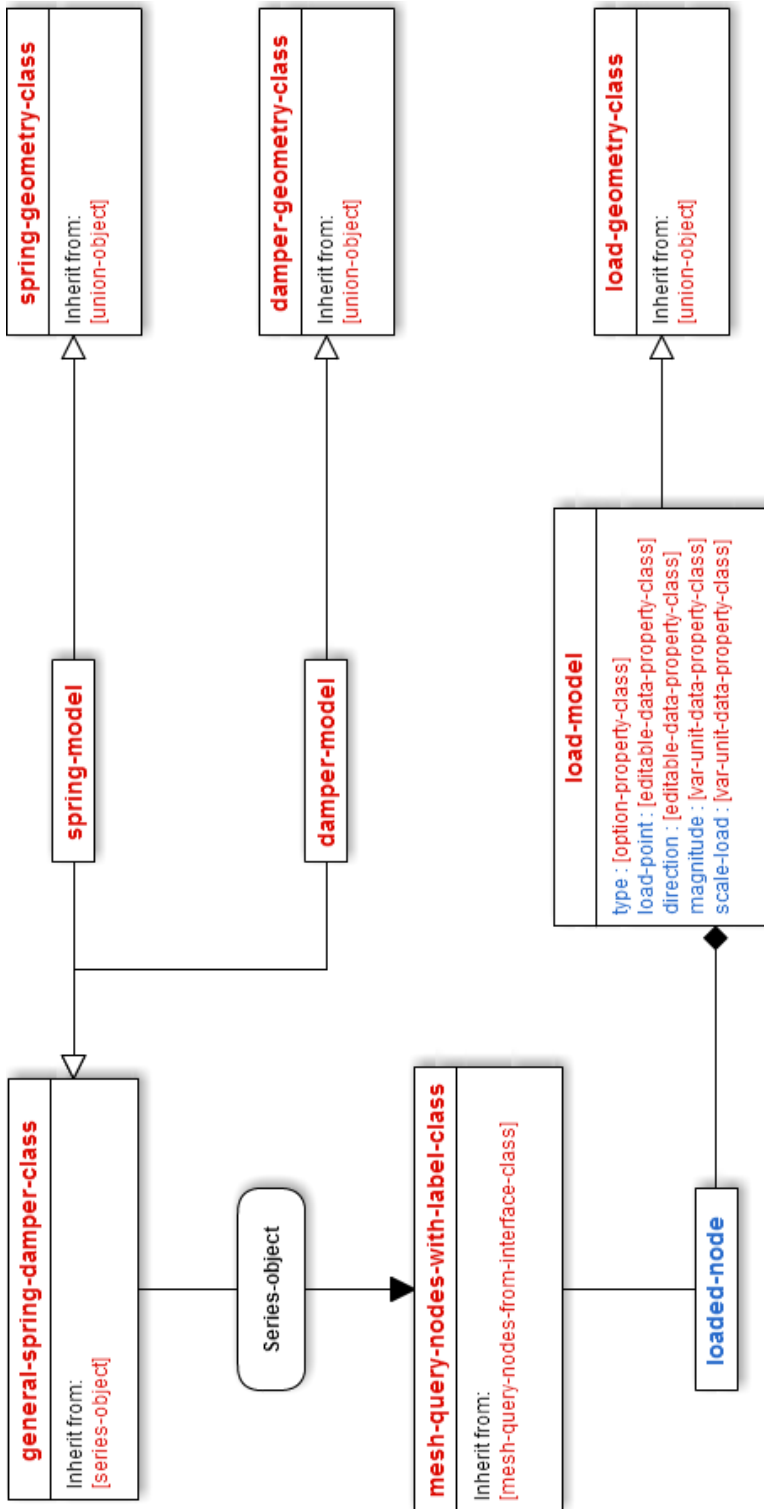


Figure B.6: Class-object diagram of loads, springs and dampers

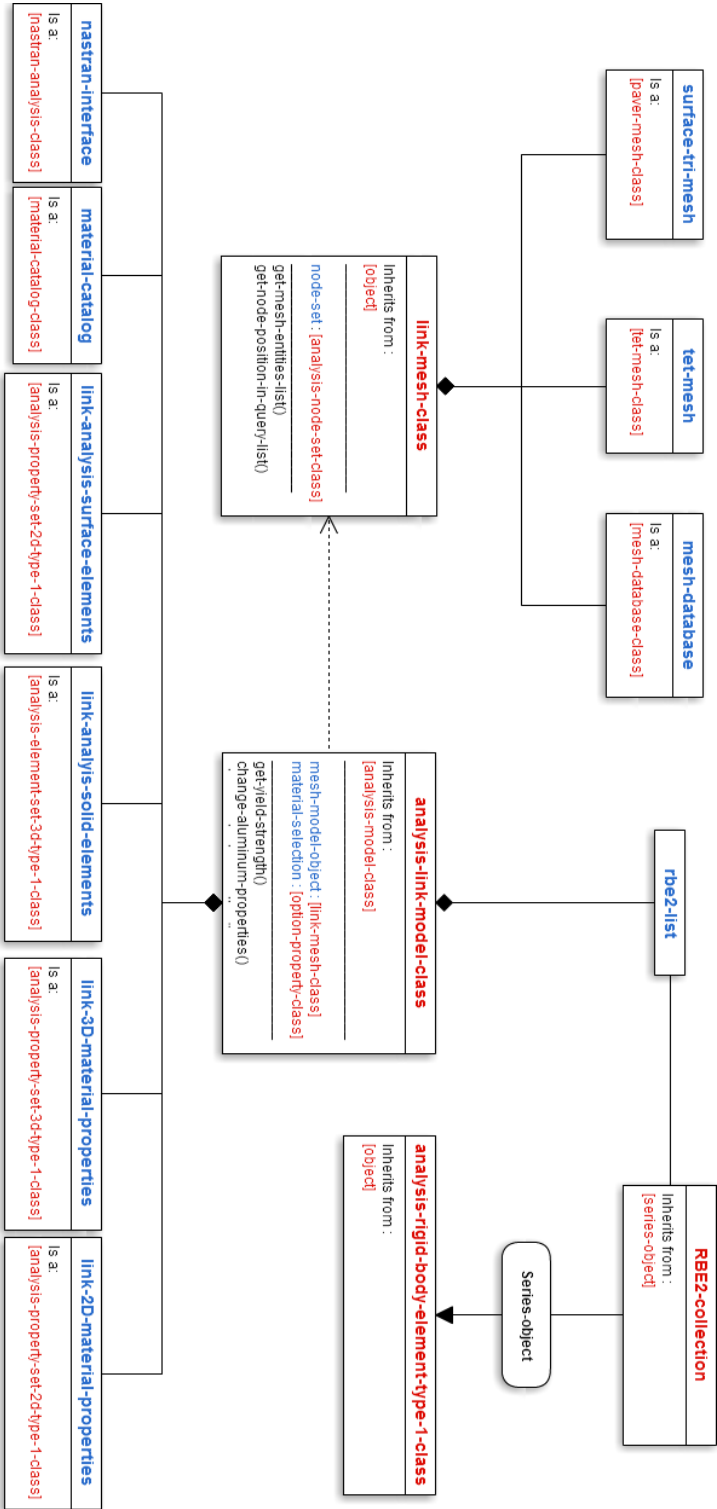


Figure B.7: Class-object diagram of the meshing and analysis models

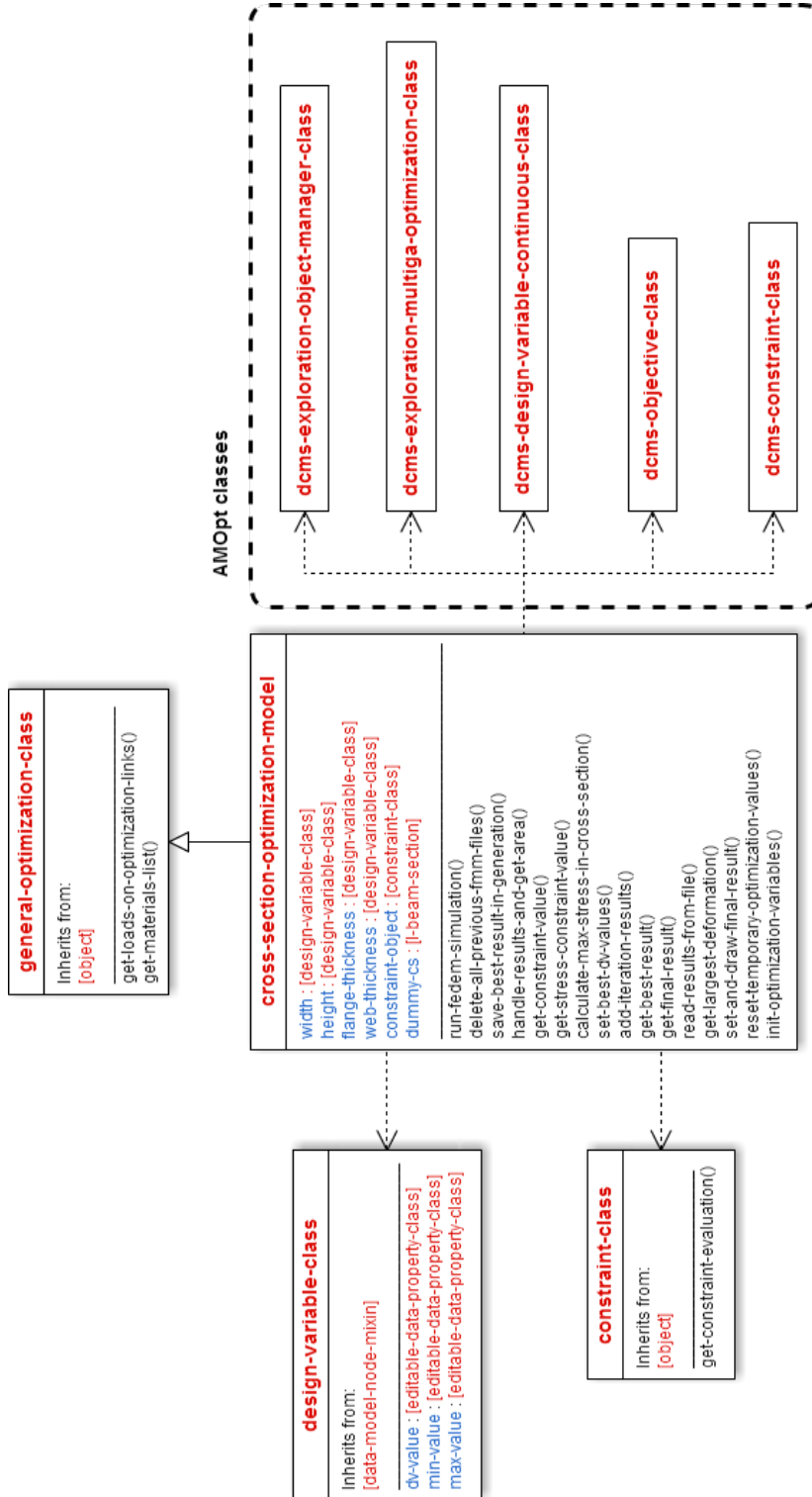


Figure B.8: Class-object diagram of the optimization models

# Appendix C

## Graphical User Interface

This Appendix will shortly describe each individual GUI frame (not to be confused with coordinate frames) for the most important objects of the model tree. A demo of the GUI functionality can be found on: <https://vimeo.com/170026181>.

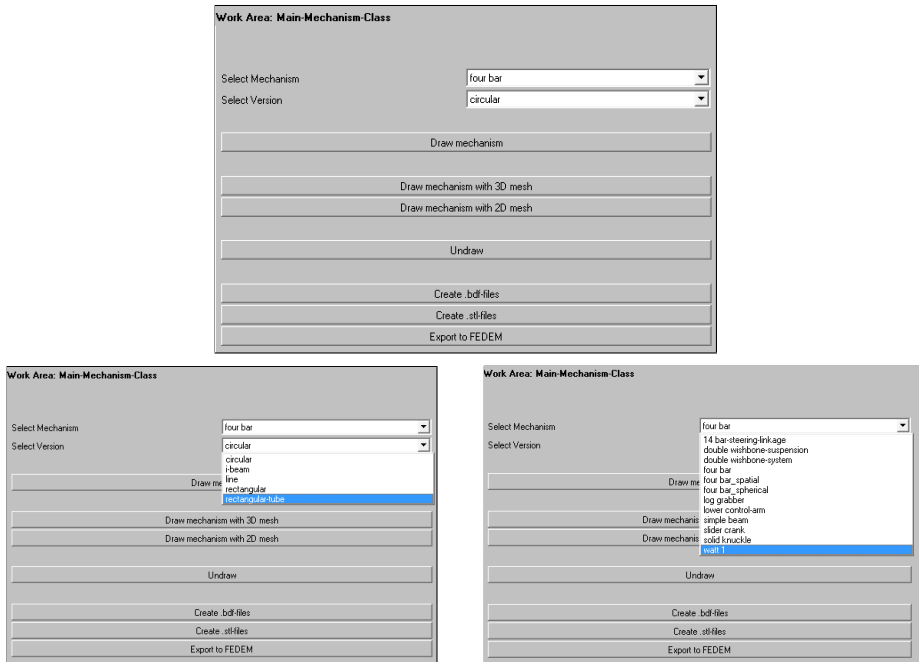


Figure C.1: Mechanism selection and export GUI

Figure C.1 shows the top level frame. At this frame, the different mechanisms and their versions, can be selected. Switching between mechanisms will update the model tree. The frame also contains drawing and exporting functions.

Figure C.2 shows the properties editable for a coordinate system frame, like main-frames, sub-frames, start- and end-frames.

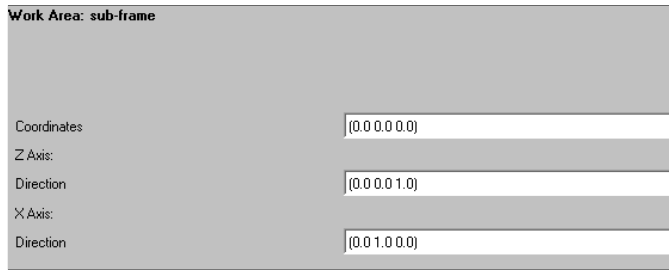


Figure C.2: Coordinate system frame GUI

Joint models have two GUI frames. One of them contains common sizing variables for both joint elements (both elements must have the same sizes to match), seen in Figure C.3a. Each joint element also has a GUI frame, letting the user draw and visualize the joint geometry, and all RBE2 nodes. In addition, the elements element mesh size can be changed.

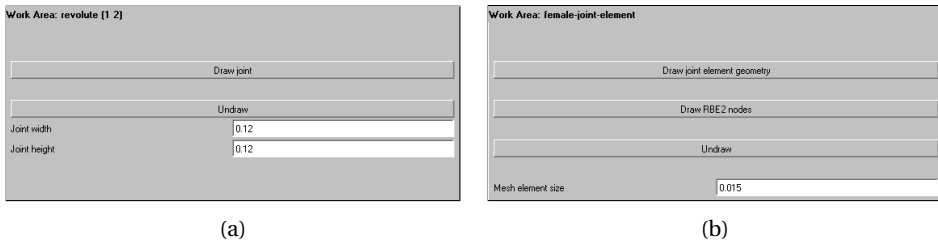


Figure C.3: Joint GUI

The general appearance for all links (shape and size) can be changed at the top level object of all links, shown in Figure C.4.



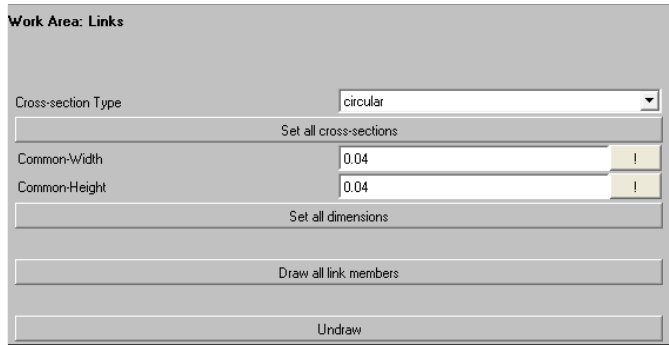
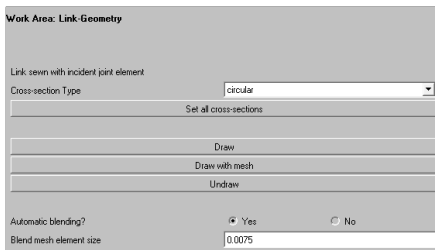
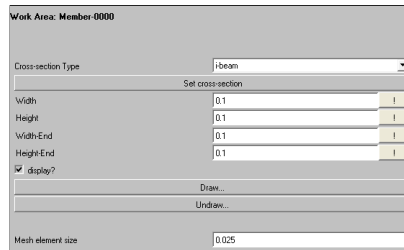


Figure C.4: GUI controlling all links

Specific shape, geometry size and mesh size can be set at the GUI frame for a member (Figure C.5b). Similarly, the shape of all members on a link can be set at the link GUI frame (Figure C.5a). In addition, the mesh size for the blend can be changed here.



(a)



(b)

Figure C.5: Link geometry and member GUI

Figure C.6 shows how to include a link surface through the 'display' check-box, and how to set its element mesh size.

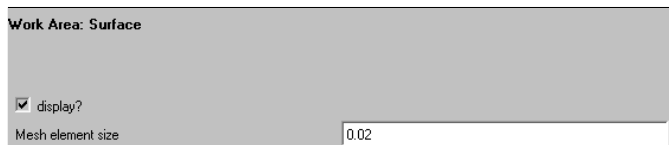


Figure C.6: Overall links GUI

The springs and dampers GUI contains buttons for drawing their 'geometries' and mesh connection nodes. Figure C.7a only shows the damper GUI frame, but the spring frame

is identical, as they inherit from the same superclass. The spring and damper coefficients can also be edited, to avoid having to change it in the input file. Load metadata can also be edited, seen in Figure C.7b.

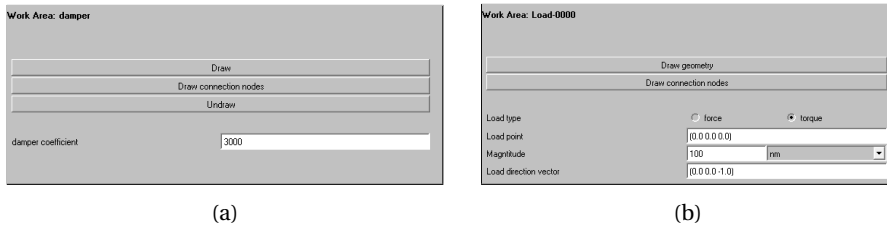


Figure C.7: Dampers and loads

The AML mesher generating surface meshes, comes with its own GUI frame, shown in Figure C.8. In this frame, the mesh type (structured, unstructured and hybrid) and refinement values can be set. The mechanism system initiates these refinement values, so changing them should be done carefully.

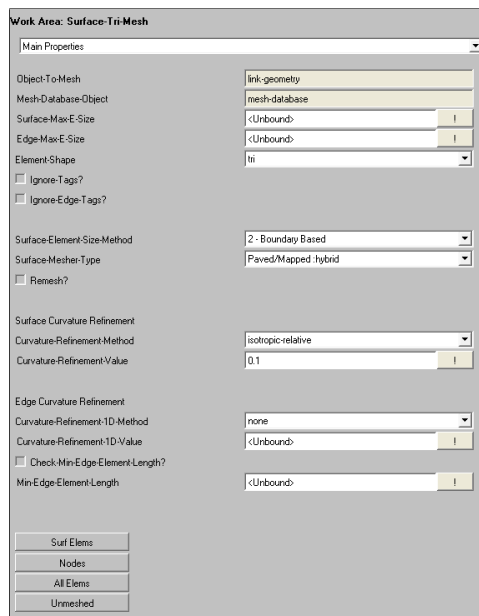


Figure C.8: Overall links GUI

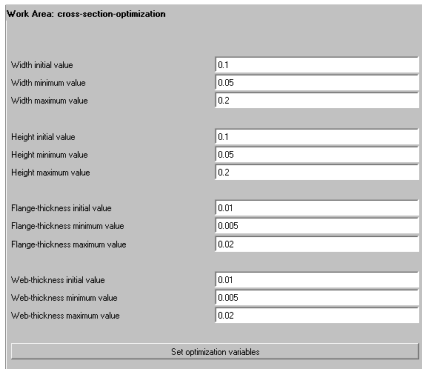
The material for a link can be set at the analysis GUI frame. Once the material is picked,

all material properties can also be edited.

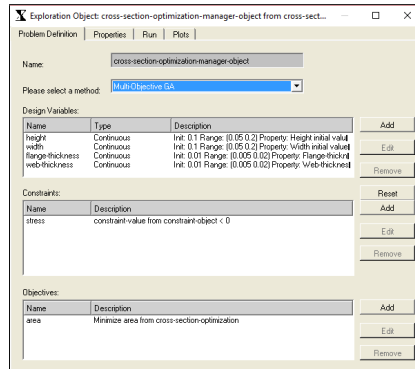


Figure C.9: Link material properties

Figure C.10a shows where the initial values and move limits for the design variables in a design optimization are set. These values are used to automatically instantiate the correct objects for the AMOpt model. The interface for the problem formulation in AMOpt is shown in Figure C.10b.



(a)



(b)

Figure C.10: Design optimization GUIs

# Appendix D

## Example Model File

The following Fedem Model File is automatically generated from the mechanism system and represents a double wishbone suspension with parts (points to .bdf files), joints, springs, dampers, and loads.

```
FEDEMMODELFILE {R7.0.4 ASCII}

GLOBAL_VIEW_SETTINGS
{
  ID = 1;
  SYMBOL_SCALE = 0.1;
  SYMBOL_LINE_WIDTH = 1;
  BACKGROUND_COLOR = 0.098039 0.305882 0.458823;
  CAMERA_FOCAL_DIST = 0.707107;
  CAMERA_HEIGHT = 1.41421;
  CAMERA_ORIENTATION =
  1.00000000 0.00000000 0.00000000 0.00000000
  0.00000000 1.00000000 0.00000000 0.00000000
  0.00000000 0.00000000 1.00000000 0.70710678;
}

MECHANISM
{
  ID = 1;
  BASE_ID = 1;
  GRAVITY = 0 0 -9.81;
  POSITION_TOLERANCE = 0.0001;
}

REF_PLANE
{
  ID = 1;
  BASE_ID = 2;
  HEIGHT = 0.1;
  WIDTH = 0.1;
  COLOR = 1 1 1;

  TRANSPARENCY = 0.65;
}

LINK
{
  BASE_ID = 3;
  COORDINATE_SYSTEM =
  1.00000000 0.00000000 0.00000000 0.00000000
  0.00000000 1.00000000 0.00000000 0.00000000
  0.00000000 0.00000000 1.00000000 0.00000000;
  ID = 1;
  LINE_COLOR = 1 1 1;
  MASS_PROP_DAMP = 0;
  ORIGINAL_FE_FILE = "C:\mechanism-system-path\
double-wishbone-suspension\link-0000.bdf";
  POLYS_ON_POINTS_OFF = true;
  STIF_PROP_DAMP = 0;
  USE_MASS_CALCULATION = true;
}

TRIAD
{
  BASE_ID = 4;
  ID = 1;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 0.0
  0.0 1.0 0.0 0.0
  0.0 0.0 1.0 0.0;
  LOCAL DIRECTIONS = GLOBAL;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  OWNER_LINK = 1;
```

APPENDIX D. EXAMPLE MODEL FILE

```

FE_NODE_NO = 10017;
NDOFS = 6;
}

TRIAD
{
BASE_ID = 5;
ID = 2;
COORDINATE_SYSTEM =
1.0 0.0 0.0 0.25
0.0 1.0 0.0 0.0
0.0 0.0 1.0 1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = 1;
FE_NODE_NO = 10018;
NDOFS = 6;
}

TRIAD
{
BASE_ID = 6;
ID = 4;
COORDINATE_SYSTEM =
1.0 0.0 0.0 0.25
0.0 1.0 0.0 0.0
0.0 0.0 1.0 -1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = 1;
FE_NODE_NO = 10019;
NDOFS = 6;
}

TRIAD
{
BASE_ID = 7;
ID = 6;
COORDINATE_SYSTEM =
1.0 0.0 0.0 0.0
0.0 1.0 0.0 0.55
0.0 0.0 1.0 0.1;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = 1;
FE_NODE_NO = 10020;
NDOFS = 6;
}

LINK
{
BASE_ID = 8;
COORDINATE_SYSTEM =
1.00000000 0.00000000 0.00000000 0.00000000
0.00000000 1.00000000 0.00000000 0.00000000
0.00000000 0.00000000 1.00000000 0.00000000;
ID = 2;
LINE_COLOR = 1 1 1;
MASS_PROP_DAMP = 0;
ORIGINAL_FE_FILE = "C:\mechanism-system-path\
double-wishbone-suspension\link-0001.bdf";
POLYS_ON_POINTS_OFF = true;
STIF_PROP_DAMP = 0;
}

USE_MASS_CALCULATION = true;
}

TRIAD
{
BASE_ID = 9;
ID = 3;
COORDINATE_SYSTEM =
1.0 0.0 0.0 0.25
0.0 1.0 0.0 0.0
0.0 0.0 1.0 1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = 2;
FE_NODE_NO = 3156;
NDOFS = 6;
}

TRIAD
{
BASE_ID = 10;
ID = 7;
COORDINATE_SYSTEM =
1.0 0.0 0.0 1.25
0.0 1.0 0.0 0.5
0.0 0.0 1.0 1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = 2;
FE_NODE_NO = 3157;
NDOFS = 6;
}

TRIAD
{
BASE_ID = 11;
ID = 8;
COORDINATE_SYSTEM =
1.0 0.0 0.0 1.25
0.0 1.0 0.0 -0.5
0.0 0.0 1.0 1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = 2;
FE_NODE_NO = 3158;
NDOFS = 6;
}

LINK
{
BASE_ID = 12;
COORDINATE_SYSTEM =
1.00000000 0.00000000 0.00000000 0.00000000
0.00000000 1.00000000 0.00000000 0.00000000
0.00000000 0.00000000 1.00000000 0.00000000;
ID = 3;
LINE_COLOR = 1 1 1;
MASS_PROP_DAMP = 0;
ORIGINAL_FE_FILE = "C:\mechanism-system-path\
double-wishbone-suspension\link-0001.bdf";
POLYS_ON_POINTS_OFF = true;
STIF_PROP_DAMP = 0;
USE_MASS_CALCULATION = true;
}

```

## APPENDIX D. EXAMPLE MODEL FILE

---

```

}

TRIAD
{
  BASE_ID = 13;
  ID = 5;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 0.25
  0.0 1.0 0.0 0.0
  0.0 0.0 1.0 -1.0;
  LOCAL_DIRECTIONS = GLOBAL;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  OWNER_LINK = 3;
  FE_NODE_NO = 10817;
  NDOFS = 6;
}

TRIAD
{
  BASE_ID = 14;
  ID = 9;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 1.5
  0.0 1.0 0.0 0.5
  0.0 0.0 1.0 -1.0;
  LOCAL_DIRECTIONS = GLOBAL;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  OWNER_LINK = 3;
  FE_NODE_NO = 10818;
  NDOFS = 6;
}

TRIAD
{
  BASE_ID = 15;
  ID = 10;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 1.5
  0.0 1.0 0.0 -1.0
  0.0 0.0 1.0 -1.0;
  LOCAL_DIRECTIONS = GLOBAL;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  OWNER_LINK = 3;
  FE_NODE_NO = 10819;
  NDOFS = 6;
}

REV_JOINT
{
  BASE_ID = 16;
  COORDINATE_SYSTEM =
  6.123233995736766E-17 0.0 -1.0 0.0
  0.0 1.0 0.0 0.0
  1.0 0.0 6.123233995736766E-17 0.0;
  HAS_Z_TRANS_DOF = false;
  ID = 1;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  MASTER_TRIAD = 11;
  MOVE_MASTER_TRIAD_ALONG = false;
  MOVE_SLAVE_TRIAD_ALONG = false;
  ROT_FORMULATION = FOLLOWER_AXIS;
  ROT_SEQUENCE = ZYX;
  ROT_SPRING_CPL = NONE;
}

SLAVE_TRIAD = 1;
TRAN_SPRING_CPL = NONE;
VAR_QUADRANTS = 0 0 0;
Z_ROT_STATUS = FREE;
Z_TRANS_STATUS = FREE;
}

TRIAD
{
  BASE_ID = 17;
  ID = 11;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 0.0
  0.0 1.0 0.0 0.0
  0.0 0.0 1.0 0.0;
  LOCAL_DIRECTIONS = GLOBAL;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  OWNER_LINK = -1;
  FE_NODE_NO = -1;
  NDOFS = 6;
}

BALL_JOINT
{
  BASE_ID = 18;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 0.0
  0.0 1.0 0.0 0.0
  0.0 0.0 1.0 0.0;
  FRICTION_DOF = 3;
  ID = 2;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  MASTER_TRIAD = 3;
  MOVE_MASTER_TRIAD_ALONG = false;
  MOVE_SLAVE_TRIAD_ALONG = false;
  ROT_FORMULATION = FOLLOWER_AXIS;
  ROT_SEQUENCE = ZYX;
  ROT_SPRING_CPL = NONE;
  SLAVE_TRIAD = 2;
  TRAN_SPRING_CPL = NONE;
  VAR_QUADRANTS = 0 0 0;
  X_ROT_STATUS = FREE;
  Y_ROT_STATUS = FREE;
  Z_ROT_STATUS = FREE;
}

BALL_JOINT
{
  BASE_ID = 19;
  COORDINATE_SYSTEM =
  -1.0 0.0 0.0 0.0
  0.0 -1.0 0.0 0.0
  0.0 0.0 -1.0 0.0;
  FRICTION_DOF = 3;
  ID = 3;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  MASTER_TRIAD = 5;
  MOVE_MASTER_TRIAD_ALONG = false;
  MOVE_SLAVE_TRIAD_ALONG = false;
  ROT_FORMULATION = FOLLOWER_AXIS;
  ROT_SEQUENCE = ZYX;
  ROT_SPRING_CPL = NONE;
  SLAVE_TRIAD = 4;
}

```

APPENDIX D. EXAMPLE MODEL FILE

```

TRAN_SPRING_CPL = NONE;
VAR_QUADRANTS = 0 0 0;
X_ROT_STATUS = FREE;
Y_ROT_STATUS = FREE;
Z_ROT_STATUS = FREE;
}

BALL_JOINT
{
  BASE_ID = 20;
  COORDINATE_SYSTEM =
  0.981327572779731 0.0 0.0 0.0
  0.0 0.9805806756909202 0.0384615384615384 0.0
  0.0 -0.0384615384615384 0.9805806756909202 0.0;
  FRICTION_DOF = 3;
  ID = 4;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  MASTER_TRIAD = 12;
  MOVE_MASTER_TRIAD_ALONG = false;
  MOVE_SLAVE_TRIAD_ALONG = false;
  ROT_FORMULATION = FOLLOWER_AXIS;
  ROT_SEQUENCE = ZYX;
  ROT_SPRING_CPL = NONE;
  SLAVE_TRIAD = 6;
  TRAN_SPRING_CPL = NONE;
  VAR_QUADRANTS = 0 0 0;
  X_ROT_STATUS = FREE;
  Y_ROT_STATUS = FREE;
  Z_ROT_STATUS = FREE;
}

TRIAD
{
  BASE_ID = 21;
  ID = 12;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 0.0
  0.0 1.0 0.0 0.55
  0.0 0.0 1.0 0.1;
  LOCAL_DIRECTIONS = GLOBAL;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  OWNER_LINK = -1;
  FE_NODE_NO = -1;
  NDOFS = 6;
}

REV_JOINT
{
  BASE_ID = 22;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 0.0
  0.0 6.123233995736766E-17 1.0 0.0
  0.0 -1.0 6.123233995736766E-17 0.0;
  HAS_Z_TRANS_DOF = false;
  ID = 5;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  MASTER_TRIAD = 13;
  MOVE_MASTER_TRIAD_ALONG = false;
  MOVE_SLAVE_TRIAD_ALONG = false;
  ROT_FORMULATION = FOLLOWER_AXIS;
  ROT_SEQUENCE = ZYX;
  ROT_SPRING_CPL = NONE;
  SLAVE_TRIAD = 7;
}

TRAN_SPRING_CPL = NONE;
VAR_QUADRANTS = 0 0 0;
Z_ROT_STATUS = FREE;
Z_TRANS_STATUS = FREE;
}

TRIAD
{
  BASE_ID = 23;
  ID = 13;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 1.25
  0.0 1.0 0.0 0.5
  0.0 0.0 1.0 1.0;
  LOCAL_DIRECTIONS = GLOBAL;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  OWNER_LINK = -1;
  FE_NODE_NO = -1;
  NDOFS = 6;
}

REV_JOINT
{
  BASE_ID = 24;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 0.0
  0.0 6.123233995736766E-17 -1.0 0.0
  0.0 1.0 6.123233995736766E-17 0.0;
  HAS_Z_TRANS_DOF = false;
  ID = 6;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  MASTER_TRIAD = 14;
  MOVE_MASTER_TRIAD_ALONG = false;
  MOVE_SLAVE_TRIAD_ALONG = false;
  ROT_FORMULATION = FOLLOWER_AXIS;
  ROT_SEQUENCE = ZYX;
  ROT_SPRING_CPL = NONE;
  SLAVE_TRIAD = 8;
  TRAN_SPRING_CPL = NONE;
  VAR_QUADRANTS = 0 0 0;
  Z_ROT_STATUS = FREE;
  Z_TRANS_STATUS = FREE;
}

TRIAD
{
  BASE_ID = 25;
  ID = 14;
  COORDINATE_SYSTEM =
  1.0 0.0 0.0 1.25
  0.0 1.0 0.0 -0.5
  0.0 0.0 1.0 1.0;
  LOCAL_DIRECTIONS = GLOBAL;
  LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
  OWNER_LINK = -1;
  FE_NODE_NO = -1;
  NDOFS = 6;
}

REV_JOINT
{
  BASE_ID = 26;
  COORDINATE_SYSTEM =

```

## APPENDIX D. EXAMPLE MODEL FILE

---

```

1.0 0.0 0.0 0.0
0.0 6.123233995736766E-17 1.0 0.0
0.0 -1.0 6.123233995736766E-17 0.0;
HAS_Z_TRANS_DOF = false;
ID = 7;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
MASTER_TRIAD = 15;
MOVE_MASTER_TRIAD_ALONG = false;
MOVE_SLAVE_TRIAD_ALONG = false;
ROT_FORMULATION = FOLLOWER_AXIS;
ROT_SEQUENCE = ZYX;
ROT_SPRING_CPL = NONE;
SLAVE_TRIAD = 9;
TRAN_SPRING_CPL = NONE;
VAR_QUADRANTS = 0 0 0;
Z_ROT_STATUS = FREE;
Z_TRANS_STATUS = FREE;
}

TRIAD
{
BASE_ID = 27;
ID = 15;
COORDINATE_SYSTEM =
1.0 0.0 0.0 1.5
0.0 1.0 0.0 0.5
0.0 0.0 1.0 -1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = -1;
FE_NODE_NO = -1;
NDOFS = 6;
}

REV_JOINT
{
BASE_ID = 28;
COORDINATE_SYSTEM =
1.0 0.0 0.0 0.0
0.0 6.123233995736766E-17 -1.0 0.0
0.0 1.0 6.123233995736766E-17 0.0;
HAS_Z_TRANS_DOF = false;
ID = 8;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
MASTER_TRIAD = 16;
MOVE_MASTER_TRIAD_ALONG = false;
MOVE_SLAVE_TRIAD_ALONG = false;
ROT_FORMULATION = FOLLOWER_AXIS;
ROT_SEQUENCE = ZYX;
ROT_SPRING_CPL = NONE;
SLAVE_TRIAD = 10;
TRAN_SPRING_CPL = NONE;
VAR_QUADRANTS = 0 0 0;
Z_ROT_STATUS = FREE;
Z_TRANS_STATUS = FREE;
}

TRIAD
{
BASE_ID = 29;
ID = 16;
COORDINATE_SYSTEM =
1.0 0.0 0.0 1.5
0.0 1.0 0.0 -1.0
0.0 0.0 1.0 -1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = 3;
FE_NODE_NO = 2101;
NDOFS = 6;
}

0.0 1.0 0.0 -1.0
0.0 0.0 1.0 -1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = -1;
FE_NODE_NO = -1;
NDOFS = 6;
}

TRIAD
{
BASE_ID = 30;
ID = 17;
COORDINATE_SYSTEM =
1.0 0.0 0.0 0.3697581332482862
0.0 1.0 0.0 0.0005969073302643464
0.0 0.0 1.0 -1.007591654794891;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = 3;
FE_NODE_NO = 2101;
NDOFS = 6;
}

TRIAD
{
BASE_ID = 31;
ID = 18;
COORDINATE_SYSTEM =
1.0 0.0 0.0 1.5
0.0 1.0 0.0 0.0
0.0 0.0 1.0 1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = -1;
FE_NODE_NO = -1;
NDOFS = 6;
}

AXIAL_SPRING
{
BASE_ID = 32;
ID = 1;
INIT_LENGTH = 0;
INIT_STIFFNESS = nil;
TRIAD_CONNECTIONS = 18 17;
USE_INIT_DEFLECTION = true;
}

TRIAD
{
BASE_ID = 33;
ID = 19;
COORDINATE_SYSTEM =
1.0 0.0 0.0 0.3697581332482862
0.0 1.0 0.0 0.0005969073302643464
0.0 0.0 1.0 -1.007591654794891;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = 3;
FE_NODE_NO = 2101;
NDOFS = 6;
}

```



APPENDIX D. EXAMPLE MODEL FILE

```
TRIAD
{
BASE_ID = 34;
ID = 20;
COORDINATE_SYSTEM =
1.0 0.0 0.0 1.5
0.0 1.0 0.0 0.0
0.0 0.0 1.0 1.0;
LOCAL_DIRECTIONS = GLOBAL;
LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;
OWNER_LINK = -1;
FE_NODE_NO = -1;
NDOFS = 6;
}

AXIAL_DAMPER
{
BASE_ID = 35;
ID = 2;
INIT_DAMPING = nil;
TRIAD_CONNECTIONS = 20 19;
}

LOAD
{
BASE_ID = 36;
ID = 1;
ENGINE = 1 FcENGINE;

INIT_LOAD = 0;
OWNER_TRIAD = 1;
LOAD_TYPE = 1;
SCALE_LOAD = 1;
FROM_OBJECT = -1 FcLINK;
FROM_POINT = 0 0 0;
TO_OBJECT = -1 FcLINK;
TO_POINT = 0.0 0.0 1.0;
}

ENGINE
{
BASE_ID = 37;
ID = 1;
MATH_FUNC = 1 FcfSCALE;
SENSOR = 1 FcTIME_SENSOR;
}

FUNC_SCALE
{
BASE_ID = 38;
FUNC_USE = GENERAL;
ID = 1;
SCALE = 30;
}

END {FEDEMMODELFILE}
```

# Appendix E

## Work Log

A work log was kept throughout the work with this thesis, both to document the development process and enforce that the schedule was followed. Additionally, the commit history from GitHub to illustrate the work done. The commits can be seen by visiting ['http://ntnu-ipm.github.io/KBE/'](http://ntnu-ipm.github.io/KBE/).

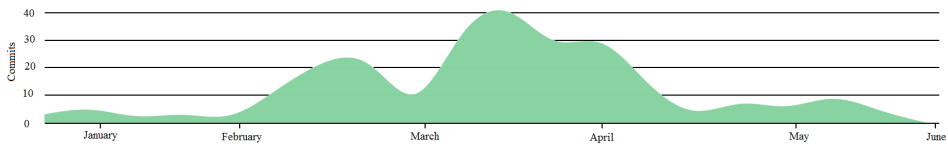


Figure E.1: GitHub commit history

| Date     | Hours | Main tasks  |
|----------|-------|---|
| 18-01-16 | 10    | Meeting with Sivertsen, downloaded FEDEM and created work log doc.                            |
| 19-01-16 | 14    | Troubleshooting errors with meshing and file export.  |
| 20-01-16 | 10    | Troubleshooting errors with meshing and file export.  |
| 21-01-16 | 12    | Troubleshooting errors with meshing and file export.  |
| 24-01-16 | 12    | Troubleshooting errors with meshing and file export.  |
| 25-01-16 | 12    | Meeting with Sivertsen regarding errors with meshing and file export, troubleshooting, FEDEM. |
| 26-01-16 | 12    | Working with a solution for reading and using design variables file in AML. FEDEM tutorials   |
| 27-01-16 | 10    | Working with a solution for reading and using design variables file in AML. FEDEM tutorials   |

| Date     | Hours | Main tasks   |
|----------|-------|--|
| 28-01-16 | 14    | Exploring design optimization in FEDEM, and creating a folder structure for mechanism versions in AML                        |
| 29-01-16 | 10    | Exploring design optimization in FEDEM, creating management of design variables in AML                                       |
| 30-01-16 | 8     | Exploring design optimization in FEDEM, creating management of design variables in AML                                       |
| 01-02-16 | 16    | Parametrization and integration of design variables in AML, researched different design optimization tools                   |
| 02-02-16 | 16    | Parametrization and integration of design variables in AML, researched design optimization in MATLAB                         |
| 03-02-16 | 16    | Meeting with Sivertsen, parametrization and integration of design variables in AML, researched design optimization in MATLAB |
| 04-02-16 | 8     | Parametrization and integration of design variables in AML   |
| 05-02-16 | 12    | Parametrization and integration of design variables in AML   |
| 08-02-16 | 18    | Troubleshooting solid mesh generation in AML, AMOpt testing  |
| 09-02-16 | 16    | Parametrization and integration of design variables in AML   |
| 10-02-16 | 17    | Meeting with Sivertsen, troubleshooting solid mesh AML, design optimization in MATLAB  |
| 11-02-16 | 15    | Troubleshooting solid mesh AML, design optimization in MATLAB + researched design optimization                               |
| 12-02-16 | 15    | Troubleshooting solid mesh AML, design optimization in MATLAB + researched design optimization                               |
| 15-02-16 | 16    | Generating solid mesh in AML, design optimization in MATLAB, scripting to FEDEM.   |
| 16-02-16 | 18    | Methods for controlling the solid mesh in AML  |
| 17-02-16 | 15    | Meeting with Sivertsen, exploring the usage of AMOpt, design optimization in MATLAB  |
| 18-02-16 | 16    | Exploring the usage of AMOpt, design optimization in MATLAB  |
| 19-02-16 | 15    | Meeting with Trier and Sivertsen, exploring the usage of AMOpt, design optimization in MATLAB                                |
| 22-02-16 | 15    | Exploring the usage of AMOpt, integrating FEDEM and AML  |
| 23-02-16 | 15    | Exploring the usage of AMOpt, integrating FEDEM and AML  |
| 24-02-16 | 16    | Meeting with Sivertsen, .bdf export and integrating FEDEM and AML  |
| 25-02-16 | 16    | Meeting with Haugen and Sivertsen, RBE2/3 and integrating FEDEM and AML  |
| 26-02-16 | 18    | RBE2 and integrating FEDEM and AML   |
| 29-02-16 | 16    | RBE2 and integrating FEDEM and AML   |
| 01-03-16 | 18    | RBE2 and integrating FEDEM and AML   |
| 02-03-16 | 16    | Meeting with Sivertsen, RBE2 and integrating FEDEM and AML   |
| 03-03-16 | 16    | RBE2 and integrating FEDEM and AML   |

APPENDIX E. WORK LOG

| Date     | Hours | Main tasks   |
|----------|-------|--|
| 04-03-16 | 18    | Integrating FEDEM and AML  |
| 07-03-16 | 16    | Integrating FEDEM and AML  |
| 08-03-16 | 18    | Integrating FEDEM and AML  |
| 09-03-16 | 16    | Meeting with Sivertsen, Integrating FEDEM and AML. Creating load case for four bar from AML to FEDEM       |
| 10-03-16 | 14    | Created rotation matrices for joints, investigating open constraint type issues                            |
| 11-03-16 | 14    | Integrating load-cases and control systems between AML and FEDEM   |
| 14-03-16 | 16    | Integrating load-cases and control systems between AML and FEDEM   |
| 15-03-16 | 17    | Modeling load cases and springs/dampers  |
| 16-03-16 | 17    | Meeting with Sivertsen, modeling load cases and springs/dampers  |
| 17-03-16 | 16    | Meeting with Sivertsen and Haugen, making the system more loosely coupled. Making input data more readable |
| 28-03-16 | 12    | Making the system more loosely coupled   |
| 29-03-16 | 14    | Making the system more loosely coupled. Commenting code  |
| 30-03-16 | 10    | Fixed cross-sections and link-twist, making the system more loosely coupled. Commenting code               |
| 31-03-16 | 18    | Removing workaround links. Fixed generic link twist  |
| 01-04-16 | 22    | Removed workaround links. Library updates  |
| 02-04-16 | 12    | Fixed surface between members of a link  |
| 03-04-16 | 14    | Loads, analysis in FEDEM - control system setup  |
| 04-04-16 | 18    | Loads, analysis in FEDEM, RBE for free/fixed constraints and line cross-sections                           |
| 05-04-16 | 18    | Meeting with Sivertsen, loads, analysis in FEDEM, RBE for free/fixed constraints and line cross-sections   |
| 06-04-16 | 20    | Writing spring/damper properties to FEDEM, creating spring/damper geometry                                 |
| 07-04-16 | 22    | Writing spring/damper and load properties to FEDEM, creating spring/damper geometry                        |
| 08-04-16 | 13    | Created the geometry for a log grabber mechanism   |
| 09-04-16 | 13    | Created the geometry for a 14-bar steering linkage, fixed drawing issue of member                          |
| 10-04-16 | 17    | Finished the log grabber and completed the initial setup of the steering mechanism                         |
| 11-04-16 | 17    | Preparing geometries for optimization, preparing analysis results in FEDEM for optimization                |
| 12-04-16 | 16    | Meeting with Sivertsen, Log grabber in FEDEM. Creating AML models for optimization parametrization         |
| 13-04-16 | 17    | Creating AML models for optimization parametrization   |
| 14-04-16 | 16    | Creating AML models for optimization parametrization. Setting up special load case                         |
| 15-04-16 | 16    | Creating AML models for optimization parametrization. Setting up special load case                         |

| <b>Date</b> | <b>Hours</b> | <b>Main tasks</b>   |
|-------------|--------------|---|
| 18-04-16    | 23           | Setting up special optimization case, setting up steering mechanism                         |
| 19-04-16    | 20           | Meeting with Sivertsen, setting up special optimization case, simulating steering mechanism |
| 20-04-16    | 22           | Setting up special optimization case, finished simulation of steering mechanism             |
| 21-04-16    | 22           | Setting up special optimization case, slider-crank mechanism                                |
| 22-04-16    | 17           | Setting up special optimization case, slider-crank mechanism                                |
| 25-04-16    | 17           | Report writing  |
| 26-04-16    | 18           | Meeting with Sivertsen, Report writing  |
| 27-04-16    | 18           | Report writing  |
| 28-04-16    | 20           | Report writing  |
| 29-04-16    | 17           | Report writing  |
| 02-05-16    | 17           | Report writing  |
| 03-05-16    | 20           | Report writing  |
| 04-05-16    | 20           | Meeting with Sivertsen, Report writing  |
| 05-05-16    | 20           | Report writing  |
| 06-05-16    | 18           | Report writing  |
| 07-05-16    | 18           | Report writing  |
| 08-05-16    | 17           | Report writing  |
| 09-05-16    | 17           | Report writing  |
| 10-05-16    | 17           | Report writing  |
| 11-05-16    | 18           | Meeting with Sivertsen, Report writing  |
| 12-05-16    | 22           | Report writing  |
| 13-05-16    | 20           | Report writing  |
| 14-05-16    | 19           | Report writing  |
| 15-05-16    | 19           | Report writing  |
| 16-05-16    | 21           | Report writing  |
| 18-05-16    | 20           | Report writing  |
| 19-05-16    | 20           | Implemented fillet, mesh refinement   |
| 20-05-16    | 24           | Implemented fillet, mesh refinement   |
| 21-05-16    | 16           | Report writing  |
| 22-05-16    | 17           | Report writing  |
| 23-05-16    | 19           | Report writing  |
| 24-05-16    | 21           | Report writing  |
| 25-05-16    | 21           | Report writing  |
| 26-05-16    | 21           | Report writing  |
| 27-05-16    | 20           | Report writing  |
| 28-05-16    | 20           | Report writing  |
| 29-05-16    | 19           | Report writing  |

*APPENDIX E. WORK LOG*

---

| <b>Date</b> | <b>Hours</b> | <b>Main tasks</b>                      |
|-------------|--------------|--|
| 30-05-16    | 20           | Report writing                         |
| 31-05-16    | 19           | Meeting with Sivertsen, Report writing |
| 01-06-16    | 20           | Report writing                         |
| 02-06-16    | 19           | Report writing                         |
| 03-06-16    | 19           | Report writing                         |
| 04-06-16    | 19           | Report writing                         |
| 05-06-16    | 17           | Report writing                         |
| 06-06-16    | 16           | Report writing                         |
| 07-06-16    | 18           | Report writing                         |
| 08-06-16    | 21           | Meeting with Sivertsen, report writing |
| 09-06-16    | 17           | Report writing                         |

# Appendix F

## Source Code

This appendix contains the source code for the entire mechanism system. The code added in this appendix has been specifically formatted in order to take up as little space as possible, thus it is as human-readable as in the delivered zip file.

The source code includes the following files (in subsequent compile order):

- System.def
- Data-models.aml
- Springs-dampers.aml
- Loads.aml
- Cross-sections.aml
- Optimizations.aml
- Constraints.aml
- Constraint-types.aml
- Meshing.aml
- Analysis.aml
- Link-member-geometry.aml

- Link-surface-geometry.aml
- Links.aml
- Collections.aml
- Geometry-export.aml

## F.1 System.def

```

=====
; System : :mechanism-system
; Purpose : AML Mechanism Model
;
;
; Authors : Anders Kristiansen, Eivind Kristoffersen, Rasmus Korvald Skaare
=====
(in-package :AML)
(defvar #MECHANISM-LIBRARY# "")
(setf #MECHANISM-LIBRARY# (logical-path :mechanism-system "library"))
(define-system :mechanism-system
  :files '(
    "data-models.aml"
    "springs-dampers.aml"
    "loads.aml"
    "cross-sections.aml"
    "optimizations.aml"
    "constraints.aml"
    "constraint-types.aml"
    "meshing.aml"
    "analysis.aml"
    "link-member-geometry.aml"
    "link-surface-geometry.aml"
    "links.aml"
    "collections.aml"
    "geometry-export.aml"))

```

## F.2 Data-models.aml



```

;=====
; Class: point-data-model
; Used for define position properties
;=====
(define-class point-data-model
  :inherit-from (point-object create-event)
  :properties (
    (coordinates :class 'editable-data-property-class
      label "Coordinates"
    )
    coord-ref (nth 1 ^coordinates)
    label nil
    id nil
    line-width 4
    color 'green
    property-objects-list (list
      (list (the superior coordinates self)
        '(apply-formula? t))
    ))
  )))
(define-method get-coordinates point-data-model ()
  !coordinates)
;=====
; END point-data-model definitions
;=====
;=====
; Class: vector-data-model
; Used for defining a direction vector
;=====
(define-class vector-data-model
  :inherit-from (vector-class)
  :properties (
    point-ref (default ^point-default)
    (point-default :class 'point-data-model
      coordinates '(0 0 0)
    )
    (direction :class 'editable-data-property-class
      label "Direction"
      formula (default)
    )
  )
  length 0.2

```

```

base-point (the coordinates (:from (the superior point-ref)))
property-objects-list (list
  (list (the coordinates self (:from (the superior point-ref)))
    '(apply-formula? t))
  (list (the superior direction self)
    '(apply-formula? t))
  )))
;=====
; END vector-data-model definitions
;=====
;=====  

; Class: frame-data-model
; Used for creating a frame, i.e. a coordinate system
;=====
(define-class frame-data-model
  :inherit-from (coordinate-system-class)
  :properties (
    ;;traverse to superior reference
    point-ref (default (the point-default (:from ^z-vector-ref)))
    z-vector-ref (default ^z-vector-default)
    (z-vector-default :class 'vector-data-model
      direction '(0 0 1)
    )
    x-vector-ref (default ^x-vector-default)
    (x-vector-default :class 'vector-data-model
      direction '(1 0 0)
    )
    vector-k (the direction (:from ^z-vector-ref))
    vector-i (the direction (:from ^x-vector-ref))
    origin (the coordinates (:from ^point-ref))
    vector-j (cross-product ^vector-k ^vector-i)
    length 0.1
    property-objects-list (list
      "Coordinates:"
      (list (the coordinates self (:from ^point-ref))
        '(apply-formula? t))
      "Z Axis:"
      (list (the direction self (:from ^z-vector-ref))
        '(apply-formula? t))
      "X Axis:"

```

```

        (list (the direction self (:from ^x-vector-ref))
              '(apply-formula? t))
      )
    )
  )
)

(define-method get-coordinates frame-data-model ()
  !origin)
;=====
; Should only be called on a spline-frame
;=====

(define-method get-direction-along-member frame-data-model ()
  !vector-i)
;=====
; END frame-data-model definitions
;=====
;=====  

; Class: sub-point-data-model
; Used for placing a constraint's sub-frame. Explicit values are set on instantiation
;=====

(define-class sub-point-data-model
  :inherit-from (point-data-model)
  :properties (
    reference-object ^main-frame
    coordinates '(0 0 0)
  )
)
;=====
; END sub-point-data-model definitions
;=====
;=====  

; Class: sub-frame-data-model
; Explicit sub-frame class
; When a sub-point-ref is specified in a joint element,
; its sub-frame will be placed accordingly
;=====

(define-class sub-frame-data-model
  :inherit-from (frame-data-model)
  :properties (
    (point-ref :class 'point-data-model
              coordinates (convert-coords ^sub-point-ref (the coordinates (:from ^sub-point-ref)))

```

```

    )
  )
)
;=====
; END sub-frame-data-model definitions
;=====

```

## E3 Springs-dampers.aml

```

;=====
; Class: general-spring-damper-class
; Superclass for spring-model-class and damper-model-class
; Contains information about spring/damper start and end points,
; as well as which link(s) they are connected to.
; Has any link mesh nodes that the link/damper is connected to, as subobjects
;=====
(define-class general-spring-damper-class
  :inherit-from (series-object)
  :properties (
    ;; 'nil' properties are set on instantiation
    label          nil
    start-point-data-model nil
    end-point-data-model nil
    incident-links  nil
    stiffness-damping nil
    type           nil
    start-point    (get-coordinates ^start-point-data-model)
    end-point      (get-coordinates ^end-point-data-model)
    ;; If the spring/damper is supposed to be connected to the ground and not to the mechanism,
    ;; a separate node has to be created instead of finding the closest node in the mesh
    quantity      (length ^incident-links)
    class-expression 'mesh-query-nodes-with-label-class
    init-form '(
      label          (format nil "closest-mesh-node--d" (1+ ^index))
      owner-link     (nth ^index ^incident-links)
      mesh-database-object (get-mesh-database (nth ^index ^incident-links))
      interface-object (if (= 0 ^index)
                          ^start-point-data-model
                          ^end-point-data-model)
    )
  )
)

```

```

subset-mesh-query-object-list (append
  (get-link-mesh-node-query-objects-list
    (get-mesh-model-object
      (nth ^index ^incident-links)))
    (get-rbe2-independent-node-list
      (get-mesh-model-object
        (nth ^index ^incident-links))))
  tolerance 1.0e3
  quantity 1
  color 'green
  line-width 5
)))

(define-method get-stiffness-damping general-spring-damper-class ()
  !stiffness-damping)

(define-method get-end-point general-spring-damper-class ()
  !end-point)

(define-method get-type general-spring-damper-class ()
  !type)

;=====
; END main-mechanism-class
;=====

;=====
; Class: spring-geometry-class
; Creates the visual representation for a spring in the mechanism system
;=====

(define-class spring-geometry-class
  :inherit-from (union-object)
  :properties (
    start-point (default '(0 0 0))
    end-point (default '(1 1 1))
    spring-direction (subtract-points ^end-point ^start-point)
    spring-length (points-distance ^start-mid-point ^end-mid-point)
    spring-radius (/ (points-distance ^start-point ^end-point) 10)
    start-mid-point (add-points ^start-point (multiply-vector-by-scalar ^spring-direction 0.25))
    end-mid-point (add-points ^start-point (multiply-vector-by-scalar ^spring-direction 0.75))
    x1-direction (normalize (arbitrary-normal-to-vector ^spring-direction))
    x2-direction (compute-plane-normal ^start-point ^end-point
      (add-points ^start-mid-point ^x1-direction))
    object-list (list ^coil ^start-line ^end-line ^start-spring ^end-spring)
    (coil :class 'curve-from-points-class

```

```

reference-coordinate-system ^spring-coordinate-system
points-coordinates-list (helical-curve ^^spring-radius ^^spring-length 6 :n 100)
(spring-coordinate-system :class 'coordinate-system-class
  origin ^^start-mid-point
  vector-i ^^x1-direction
  vector-j ^^x2-direction
  vector-k ^^spring-direction)
(start-line :class 'line-object
  point1 ^^start-point
  point2 ^^start-mid-point)
(end-line :class 'line-object
  point1 ^^end-mid-point
  point2 ^^end-point)
(start-spring :class 'line-object
  point1 ^^start-mid-point
  point2 (add-points ^^start-mid-point (multiply-vector-by-scalar ^^x1-direction
    ^^spring-radius))
)
(end-spring :class 'line-object
  point1 ^^end-mid-point
  point2 (add-points ^^end-mid-point (multiply-vector-by-scalar ^^x1-direction
    ^^spring-radius)))
color 'purple))
;=====
; END spring-geometry-class
;=====
;=====
; Class: damper-geometry-class
; Creates the visual representation for a damper in the mechanism system
;=====
(define-class damper-geometry-class
  :inherit-from (union-object)
  :properties (
    start-point (default '(0 0 0))
    end-point (default '(1 1 1))
    damper-direction (subtract-points ^end-point ^start-point)
    damper-width (/ (points-distance ^start-point ^end-point) 12)
    x1-direction (arbitrary-normal-to-vector ^damper-direction)
    x1-negative-direction (multiply-vector-by-scalar ^x1-direction -1)
    x2-direction (compute-plane-normal ^start-point ^end-point

```

```

    (add-points ^start-mid-point (multiply-vector-by-scalar ^x1-direction ^damper-width))
x2-negative-direction    (multiply-vector-by-scalar ^x2-direction -1)
cross-point    (add-points ^start-point (multiply-vector-by-scalar ^damper-direction 0.4))
start-mid-point (add-points ^start-point (multiply-vector-by-scalar ^damper-direction 0.25))
end-mid-point (add-points ^start-point (multiply-vector-by-scalar ^damper-direction 0.75))
object-list ( list ^start-mid-line ^end-mid-line ^lower-cross-line1 ^upper-cross-line1
    ^lower-cross-line2 ^upper-cross-line2 ^lower-box-line1 ^lower-box-line2
    ^lower-box-line3 ^lower-box-line4 ^upper-box-line1 ^upper-box-line2
    ^upper-box-line3 ^upper-box-line4 ^upper-cross-box-line1
    ^upper-cross-box-line2
    ^upper-cross-box-line3 ^upper-cross-box-line4 ^box-line1 ^box-line2
    ^box-line3 ^box-line4)
(start-mid-line :class 'line-object
  point1 ^^start-point
  point2 ^^start-mid-point)
(end-mid-line :class 'line-object
  point1 ^^cross-point
  point2 ^^end-point)
(lower-cross-line1 :class 'line-object
  point1 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x1-direction ^^damper-width))
  point2 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x1-negative-direction ^^damper-width)))
(lower-cross-line2 :class 'line-object
  point1 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x2-direction ^^damper-width))
  point2 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x2-negative-direction ^^damper-width)))
(upper-cross-line1 :class 'line-object
  point1 (add-points ^^cross-point (multiply-vector-by-scalar
    ^^x2-direction ^^damper-width))
  point2 (add-points ^^cross-point (multiply-vector-by-scalar
    ^^x2-negative-direction ^^damper-width)))
(upper-cross-line2 :class 'line-object
  point1 (add-points ^^cross-point (multiply-vector-by-scalar
    ^^x1-direction ^^damper-width))
  point2 (add-points ^^cross-point (multiply-vector-by-scalar
    ^^x1-negative-direction ^^damper-width)))
(lower-box-line1 :class 'line-object
  point1 (add-points ^^start-mid-point (multiply-vector-by-scalar

```

```

    ^^x1-direction ^^damper-width))
  point2 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x2-direction ^^damper-width)))
(lower-box-line2 :class 'line-object
  point1 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x2-direction ^^damper-width))
  point2 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x1-negative-direction ^^damper-width)))
(lower-box-line3 :class 'line-object
  point1 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x1-negative-direction ^^damper-width))
  point2 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x2-negative-direction ^^damper-width)))
(lower-box-line4 :class 'line-object
  point1 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x2-negative-direction ^^damper-width))
  point2 (add-points ^^start-mid-point (multiply-vector-by-scalar
    ^^x1-direction ^^damper-width)))
(upper-box-line1 :class 'line-object
  point1 (add-points ^^end-mid-point (multiply-vector-by-scalar
    ^^x1-direction ^^damper-width))
  point2 (add-points ^^end-mid-point (multiply-vector-by-scalar
    ^^x2-direction ^^damper-width)))
(upper-box-line2 :class 'line-object
  point1 (add-points ^^end-mid-point (multiply-vector-by-scalar
    ^^x2-direction ^^damper-width))
  point2 (add-points ^^end-mid-point (multiply-vector-by-scalar
    ^^x1-negative-direction ^^damper-width)))
(upper-box-line3 :class 'line-object
  point1 (add-points ^^end-mid-point (multiply-vector-by-scalar
    ^^x1-negative-direction ^^damper-width))
  point2 (add-points ^^end-mid-point (multiply-vector-by-scalar
    ^^x2-negative-direction ^^damper-width)))
(upper-box-line4 :class 'line-object
  point1 (add-points ^^end-mid-point (multiply-vector-by-scalar
    ^^x2-negative-direction ^^damper-width))
  point2 (add-points ^^end-mid-point (multiply-vector-by-scalar
    ^^x1-direction ^^damper-width)))
(upper-cross-box-line1 :class 'line-object
  point1 (add-points ^^cross-point (multiply-vector-by-scalar

```



```

    ^^x1-direction ^^damper-width))
  point2 (add-points ^^cross-point (multiply-vector-by-scalar
    ^^x2-direction ^^damper-width)))
  (upper-cross-box-line2 :class 'line-object
    point1 (add-points ^^cross-point (multiply-vector-by-scalar
      ^^x2-direction ^^damper-width))
    point2 (add-points ^^cross-point (multiply-vector-by-scalar
      ^^x1-negative-direction ^^damper-width)))
  (upper-cross-box-line3 :class 'line-object
    point1 (add-points ^^cross-point (multiply-vector-by-scalar
      ^^x1-negative-direction ^^damper-width))
    point2 (add-points ^^cross-point (multiply-vector-by-scalar
      ^^x2-negative-direction ^^damper-width)))
  (upper-cross-box-line4 :class 'line-object
    point1 (add-points ^^cross-point (multiply-vector-by-scalar
      ^^x2-negative-direction ^^damper-width))
    point2 (add-points ^^cross-point (multiply-vector-by-scalar
      ^^x1-direction ^^damper-width)))
  (box-line1 :class 'line-object
    point1 (add-points ^^start-mid-point (multiply-vector-by-scalar
      ^^x1-direction ^^damper-width))
    point2 (add-points ^^end-mid-point (multiply-vector-by-scalar
      ^^x1-direction ^^damper-width)))
  (box-line2 :class 'line-object
    point1 (add-points ^^start-mid-point (multiply-vector-by-scalar
      ^^x2-direction ^^damper-width))
    point2 (add-points ^^end-mid-point (multiply-vector-by-scalar
      ^^x2-direction ^^damper-width)))
  (box-line3 :class 'line-object
    point1 (add-points ^^start-mid-point (multiply-vector-by-scalar
      ^^x1-negative-direction ^^damper-width))
    point2 (add-points ^^end-mid-point (multiply-vector-by-scalar
      ^^x1-negative-direction ^^damper-width)))
  (box-line4 :class 'line-object
    point1 (add-points ^^start-mid-point (multiply-vector-by-scalar
      ^^x2-negative-direction ^^damper-width))
    point2 (add-points ^^end-mid-point (multiply-vector-by-scalar
      ^^x2-negative-direction ^^damper-width)))
  color 'red ))
  (defun get-mid-point (p1 p2)

```

```

    (list (/ (+ (nth 0 p1) (nth 0 p2)) 2) (/ (+ (nth 1 p1) (nth 1 p2)) 2) (/ (+ (nth 2 p1) (nth 2 p2)) 2)))
;=====
; END damper-geometry-class
;=====
;=====
; Class: damper-model
; Specifies the damper implementation of a general-spring-damper-class
;=====
(define-class damper-model
  :inherit-from (general-spring-damper-class damper-geometry-class)
  :properties (
    type 'damper
    damping (get-stiffness-damping !superior)))
(define-method get-damping damper-model ()
  !damping)
;=====
; END damper-model
;=====
;=====
; Class: spring-model
; Specifies the spring implementation of a general-spring-damper-class
;=====
(define-class spring-model
  :inherit-from (general-spring-damper-class spring-geometry-class)
  :properties (
    type 'spring
    stiffness (get-stiffness-damping !superior)))
(define-method get-stiffness spring-model ()
  !stiffness)
;=====
; END spring-model
;=====
;=====
; Class: mesh-query-nodes-with-label-class
; Is a mesh-query-nodes-from-interface-class,
; but with additional information about which link mesh it is connected to
;=====
(define-class mesh-query-nodes-with-label-class
  :inherit-from (mesh-query-nodes-from-interface-class)
  :properties (

```

```

        label      nil
        owner-link nil))
(define-method get-owner-link mesh-query-nodes-with-label-class ()
  !owner-link)
;=====
; Returns the position this node has in the link mesh nodes query list
;=====
(define-method get-node-position mesh-query-nodes-with-label-class ()
  (first !mesh-entities-list))
;=====
; END mesh-query-nodes-with-label-class
;=====

```

## F4 Loads.aml

```

;=====
; Class: load-geometry-class
; Creates the geometry for a load
;=====
(define-class load-geometry-class
  :inherit-from (union-object)
  :properties (
    load-point nil
    mechanism-size nil
    type nil
    direction nil
    ;; This defines the line length, controlled by the size of the mechanism
    end-point (subtract-points ^load-point (multiply-vector-by-scalar
      (normalize ^direction)/( ^mechanism-size 6)))
    total-vector (subtract-points ^end-point ^load-point)
    translation (/ ^mechanism-size 40)
    random-direction-normal (normalize (arbitrary-normal-to-vector ^direction))
    negative-random (multiply-vector-by-scalar ^random-direction-normal -1)
    arrow-vector1
      (add-points
        (add-points ^load-point (multiply-vector-by-scalar
          ^total-vector 0.2)) ;; Small portion of the total line
        (multiply-vector-by-scalar ^random-direction-normal
          (/ ^mechanism-size 70)) ;; How far the arrow points
      )
      ;; move from the line
  )

```

```

arrow-vector2      (add-points
                   (add-points ^load-point (multiply-vector-by-scalar
                                           ^total-vector 0.2)) ;; This is the small portion
                                           ;; of the total line
                   (multiply-vector-by-scalar ^negative-random
                                           (/ ^mechanism-size 70))
                   )
object-list (case ^type
             ('force (list ^line ^arrow1 ^arrow2))
             ('torque (list ^line ^arrow1 ^arrow2 ^arrow3 ^arrow4))
             )
(line :class 'line-object
     point1 ^^load-point
     point2 ^^end-point
     )
(arrow1 :class 'line-object
       point1 ^^load-point
       point2 ^^arrow-vector1
       )
(arrow2 :class 'line-object
       point1 ^^load-point
       point2 ^^arrow-vector2
       )
(arrow3 :class 'line-object
       point1 ^^load-point
       point2 ^^arrow-vector1
       orientation (list (translate
                          (multiply-vector-by-scalar ^total-vector 0.1)
                          ))
       )
(arrow4 :class 'line-object
       point1 ^^load-point
       point2 ^^arrow-vector2
       orientation (list (translate
                          (multiply-vector-by-scalar ^total-vector 0.1)
                          ))
       )
color (case ^type
      ('force 'orange)
      ('torque 'red)

```

```

    )
orientation (list (translate
                  (multiply-vector-by-scalar (normalize ^direction)
                  (* -1 ^translation))))
line-width 1
))
;=====
; END load-geometry-class
;=====
;=====
; Class: load-model
; Holds load parameters
;=====
(define-class load-model
  :inherit-from (load-geometry-class)
  :properties (
    loaded-link (default nil) ;;link-model-class
    mechanism-size (default nil) ;;Greatest distance within the mechanism
    load-point-object (default nil) ;;point-data-model
    mesh-model-object (get-mesh-model-object ^loaded-link)
    (type :class 'option-property-class
      mode 'radio
      options-list (list 'force 'torque)
      formula (default 'torque)
      label "Load type"
    )
    (load-point :class 'editable-data-property-class
      formula (get-coordinates ^load-point-object)
      label "Load point"
    )
    (direction :class 'editable-data-property-class
      formula (default '(0 0 1))
      label "Load direction vector"
    )
    (magnitude :class 'var-unit-data-property-class
      current-load (case ^type
                    ('force 'N)
                    ('torque (progn (add-unit 'Nm 1.0 '(N m)) 'Nm))
                  )
      value-in (default ^magnitude)

```

```

unit-in ^current-load
unit    ^current-load
label   "Magnititude"
)
(scale-load :class 'var-unit-data-property-class
current-load (case ^type
              (' force 'N)
              (' torque (progn (add-unit 'Nm 1.0 '(N m)) 'Nm))
              )
value-in (default ^scale-load)
unit-in ^current-load
unit    ^current-load
label   "Scaling load"
)
property-objects-list (list
                      (list (the superior type self)
                            '(automatic-apply? t))
                      (list (the superior load-point self)
                            '(automatic-apply? t))
                      (if ^magnitude
                          (list (the superior magnitude self)
                                '(automatic-apply? t))
                          (list (the superior scale-load self)
                                '(automatic-apply? t))
                          )
                      (list (the superior direction self)
                            '(automatic-apply? t)))
:subobjects (
  (loaded-node :class 'mesh-query-nodes-with-label-class
  label       "loaded-node"
  owner-link  ^^loaded-link
  mesh-database-object (get-mesh-database ^^loaded-link)
  interface-object    ^^load-point-object
  subset-mesh-query-object-list (append (get-link-mesh-node-query-objects-list
                                         ^^mesh-model-object)
                                         (get-rbe2-independent-node-list
                                         ^^mesh-model-object))
  tolerance      1.0e3
  quantity       1
  color          'green

```

```

        line-width      5
      )
    ))
(define-method get-load-point load-model ()
  !load-point)
(define-method get-load-type load-model ()
  !type)
(define-method get-loaded-link load-model ()
  !loaded-link)
(define-method get-magnitude load-model ()
  (the magnitude value-in))
(define-method get-scale-load load-model ()
  (the scale-load value-in))
(define-method get-direction load-model ()
  !direction)
(define-method get-loaded-node load-model ()
  !loaded-node)
(define-method get-load-ID load-model ()
  (1+ !index))
;=====
; END load-model definitions
;=====

```

## E5 Cross-sections.aml

```

;=====
; Class: cross-section-model
; Superclass used for querying class-names
;=====
(define-class cross-section-model
  :inherit-from(tagging-object position-object)
  :properties (
    optimization-object (default nil)
    type                (default nil)
    area                (get-area !superior ^type)
    max-element-size 0.004
    tag-dimensions    '(1 2)
    tag-attributes    (list ^max-element-size .1
                          0 0.1 0 20.0 1.0e-5)
  )

```

```

    )
  )
(define-method get-width cross-section-model ()
  !width)
(define-method get-height cross-section-model ()
  !height)
(define-method get-area cross-section-model (type)
  (case type
    ('circular      (* pi (/ (expt !diameter 2) 4)))
    ('circular-tube (* pi 0.25 (- (expt !outer-diameter 2) (expt !inner-diameter 2))))
    ('rectangular   (* !width !height))
    ('rectangular-tube (- (* !width !height) (* !inner-width !inner-height)))
    ('line          nil)
    ('i-beam        (+ (* !width !flange-thickness 2) (* !height !web-thickness)))
    ('h-beam        (+ (* !height !flange-thickness 2) (* !width !web-thickness)))
    ('hexagonal     (* 2 (sqrt 3) !R)))
;=====
; END cross-section-model
;=====
;=====
; Class: circular-section
;=====
(define-class circular-section
  :inherit-from (imprint-class cross-section-model)
  :properties (
    type      'circular
    target-object ^disc
    tool-object-list (list ^p1)
    diameter    (average ^width ^height)
    (disc :class 'disc-object
      diameter ^^diameter
    )
    (p1 :class 'point-object
      coordinates (list (/ ^diameter 2) 0 0)
    ))
(define-method get-width circular-section ()
  !diameter
)
(define-method get-height circular-section ()
  !diameter

```



```

)
;=====
; END circular-section
;=====
;=====
; Class: circular-tube-section
;=====
(define-class circular-tube-section
  :inherit-from (imprint-class cross-section-model)
  :properties (
    type      'circular-tube
    target-object ^diff-object
    tool-object-list (list ^p1 ^p2)
    outer-diameter (average ^width ^height)
    thickness      (* 0.2 ^outer-diameter)
    inner-diameter (- ^outer-diameter ^thickness)
    (p1 :class 'point-object
      coordinates (list (/ ^outer-diameter 2) 0 0)
    )
    (p2 :class 'point-object
      coordinates (list (/ (- ^outer-diameter ^thickness) 2) 0 0)
    )
    (diff-object :class 'difference-object
      object-list (list ^outer-circular ^inner-circular)
      simplify? t
    )
    (outer-circular :class 'circular-section
      diameter ^outer-diameter
    )
    (inner-circular :class 'circular-section
      diameter ^^inner-diameter
    )))
(define-method get-width circular-tube-section ()
  !outer-diameter
)
(define-method get-height circular-tube-section ()
  !outer-diameter
)
;=====
; END circular-tube-section

```

```

;=====
;=====
; Class: rectangular-section
;=====
(define-class rectangular-section
  :inherit-from (imprint-class cross-section-model)
  :properties (
    type      'rectangular
    width     (default 0.04)
    height    (default 0.01)
    target-object ^sheet
    tool-object-list (list ^p1)
    (p1 :class 'point-object
      coordinates (list (/ ^width 2) 0 0)
    )
    (sheet :class 'sheet-object
      width ^^width
      height ^^height
    )))
(define-method get-width rectangular-section ()
  !width
)
(define-method get-height rectangular-section ()
  !height
)
;=====
; END rectangular-section
;=====
;=====
; Class: rectangular-tube-section
;=====
(define-class rectangular-tube-section
  :inherit-from (cross-section-model difference-object)
  :properties (
    type      'rectangular-tube
    width     (default 0.04)
    height    (default 0.01)
    inner-width (- ^^width ^thickness)
    inner-height (- ^^height ^thickness)
    object-list (list ^outer-rectangle ^inner-rectangle)
  )
)

```

```

    thickness (* 0.2 (average ^width ^height))
    simplify? t
    (outer-rectangle :class 'sheet-object
      width ^^width
      height ^^height
    )
    (inner-rectangle :class 'sheet-object
      width ^^inner-width
      height ^^inner-height
    )))
(define-method get-width rectangular-tube-section ()
  !width
)
(define-method get-height rectangular-tube-section ()
  !height
)
;=====
; END rectangular-tube-section
;=====
;=====  

; Class: line-section definitions
;=====
(define-class line-section
  :inherit-from (line-object cross-section-model)
  :properties (
    type 'line
    height (default 0.04)
    point1 (list 0 (- (/ ^height 2)) 0)
    point2 (list 0 (/ ^height 2) 0)
  )
)
(define-method get-width line-section ()
  !height
)
(define-method get-height line-section ()
  0
)
;=====
; END line-section definitions
;=====

```

```

;=====
; Class: H-beam-section
;=====
(define-class H-beam-section
  :inherit-from (union-object cross-section-model)
  :properties (
    type      'h-beam
    width     (if ^optimization-object
                  (get-width ^optimization-object)
                  (default 0.04)
                )
    height    (if ^optimization-object
                  (get-height ^optimization-object)
                  (default 0.04)
                )
    flange-thickness (if ^optimization-object
                          (get-flange ^optimization-object)
                          (* 0.1 ^height)
                        )
    web-thickness (if ^optimization-object
                      (get-web ^optimization-object)
                      (* 0.1 ^width)
                    )
    object-list (list ^top-flange ^web ^bottom-flange)
    simplify?   t
    (top-flange :class 'sheet-object
                width  ^^width
                height ^^flange-thickness
                orientation (list
                              (translate (list 0 (half ^^height) 0))
                            )
                )
    (web :class 'sheet-object
         width  ^^web-thickness
         height ^^height
         )
    (bottom-flange :class 'sheet-object
                   width  ^^width
                   height ^^flange-thickness
                   orientation (list
                                (translate (list 0 (half ^^height) 0))
                              )
                   )
  )

```

```

                (translate (list 0 (- (half ^^height) 0))
                )))
(define-method get-width H-beam-section ()
  !height)
(define-method get-height H-beam-section ()
  (+ !width !flange-thickness))
;=====
; END H-beam-section definitions
;=====
;=====
; Class: I-beam-section
;=====
(define-class I-beam-section
  :inherit-from (union-object cross-section-model)
  :properties (
    type      'I-beam
    width     (if ^optimization-object
                 (get-width ^optimization-object)
                 (default 0.04)
              )
    height    (if ^optimization-object
                 (get-height ^optimization-object)
                 (default 0.04)
              )
    flange-thickness (if ^optimization-object
                        (get-flange ^optimization-object)
                        (* 0.1 ^width)
                      )
    web-thickness (if ^optimization-object
                    (get-web ^optimization-object)
                    (* 0.1 ^height)
                  )
    second-moment-of-area (calculate-second-moment-of-area-about-z-axis !superior)
    object-list      (list ^left-flange ^web ^right-flange)
    simplify?       t
    (right-flange :class 'sheet-object
                 width  ^^flange-thickness
                 height ^^width
                 orientation (list
                              (translate (list (half (+ ^^height ^^flange-thickness) 0 0))

```

```

    )
  )
  (web :class 'sheet-object
    width ^^height
    height ^^web-thickness
  )
  (left-flange :class 'sheet-object
    width ^^flange-thickness
    height ^^width
    orientation (list
      (translate (list (- (half (+ ^^height ^^flange-thickness))) 0 0))
    )))
(define-method get-width I-beam-section ()
  (+ !flange-thickness !height))
(define-method get-height I-beam-section ()
  !width)
(define-method calculate-second-moment-of-area-about-z-axis I-beam-section ()
  (+
    (+ (/ (* !width (expt !flange-thickness 3) 6)
      (* 0.5 !flange-thickness !width (expt !height 2)
      (* (expt !flange-thickness 2) !width !height)
      (* 0.5 (expt !flange-thickness 3) !width))
    (/ (* !web-thickness (expt !height 3) 12)))
(define-method get-second-moment-of-area-about-z-axis I-beam-section ()
  !second-moment-of-area)
;=====
; END I-beam-section definitions
;=====
;=====
; Class: hexagonal-section
;=====
(define-class hexagonal-section
  :inherit-from (imprint-class cross-section-model)
  :properties (
    type      'hexagonal
    target-object ^poly
    tool-object-list (list ^p1)
    R          (/ (average ^width ^height) 2)
    (poly :class 'polygon-object
      vertices (list

```

```

      (list ^R 0 0)
      (list (/ ^R 2) (- (/ (* ^R (sqrt 3)) 2) 0)
      (list (- (/ ^R 2)) (- (/ (* ^R (sqrt 3)) 2) 0)
      (list (- ^R) 0 0)
      (list (- (/ ^R 2)) (/ (* ^R (sqrt 3)) 2) 0)
      (list (/ ^R 2) (/ (* ^R (sqrt 3)) 2) 0)
    )
  dimension 2
)
(p1 :class 'point-object
  coordinates (list 0 (/ (* ^R (sqrt 3)) 2) 0)
)))
(define-method get-width hexagonal-section ()
  (* 2 !R)
)
(define-method get-height hexagonal-section ()
  (* !R (sqrt 3))
)
;=====
; END hexagonal-section definitions
;=====

```

## F.6 Optimizations.aml

```

;=====
; Class: design-variable-class
; Holding the design variable value, as well as min and max values for it
;=====
(define-class design-variable-class
  :inherit-from (data-model-node-mixin)
  :properties (
    (dv-value :class 'editable-data-property-class
      label (format nil "~:~(-a~) initial value" (object-name ^superior))
    )
    (min-value :class 'editable-data-property-class
      formula (default (* 0.5 ^dv-value))
      label (format nil "~:~(-a~) minimum value" (object-name ^superior))
    )
    (max-value :class 'editable-data-property-class

```

```

        formula (default (* 2 ^dv-value))
        label (format nil "~:(~a~) maximum value" (object-name ^superior))
    )
))

(define-method get-dv-value design-variable-class ()
  !dv-value)

(define-method get-min-value design-variable-class ()
  !min-value)

(define-method get-max-value design-variable-class ()
  !max-value)

;=====
; END design-variable-class definitions
;=====

;=====
; Class: constraint-class
; Holds properties needed for a constraint. constraint-value is the
; actual constraint value to be evaluated. Limit is the right side value of
; the constraint equation. inequality-type is the relation between value and
; limit. For example, if value = -2, inequality-type = "<=" and limit = 0, means: -2 <= 0
; Penalty-weight-factor and penalty-power are penalty functions for the optimization,
; giving a penalty if the constraint inequality is not satisfied
;=====

(define-class constraint-class
  :inherit-from (object)
  :properties (
    type          (default nil)
    constraint-value (default nil)
    limit         (default 0)
    inequality-type (default (nth 0 (list :lt :lt eq :gt :gteq :eq))) ;; Default less than
    penalty-weight-factor (default 10)
    penalty-power   (default 2)))

(define-method get-constraint-evaluation constraint-class (&optional constraint-value)
  (let (
    (val (if constraint-value constraint-value !constraint-value))
  )
    (case !inequality-type
      (:lt (< val !limit))
      (:lt eq (<= val !limit))
      (:gt (> val !limit))
      (:gteq (>= val !limit))
    )
  )

```



```

      (:eq (= val !limit))))
;=====
; END constraint-class definitions
;=====
;=====
; Class: general-optimization-model
; Superclass for all optimization classes
;=====
(define-class general-optimization-model
  :inherit-from (object)
  :properties (
    type (default nil)
    affected-links (default nil)
    init-values (default nil)
    constraint-type (default nil)
    max-allowed-deformation (default nil)
    load-objects (default nil)
    links (default nil) ;;link-model-classes
    current-path (default nil) ;;Directory path to the current mechanism
    main-mech-ref (default nil)
    label (default nil)
    materials-list (get-materials-list !superior))
(define-method get-affected-links general-optimization-model ()
  !affected-links)
(define-method get-loads-on-optimization-links general-optimization-model ()
  (loop for load in !load-objects
    if (position (get-loaded-link load) !links)
      collect load))
(define-method get-materials-list general-optimization-model ()
  (loop for link in !links
    append (list link (get-material-type link))))
;=====
; END general-optimization-model definitions
;=====
;=====
; Class: cross-section-optimization-model
; Class for dealing with the optimization of one or more cross sections in the mechanism.
; The class and its methods assume that all cross sections to be optimized are equal.
;
;=====

```

```

(define-class cross-section-optimization-model
  :inherit-from (general-optimization-model)
  :properties (
    type      'cross-section
    cs-types  (remove nil (get-cs-types !superior))
    ;;Design variables
    (width :class 'design-variable-class
      dv-value (nth 0 ^^init-values)
    )
    (height :class 'design-variable-class
      dv-value (nth 1 ^^init-values)
    )
    (flange-thickness :class 'design-variable-class
      dv-value (default (* 0.1 (get-dv-value ^^width)))
    )
    (web-thickness :class 'design-variable-class
      dv-value (default (* 0.1 (get-dv-value ^^height)))
    )
    ;;Constraint
    (constraint-object :class 'constraint-class
      type      ^constraint-type
      constraint-value (get-constraint-value ^superior)
    )
    ;;Objective function
    area      (handle-results-and-get-area !superior)
    ;;FEDEM results
    fedem-model-file (format nil "~a\\model.fmm" ^current-path)
    results-file-path (format nil "~a\\deformation.asc" ^current-path)
    ;;Used to call (get-area ...) on a cross-section-model
    (dummy-cs :class 'I-beam-section
      width      (get-dv-value ^^width)
      height     (get-dv-value ^^height)
      flange-thickness (get-dv-value ^^flange-thickness)
      web-thickness (get-dv-value ^^web-thickness)
    )
    iteration-results-list nil ;;Holds the results for each iteration
    ;;Holds the best result from each generation,
    ;;including the deformation value from the FEDEM simulation
    best-result-from-each-gen-list nil
    counter      0
  )

```

```

first-run?          t
generation-counter  0
final-result        nil
parent-exploration-object (get-exploration-object !superior)
multiga-exploration-object (get-multiga-object ^parent-exploration-object)
;; Can only set optimization values if the exploration-object exists
property-objects-list (if (not ^parent-exploration-object)
  (list
    '("Load AMOpt" (button1-parameters :load-amopt)
      ui-work-area-action-button-class)
    '("Add optimization exploration object"
      (button1-parameters :add-exploration-object)
      ui-work-area-action-button-class))
  (list
    (list (the superior width dv-value self)
      '(automatic-apply? t))
    (list (the superior width min-value self)
      '(automatic-apply? t))
    (list (the superior width max-value self)
      '(automatic-apply? t))
    ""
    (list (the superior height dv-value self)
      '(automatic-apply? t))
    (list (the superior height min-value self)
      '(automatic-apply? t))
    (list (the superior height max-value self)
      '(automatic-apply? t))
    ""
    (list (the superior flange-thickness dv-value self)
      '(automatic-apply? t))
    (list (the superior flange-thickness min-value self)
      '(automatic-apply? t))
    (list (the superior flange-thickness max-value self)
      '(automatic-apply? t))
    ""
    (list (the superior web-thickness dv-value self)
      '(automatic-apply? t))
    (list (the superior web-thickness min-value self)
      '(automatic-apply? t))
    (list (the superior web-thickness max-value self)
      '(automatic-apply? t))
  ))

```

```

      '(automatic-apply? t)
      ""
      '( "Set optimization variables"
        (button1-parameters :init-optimization-variables)
        ui-work-area-action-button-class))))
;=====
; Returns the largest deformation of the mechanism simulated in FEDEM.
; Will initiate a new simulation if it's the first run or if it's the
; beginning of a new generation. For a new generation simulation, the best
; mechanism design from the previous generation will be simulated
;=====
(define-method run-fedem-simulation cross-section-optimization-model ()
  (delete-all-previous-fmm-files (the))
  (reset-all-database-values !main-mech-ref)
  (draw-sewn-with-tri-mesh !main-mech-ref)
  (write-nastran-bdf-files !main-mech-ref)
  (write-fmm-model-file !main-mech-ref :from-opt? t)
  (run-program (format nil "fedem -f ~a -solve dynamics" !fedem-model-file)
    (run-program (format nil "fedem_graphexp -curvePlotFile ~a -curvePlotType 5 -modelfile ~a"
      !results-file-path !fedem-model-file))
  (undraw !main-mech-ref)
  (save-best-result-in-generation (the)))
;=====
; Deletes the files FEDEM use for reduction of the different parts,
; as well as any simulation files. Requires that XEmacs runs in administrator mode,
; and that the fmm file is not open in FEDEM.
;=====
(define-method delete-all-previous-fmm-files cross-section-optimization-model ()
  (when (probe-file !results-file-path)
    (delete-file !results-file-path))
  (when (probe-file (format nil "~a\\model_RDB" !current-path))
    (tsi ::leave-directory-empty (format nil "~a\\model_RDB" !current-path))
    (delete-directory (format nil "~a\\model_RDB" !current-path)))
  (when (probe-file (format nil "~a.bak" !fedem-model-file))
    (delete-file (format nil "~a.bak" !fedem-model-file)))
  (when (probe-file !fedem-model-file)
    (delete-file !fedem-model-file)))
(define-method save-best-result-in-generation cross-section-optimization-model ()
  (change-value !best-result-from-each-gen-list
    (append !best-result-from-each-gen-list

```

```

    (list (list (list 'area (get-area !dummy-cs 'I-beam)) (list 'width (get-dv-value !width))
      (list 'height (get-dv-value !height)) (list 'flange (get-dv-value !flange-thickness))
      (list 'web (get-dv-value !web-thickness)) (list 'constraint (get-constraint-value (the)))
      (list 'deformation (get-largest-deformation (the))))))
;=====
; This method returns the cross section area, which is the objective function
; of the optimization. Before it returns the area, however, the current state of
; the optimization is saved. It assumes the use of the multi-objective genetic
; algorithm (multiga), and as such, at the end of each generation, will mesh the
; best results in that generation, and run a FEDEM simulation on it.
;=====
(define-method handle-results-and-get-area cross-section-optimization-model ()
  (when !first-run?
    (reset-temporary-optimization-values (the))
    (change-value !first-run? nil)
    (run-fedem-simulation (the)))
  (add-iteration-result (the))
  (change-value !counter (1+ !counter))
  ;;At the end of each generation
  (when (= (the population-size (:from !multiga-exploration-object)) !counter)
    (set-best-dv-values (the))
    (run-fedem-simulation (the))
    (change-value !counter 0)
    (change-value !iteration-results-list nil)
    (change-value !generation-counter (1+ !generation-counter))
    (when (= !generation-counter
      (the number-of-generations (:from !multiga-exploration-object))) ;;End of optimization
      (set-and-draw-final-result (the))))
  (get-area !dummy-cs 'I-beam))
(define-method get-constraint-value cross-section-optimization-model ()
  (case (the constraint-object type)
    ('stress (get-stress-constraint-value (the)))
    ('strain (get-strain-constraint-value (the))))
;=====
; Normalized stress constraint for the max stress in the loaded links.
; g(x) = (sigma(x) / sigma_yield) - 1
;=====
(define-method get-stress-constraint-value cross-section-optimization-model ()
  (loop for load in (get-loads-on-optimization-links (the))
    maximize (- (/ (calculate-max-stress-in-cross-section (the))

```

```

      (case (get-load-type load)
        ('torque :torque)
        ('force :force)
      )
      (if (get-scale-load load)
        (get-scale-load load)
        (get-magnitude load)))
      (get-yield-strength (get-material-type (get-loaded-link load)))) 1)
    into max-stress
  finally (return max-stress))
;=====
; To be implemented?
;=====
(define-method get-strain-constraint-value cross-section-optimization-model()
  nil)
;=====
; sigma = (M_torque / I_z) * y_max
;=====
(define-method calculate-max-stress-in-cross-section cross-section-optimization-model (&key
  torque force)
  (if torque
    (* (/ torque                ;;M_torque
        (get-second-moment-of-area-about-z-axis !dummy-cs) ;;I_z
        (+ (/ (get-height (the)) 2) (get-flange (the))) ;;y_max
        )))
  ))
(define-method set-best-dv-values cross-section-optimization-model ()
  (let (
    (best-result (get-best-result (the)))
  )
    (progn
      (change-value (the width dv-value) (nth 1 (nth 1 best-result)))
      (change-value (the height dv-value) (nth 1 (nth 2 best-result)))
      (change-value (the flange-thickness dv-value) (nth 1 (nth 3 best-result)))
      (change-value (the web-thickness dv-value) (nth 1 (nth 4 best-result))))))
;=====
; Function that will continue calling itself until the results from FEDEM are ready
;=====
(defun wait-for-fedem-results (results-file-path)
  (unless (probe-file results-file-path)
    (sleep 1)

```

```

(wait-for-fedem-results results-file-path))
;=====
; For each change in the design variables, this method is called, adding the current state
; to the iteration-results-list object
;=====
(define-method add-iteration-result cross-section-optimization-model ()
  (change-value !iteration-results-list
    (append !iteration-results-list
      (list (list (list 'area (get-area !dummy-cs 'I-beam)) (list 'width (get-dv-value !width))
        (list 'height (get-dv-value !height)) (list 'flange (get-dv-value !flange-thickness))
        (list 'web (get-dv-value !web-thickness)) (list 'constraint (get-constraint-value (the))))))
      )))
;=====
; Method retrieving the best result from the iteration-results-list
; Disregarding all results where the constraint is not held
;=====
(define-method get-best-result cross-section-optimization-model ()
  (let (
    (min-area 1.0e10)
    (best-result nil)
  )
    (progn
      (loop for result in !iteration-results-list do
        (if (and (< (nth 1 (nth 0 result)) min-area)
          (get-constraint-evaluation !constraint-object (nth 1 (nth 5 result))))
          (progn
            (setf min-area (nth 1 (nth 0 result)))
            (setf best-result result))))
        best-result ;;return value
      )))
  (define-method get-final-result cross-section-optimization-model ()
    (let (
      (min-area 1.0e10)
      (final-result nil)
    )
      (progn
        (loop for result in !best-result-from-each-gen-list do
          (if !max-allowed-deformation
            (when (and (< (nth 1 (nth 0 result)) min-area)
              (< (nth 1 (nth 6 result)) !max-allowed-deformation))

```

```

        (setf min-area (nth 1 (nth 0 result)))
        (setf final-result result)
    ;; If no max allowed deformation exists, it should not be evaluated.
    (when (< (nth 1 (nth 0 result)) min-area)
        (setf min-area (nth 1 (nth 0 result)))
        (setf final-result result)))
    final-result ;;return value
)))

;=====
; Reads the result from fedem and stores all deformation entries in a list
;=====

(define-method read-results-from-file cross-section-optimization-model ()
  (if (probe-file !results-file-path)
      (with-open-file (file !results-file-path :direction :input)
          (loop for line = (read-line file nil :eof)
                until (equal line :eof)
                for ls = (string-to-delimited-token-list line :delimiter #\tab :string-token? nil)
                if (and (numberp (nth 0 ls)) (numberp (nth 1 ls)))
                    append (list (nth 1 ls))))
      (progn
        (message (format nil "~a" is not a valid file path." !results-file-path) :append? t)
        nil ;;Return value if the file does not exist
      )))

;=====
; Retrieves the largest deformation from the fedem results list
;=====

(define-method get-largest-deformation cross-section-optimization-model ()
  (loop for deformation in (read-results-from-file (the))
        maximize deformation into max-deformation
        finally (return max-deformation)))

(define-method set-and-draw-final-result cross-section-optimization-model ()
  (change-value !first-run? t)
  (change-value !final-result (get-final-result (the)))
  (if !final-result
      (progn
        (change-value (the width dv-value) (nth 1 (nth 1 !final-result)))
        (change-value (the height dv-value) (nth 1 (nth 2 !final-result)))
        (change-value (the flange-thickness dv-value) (nth 1 (nth 3 !final-result)))
        (change-value (the web-thickness dv-value) (nth 1 (nth 4 !final-result)))
        (draw-sewn-wo-mesh !main-mech-ref)

```



```

(message-box "Final result!"
  (format nil "Flange Width: ~d~%Web Height: ~d~%Flange Thickness: ~d~%Web Thickness:
    ~d~%Cross Section Area: ~d~%Max Deformation: ~d~%"
    (the width dv-value) (the height dv-value) (the flange-thickness dv-value)
    (the web-thickness dv-value) (nth 1 (nth 0 !final-result)) (nth 1 (nth 6 !final-result))
    :mode :ok :width 400 :height 300)
)
(progn
  (change-value (the width dv-value) (nth 0 !init-values))
  (change-value (the height dv-value) (nth 1 !init-values))
  (change-value (the flange-thickness dv-value) (* 0.1 (nth 0 !init-values)))
  (change-value (the web-thickness dv-value) (* 0.1 (nth 1 !init-values)))
  (draw-sewn-wo-mesh !main-mech-ref)
  (message-box "No result!"
    (format nil "None of the tested cross sections fulfilled the criterias for max stress and-%
      deformation. Change the init values and try again." :mode :ok :width 400 :height 300))
  )))
(define-method reset-temporary-optimization-values cross-section-optimization-model ()
  (change-value !generation-counter 0)
  (change-value !iteration-results-list nil)
  (change-value !best-result-from-each-gen-list nil))
;=====
; Initializes the design variable values and the objective function value,
; from the values in the GUI
;=====
(define-method init-optimization-variables cross-section-optimization-model ()
  (add-object (the continuous-variables (:from !parent-exploration-object))
    'height
    'dcms-design-variable-continuous-class
    :init-form '(
      design-property-object (the superior superior superior height dv-value (:eval? nil))
      initial-value (the dv-value (:from ^^height))
      max-value (the max-value (:from ^^height))
      min-value (the min-value (:from ^^height))
    ))
  (add-object (the continuous-variables (:from !parent-exploration-object))
    'width
    'dcms-design-variable-continuous-class
    :init-form '(
      design-property-object (the superior superior superior width dv-value (:eval? nil))

```

```

        initial-value      (the dv-value (:from ^^width))
        max-value         (the max-value (:from ^^width))
        min-value         (the min-value (:from ^^width))
    ))
(add-object (the continuous-variables (:from !parent-exploration-object))
'flange-thickness
'dcms-design-variable-continuous-class
:init-form '(
    design-property-object (the superior superior superior
                            flange-thickness dv-value (:eval? nil))
    initial-value          (the dv-value (:from ^^flange-thickness))
    max-value              (the max-value (:from ^^flange-thickness))
    min-value              (the min-value (:from ^^flange-thickness))
    ))
(add-object (the continuous-variables (:from !parent-exploration-object))
'web-thickness
'dcms-design-variable-continuous-class
:init-form '(
    design-property-object (the superior superior superior
                            web-thickness dv-value (:eval? nil))
    initial-value          (the dv-value (:from ^^web-thickness))
    max-value              (the max-value (:from ^^web-thickness))
    min-value              (the min-value (:from ^^web-thickness))
    ))
(add-object (the constraints (:from !parent-exploration-object))
'stress
'dcms-constraint-class
:init-form '(
    left-side-property-object (the superior superior superior constraint-object
                                constraint-value (:error nil :eval? nil))
    constraint-inequality-type (the superior superior superior constraint-object
                                inequality-type)
    penalty-power              (the superior superior superior constraint-object
                                penalty-power)
    penalty-weight-factor      (the superior superior superior constraint-object
                                penalty-weight-factor)
    right-side-value           (the superior superior superior constraint-object limit)
    ))
(add-object (the objectives (:from !parent-exploration-object))
'area

```

```

'dcms-objective-class
:init-form '(
  design-property-object (the superior superior superior area (:eval? nil))
  minimize?      t
  )))
;=====
; Left-click button actions for cross-section-optimization-model
;=====
(define-method work-area-button1-action cross-section-optimization-model (params)
  (case params
    (:init-optimization-variables
     (init-optimization-variables (the))
     )
    (:load-amopt
     (load-module "amopt")
     (amopt-global-set-configuration :main-form (get-data-model-main-form))
     (display-amopt-toolbar-form)
     )
    (:add-exploration-object
     (add-object (the)
       'cross-section-optimization-manager-object
       'dcms-exploration-object-manager-class)
     )))
;=====
; Get methods below
;=====
(define-method get-width cross-section-optimization-model ()
  (get-dv-value !width))
(define-method get-height cross-section-optimization-model ()
  (get-dv-value !height))
(define-method get-flange cross-section-optimization-model ()
  (get-dv-value !flange-thickness))
(define-method get-web cross-section-optimization-model ()
  (get-dv-value !web-thickness))
(define-method get-cs-types cross-section-optimization-model ()
  (loop for link in !affected-links
    append (loop for shape in (get-shapes-on-link (nth link !links))
      collect (get-cs-type shape)
    )))
;=====

```

```

; Used for retrieving the exploration-object when an AMOPT model has been added
; to the main-mechanism-class
;=====
(define-method get-exploration-object cross-section-optimization-model ()
  (first (children (the) :class 'dcms-exploration-object-manager-class)))
;=====
; Used for retrieving the multiga-object
;=====
(define-method get-multiga-object dcms-exploration-object-manager-class ()
  (first (children (the) :class 'dcms-exploration-multiga-optimization-class)))
;=====
; END cross-section-optimization-model definitions
;=====

```

## E.7 Constraints.aml

```

;=====
; Class: master-joint-model
; Used for holding male and female instantiations as subobjects
;=====
(define-class master-joint-model
  :inherit-from (series-object frame-data-model)
  :properties (
    link-incidence (default nil)
    max-element-size (default 0.04)
    min-element-size (default 0.01)
    scale-factor (default 1.2) ;;Scaling of the constraint dimensions
    constraint-variable (default nil)
    degrees-of-freedom (default nil) ;;Only applicable for free joints
    incident-links (default nil) ;;Ref to the link-model-class objects
    constraint-type (default nil)
    gender-list (remove nil (append (list (if (nth 0 ^link-incidence) "male" nil))
                                     (list (if (nth 1 ^link-incidence) "female" nil))))
    joint-elements (when (children (the superior)) (children (the superior)))
    male-joint-element (loop for element in ^joint-elements
                            when (equal 'male (get-gender element)) do
                              (return element))
    female-joint-element (loop for element in ^joint-elements
                              when (equal 'female (get-gender element)) do

```

```

                (return element)
point-ref      (default nil)
direction     (default nil)
(z-vector-ref :class 'vector-data-model
              direction ^^direction
              )
(x-vector-ref :class 'vector-data-model
              direction (arbitrary-normal-to-vector ^^direction)
              )
dimensions    (calculate-dimensions (the superior))
(max-width :class 'editable-data-property-class
            label "Joint width"
            formula (nth 0 ^dimensions)
            )
(max-height :class 'editable-data-property-class
            label "Joint height"
            formula (nth 1 ^dimensions)
            )
quantity      (length ^gender-list)
class-expression '(read-from-string (concatenate !constraint-type "-"
                                                (nth !index ^gender-list) "-element"))
init-form '(
            master-joint-object (the superior superior) ;;Reference to the master-joint-model
            gender              (read-from-string (nth !index ^gender-list))
            incident-link       (nth !index (remove nil ^incident-links))
            label                (concatenate (nth !index ^gender-list) "-joint-element")
            )
;;Only display properties for non-free constraints
property-objects-list (if (or (equal "free" ^constraint-type) (equal "fixed" ^constraint-type))
                          (list (format nil "This is a -a constraint. No properties can be edited." ^constraint-type))
                          (list
                            ("Draw joint" (button1-parameters :draw-joint)
                                     ui-work-area-action-button-class)
                            ""
                            ("Undraw" (button1-parameters :undraw)
                                     ui-work-area-action-button-class)
                            (the superior max-width self)
                            (the superior max-height self)
                            )))
)

```

```

(define-method get-constraint-type master-joint-model ()
  !constraint-type
(define-method get-incident-links master-joint-model ()
  !incident-links
;=====
; Calculates the dimensions by retrieving the biggest width and height
; from the incident members. Dimensions are scaled with the scale-factor
;=====
(define-method calculate-dimensions master-joint-model ()
  (let (
    (max-w 0)
    (max-h 0)
  )
    (loop for joint-elem in !joint-elements do
      (loop for link in !incident-links do
        (loop for member in (get-members (get-link-geometry link)) do
          (let (
            (dimensions (get-max-dimensions member joint-elem))
          )
            (progn
              (if (> (nth 0 dimensions) max-w)
                (setf max-w (nth 0 dimensions)))
              (if (> (nth 1 dimensions) max-h)
                (setf max-h (nth 1 dimensions)))
            )))
          finally (return (list (* max-w !scale-factor) (* max-h !scale-factor)))
        ))
  ))
(define-method get-female-element master-joint-model ()
  !female-joint-element
(define-method get-male-element master-joint-model ()
  !male-joint-element
(define-method get-joint-direction-vector master-joint-model ()
  !direction
(define-method get-main-frame-coords master-joint-model (&optional params)
  (if params
    (nth params !origin)
  !origin
  )
(define-method get-direction master-joint-model ()
  !direction

```

```

(define-method get-degrees-of-freedom master-joint-model ()
  !degrees-of-freedom
(define-method is-free-constraint master-joint-model ()
  (equal "free" !constraint-type))
(define-method get-link-incidence master-joint-model ()
  !link-incidence
(define-method get-constraint-incidence master-joint-model (link)
  (let (
    (pos (position link !incident-links)))
    (when pos
      (nth pos (children (the :class 'joint-element-model))
        )))
;=====
; Left-click button methods for master-joint-model
;=====
(define-method work-area-button1-action master-joint-model (params)
  (case params
    (:draw-joint
      (when !female-joint-element
        (draw !female-joint-element :draw-subobjects? nil))
      (when !male-joint-element
        (draw !male-joint-element :draw-subobjects? nil)))
    (:undraw
      (undraw self)))
;=====
; END master-joint-model definitions
;=====
; Class: joint-element-model
; Superclass for male and female joint elements
;=====
(define-class joint-element-model
  :inherit-from (frame-data-model)
  :properties (
    label nil
    ;; union-list is the list used for a union with the incident link.
    ;; different-list is the list of objects that are to be subtracted from the final geometry.
    ;; See link-geometry-class in links.aml for the use
    union-list nil
    difference-list nil

```

```

master-joint-object nil
incident-link nil ;;link-model-class
constraint-type (get-constraint-type ^master-joint-object)
direction (get-direction ^master-joint-object)
(max-element-size :class 'editable-data-property-class
  formula (when (and ^max-width ^max-height)
    (/ (min ^max-width ^max-height) 8)
  )
  label "Mesh element size"
)
min-element-size (when (and ^max-width ^max-height)
  (/ (min ^max-width ^max-height) 16)
)
display? t
gender nil
gender_int (case ^gender ('male 0) ('female 1))
link-incidence (nth ^gender_int (get-link-incidence ^master-joint-object))
link-mesh-model-object (get-mesh-model-object ^incident-link)
members-connected-to-joint-element (loop for member in (get-members
  (get-link-geometry ^incident-link)
  when (position !superior (get-joints-on-member
    member))
  collect member
)
(sub-point-ref :class 'sub-point-data-model
)
(rbe2-independent-node :class 'mesh-node-class
  coordinates (get-rbe2-independent-node-coordinates ^superior)
  mesh-object (get-surface-mesh (get-mesh-model-object ^^incident-link))
  color 'red
  line-width 3
)
property-objects-list (list
  ("Draw joint element geometry" (button1-parameters :draw-joint)
    ui-work-area-action-button-class)
  ""
  ("Draw RBE2 nodes" (button1-parameters :draw-nodes)
    ui-work-area-action-button-class)
  ""
  ("Undraw" (button1-parameters :undraw)

```



```

        ui-work-area-action-button-class)
    ""
    (list (the superior max-element-size self)
          '(automatic-apply? t)))
:subjects (
  (main-frame :class 'frame-data-model
; Inherited frame properties used in main-frame and sub-frame
  (x-vector-ref :class 'vector-data-model
    direction (let (
      (first-sweep (nth 0 ^members-connected-to-joint-element))
      (x-dir (subtract-vectors
        (the coordinates (:from (the point-ref (:from (nth 0
          (get-joints-on-member first-sweep)
          ))))
        (the coordinates (:from (the point-ref (:from (nth 1
          (get-joints-on-member first-sweep)
          ))))
          ))
      )
      (x-dir-normal (cross-product ^^direction
        (cross-product ^^direction x-dir)))
      )
      (if (equal 0 (vector-length x-dir-normal))
        x-dir-normal
        (arbitrary-normal-to-vector ^^direction))))))
; An element's sub-frame is placed by specifying the sub-point-ref in the element's properties
  (sub-frame :class 'sub-frame-data-model)
(define-method get-constraint-type joint-element-model ()
  !constraint-type)
(define-method get-incident-link joint-element-model ()
  !incident-link)
(define-method get-gender joint-element-model ()
  !gender)
(define-method get-max-height joint-element-model ()
  (when !max-height !max-height))
(define-method get-main-frame joint-element-model ()
  !main-frame)
(define-method get-sub-frame joint-element-model ()
  !sub-frame)
(define-method get-rbe2-independent-node joint-element-model ()
  !rbe2-independent-node)

```

```

(define-method get-rbe2-dependent-nodes joint-element-model ()
  !rbe2-dependent-nodes
(define-method is-free-constraint joint-element-model ()
  (equal "free" !constraint-type))
(define-method get-joint-union-list joint-element-model ()
  !union-list
(define-method get-joint-difference-list joint-element-model ()
  !difference-list
(define-method get-main-frame-coords joint-element-model (&optional params)
  (if params
    (nth params (the main-frame origin))
    (the main-frame origin))
(define-method get-sub-frame-coords joint-element-model (&optional params)
  (if params
    (nth params (the sub-frame origin))
    (the sub-frame origin))
(define-method get-displayed-members-connected-to-joint-element joint-element-model ()
  (loop for member in !members-connected-to-joint-element
    if (is-displayed? member)
      collect member)
(define-method get-rbe2-independent-node-coordinates joint-element-model ()
  (if (or (equal "free" (get-constraint-type !master-joint-object))
    (equal "fixed" (get-constraint-type !master-joint-object)))
    (get-coordinates (get-sub-frame (the)))
    (get-coordinates (get-main-frame (the))))
(define-method get-constraint-incidence joint-element-model (link-index)
  (when (member link-index ^link-incidence) (the self)))
(define-method work-area-button1-action joint-element-model (params)
  (case params
    (:draw-joint
      (draw self))
    (:draw-nodes
      (draw !rbe2-independent-node)
      (draw !rbe2-dependent-nodes))
    (:undraw
      (undraw self)
      (undraw !rbe2-independent-node)
      (undraw !rbe2-dependent-nodes)))
;=====
; END joint-element-model definitions

```

```
=====
```

## F.8 Constraint-types.aml

```
=====
```

```
; Class: free-constraint-class
; An free constraint is a joint with no geometry
; Used for leaving the end of a link free
```

```
=====
```

```
(define-class free-constraint-class
  :inherit-from (master-joint-model)
  :properties ())
```

```
=====
```

```
; END free-constraint-class definitions
```

```
=====
```

```
=====
```

```
; Class: free-male-element
; Female element of an free constraint type.
; As this element is free, it has no geometry
```

```
=====
```

```
(define-class free-male-element
  :inherit-from (joint-element-model)
  :properties (
    ;; Retrieve all nodes from the cross-section from the end/start of a link member
    (rbe2-dependent-nodes :class 'mesh-query-nodes-from-interface-plane-class
      mesh-database-object (get-mesh-database ^^link-mesh-model-object)
      point-in-plane-coords (get-rbe2-independent-node-coordinates ^superior)
      plane-normal-vector (get-direction-along-member
        (get-spline-frame-at-joint-element (first
          (get-displayed-members-connected-to-joint-element
            ^superior)) ^superior))
      tolerance 1.0e-3 ;;Max allowable distance from plane to nodes
      quantity nil ;;Get all nodes in the plane
      subset-mesh-query-object-list (get-link-surface-mesh-elements-query-objects-list
        ^^link-mesh-model-object)
      color 'green
      line-width 3)))
```

```
=====
```

```
; END free-male-element definitions
```

```

;=====
;=====
; Class: free-female-element
; Female element of a free constraint type.
; As this element is free, it has no geometry
;=====
(define-class free-female-element
  :inherit-from (joint-element-model)
  :properties (
    ;; A free female joint could initially be placed at the wrong place,
    ;; depending on the topology of the mechanism. Its sub-frame should
    ;; then be placed accordingly. The local z coordinate is translated,
    ;; as that's the direction from the main-frame to the sub-frame.
    z-translation (get-translational-z-value (the superior))
    (sub-point-ref :class 'sub-point-data-model
      orientation (list
        (translate (list 0 0 ^z-translation))))
    ;; Retrieve all nodes from the cross-section from the end/start of a link member
    (rbe2-dependent-nodes :class 'mesh-query-nodes-from-interface-plane-class
      mesh-database-object (get-mesh-database ^^link-mesh-model-object)
      point-in-plane-coords (get-rbe2-independent-node-coordinates ^superior)
      plane-normal-vector (get-direction-along-member
        (get-spline-frame-at-joint-element
          (first
            (get-displayed-members-connected-to-joint-element
              ^superior)) ^superior))
      tolerance 1.0e-3 ;;Max allowable distance from plane to nodes
      quantity nil ;;Get all nodes in the plane
      subset-mesh-query-object-list (get-link-surface-mesh-elements-query-objects-list
        ^^link-mesh-model-object)
      color 'green
      line-width 3)))
;=====
; Method finding the translational value for an free/fixed female element
; Finds the member where the free joint is, then gets the other joint
; on that member and moves the free joint's sub-frame to match the other's
;=====
(define-method get-translational-z-value joint-element-model ()
  (let (
    (displayed-member

```

```

      (loop for member in !members-connected-to-joint-element
        when (the display? (:from member)) do
          (return member)))
    )
  (loop for joint in (get-joints-on-member displayed-member)
    when (not (eq (the) joint)) do
      (return (-(get-max-height joint))))))

;=====
; END free-female-element definitions
;=====
;=====
; Class: rigid-constraint-class
;=====
(define-class rigid-constraint-class
  :inherit-from (master-joint-model)
  :properties ())

;=====
; END rigid-constraint-class definitions
;=====
;=====
; Class: helical-constraint-class
;=====
(define-class helical-constraint-class
  :inherit-from (master-joint-model)
  :properties ())

;=====
; END helical-constraint-class definitions
;=====
;=====
; Class: prismatic-constraint-class
;=====
(define-class prismatic-constraint-class
  :inherit-from (master-joint-model)
  :properties ())

;=====
; END prismatic-constraint-class definitions
;=====
;=====
; Class: cylindric-constraint-class
;=====

```

```

(define-class cylindric-constraint-class
  :inherit-from (master-joint-model)
  :properties ())
;=====
; END cylindric-constraint-class definitions
;=====
; Class: planar-constraint-class
;=====
(define-class planar-constraint-class
  :inherit-from (master-joint-model)
  :properties ())
;=====
; END planar-constraint-class definitions
;=====
; Class: revolute-constraint-class
; Holding revolute male and female elements on instantiation
;=====
(define-class revolute-constraint-class
  :inherit-from (master-joint-model)
  :properties ())
;=====
; END revolute-constraint-class definitions
;=====
; Class: revolute-male-element
; Defines geometry for the revolute male element
;=====
(define-class revolute-male-element
  :inherit-from (union-object joint-element-model)
  :properties (
    main-frame-object (get-main-frame (the superior))
    union-list (when (and (plusp ^max-width) (plusp ^max-height))
      (list ^imprinted-pin ^eye))
    object-list (list ^imprinted-pin ^eye)
    simplify? nil
    mating-nodes-distance (/ ^^max-width 2)
    ;; Imprinted objects are objects that will receive surface nodes, used for meshing
    (imprinted-pin :class '(tagging-object geometry-with-split-periodic-faces-class)

```

```

source-object ^^pin
tag-dimensions '(1 2 3)
tag-attributes (list ^^max-element-size ^^min-element-size
                  0 0.1 0 20.0 1.0e-5)

(pin :class 'cylinder-object
  reference-object ^main-frame-object
  height (* 2 ^^max-height ^^scale-factor)
  diameter (/ ^^max-width 2)
  orientation (list
                (translate (list 0 0 (- (/ ^height 4))) )
                ))

(eye :class '(tagging-object cylinder-object)
  tag-dimensions '(1 2 3)
  tag-attributes (list ^^max-element-size ^^min-element-size
                    0 0.1 0 20.0 1.0e-5)
  reference-object ^sub-frame
  height ^^max-height
  diameter ^^max-width
  )

(sub-point-ref :class 'sub-point-data-model
  orientation (list
                (translate (list 0 0 (/ (- ^^max-height 1)))
                ))

(rbe2-dependent-nodes :class 'mesh-query-nodes-from-interface-class
  mesh-database-object (get-mesh-database ^^link-mesh-model-object)
  interface-object      (the point-ref (:from ^^main-frame-object))
  tolerance              (sqrt (+ (expt (/ (the height (:from ^^pin)) 4) 2)
                                   (expt (/ (the diameter (:from ^^pin)) 2) 2)))
  quantity               nil ;;Collect all nodes within the tolerance
  subset-mesh-query-object-list (get-link-surface-mesh-elements-query-objects-list
                                   ^^link-mesh-model-object)
  color                  'green
  line-width             3)))

;=====
; END revolute-male-element definitions
;=====

;=====
; Class: revolute-female-element
; Defines geometry for the revolute female element
;=====

```

```

(define-class revolute-female-element
  :inherit-from (difference-object joint-element-model)
  :properties(
    main-frame-object (get-main-frame (the superior))
    union-list (when (and (plusp ^max-width) (plusp ^max-height))
      (list ^fork)
    )
    difference-list (when (and (plusp ^max-width) (plusp ^max-height))
      (list ^imprinted-pin-hole))
    object-list (list ^fork ^imprinted-pin-hole)
    (imprinted-pin-hole :class '(tagging-object geometry-with-split-periodic-faces-class)
      source-object ^^pin-hole
      tag-dimensions '(1 2 3)
      tag-attributes (list ^^max-element-size ^^min-element-size
        0 0.1 0 20.0 1.0e-5))
    (pin-hole :class 'cylinder-object
      reference-object ^^main-frame-object
      height ^^max-height
      diameter (/ ^^max-width 2))
    (fork :class '(tagging-object cylinder-object)
      tag-dimensions '(1 2 3)
      tag-attributes (list ^^max-element-size ^^min-element-size
        0 0.1 0 20.0 1.0e-5)
      reference-object ^sub-frame
      height ^^max-height
      diameter ^^max-width)
    (sub-point-ref :class 'sub-point-data-model)
    (rbe2-dependent-nodes :class 'mesh-nodes-query-class
      tagged-object-list (list ^imprinted-pin-hole)
    mesh-object (get-surface-mesh (get-mesh-model-object
      (get-incident-link ^superior)))
      color 'green
      line-width 3)))
;=====
; END revolute-female-element definitions
;=====
;=====
; Class: ball-constraint-class
; Holding ball male and female elements on instantiation
;=====

```



```

(define-class ball-constraint-class
  :inherit-from (master-joint-model)
  :properties ())
;=====
; END ball-constraint-class definitions
;=====
; Class: ball-male-element
; Defines geometry for the ball male element
;=====
(define-class ball-male-element
  :inherit-from (tagging-object blend-class joint-element-model)
  :properties (
    main-frame-object (get-main-frame (the superior))
    tag-dimensions '(1 2 3)
    tag-attributes (list (/ ^^max-element-size 4) ^^min-element-size
                        0 0.1 0 20.0 1.0e-5)
    union-list (when (and (plusp ^max-width) (plusp ^max-height))
                (list !superior))
    (imprinted-stud :class '(tagging-object geometry-with-split-periodic-faces-class)
                    source-object ^^stud
                    tag-dimensions '(1 2 3)
                    tag-attributes (list ^^max-element-size ^^min-element-size
                                          0 0.1 0 20.0 1.0e-5))
    (stud :class 'cylinder-object
          reference-object ^sub-frame
          height (vector-length (subtract-vectors (the origin (:from ^sub-frame))
                                                    (the origin (:from ^main-frame-object))))
          diameter (/ (the diameter (:from ^ball)) 2)
          orientation (list
                      (translate (list 0 0 (/ ^height 2))))))
    (imprinted-ball :class '(tagging-object geometry-with-split-periodic-faces-class)
                    source-object ^^ball
                    tag-dimensions '(1 2 3)
                    tag-attributes (list ^^max-element-size ^^min-element-size
                                          0 0.1 0 20.0 1.0e-5))
    (ball :class 'sphere-object
          reference-object ^main-frame-object
          diameter (* 3 (/ ^^max-width 4)))
    (plate :class '(tagging-object cylinder-object)

```

```

tag-dimensions '(1 2 3)
tag-attributes (list ^^max-element-size ^^min-element-size
                  0 0.1 0 20.0 1.0e-5)
reference-object ^^sub-frame
diameter ^^max-width
height ^^max-height)
(sub-point-ref :class 'sub-point-data-model
orientation (list
              (translate (list 0 0 (- 0 (/ ^^max-height 2)) (* 1 (the diameter
                                                                    (:from ^^ball)))))))))
(rbe2-dependent-nodes :class 'mesh-nodes-query-class
tagged-object-list (list ^^imprinted-ball)
mesh-object      (get-surface-mesh (get-mesh-model-object
                                    (get-incident-link ^superior)))
color            'green
line-width      3)
;Properties used for blend-class:
(union-element :class 'union-object
  object-list (list ^imprinted-stud ^imprinted-ball ^plate)
)
all-edges      (vgl::k-sub-geoms (the geom (:from ^union-element)) 1)
source-object ^union-element
edge-ids       (loop for edge in ^all-edges
                    when (and (vgl::intersect-geom-p edge (the geom (:from ^plate)) edge)
                              (vgl::intersect-geom-p edge (the geom (:from ^stud)) edge))
                    collect edge)
radii          (list (/ ^^max-width 5)))
;=====
; END ball-male-element definitions
;=====
;=====
; Class: ball-female-element
; Defines geometry for the ball female element
;=====
(define-class ball-female-element
  :inherit-from (difference-object joint-element-model)
  :properties (
    joint-variable nil
    union-list (when (and (plusp ^^max-width) (plusp ^^max-height))
                  (list ^imprinted-socket)

```

```

    )
difference-list (when (and (plusp ^max-width) (plusp ^max-height))
  (list ^imprinted-hole)
)
object-list (list ^imprinted-socket ^imprinted-hole)
(imprinted-hole :class '(tagging-object geometry-with-split-periodic-faces-class)
  source-object ^^hole
  tag-dimensions '(1 2 3)
  tag-attributes (list ^^max-element-size ^^min-element-size
    0 0.1 0 20.0 1.0e-5)
)
(hole :class 'sphere-object
  reference-object ^sub-frame
  diameter (* 3 (/ ^max-width 4)))
(imprinted-socket :class '(tagging-object geometry-with-split-periodic-faces-class)
  source-object ^^socket
  tag-dimensions '(1 2 3)
  tag-attributes (list ^^max-element-size ^^min-element-size
    0 0.1 0 20.0 1.0e-5)
)
(socket :class 'intersection-object
  object-list (list ^sphere ^cyl)
  reference-object ^sub-frame)
(sphere :class 'sphere-object
  diameter ^^max-width)
(cyl :class 'cylinder-object
  height ^^max-height
  diameter ^^max-width)
orientation (list
  (rotate
    (angle-between-2-vectors ^direction ^constraint-variable)
    (cross-product ^direction ^constraint-variable)
    :axis-point (the coordinates (:from ^point-ref))))
(rbe2-dependent-nodes :class 'mesh-nodes-query-class
  tagged-object-list (list ^^imprinted-hole)
  mesh-object (get-surface-mesh (get-mesh-model-object
    (get-incident-link ^superior)))
  color 'green
  line-width 3
  )))

```

```

;=====
; END ball-female-element definitions
;=====
;=====
; Class: knuckle-constraint-class
; Holding knuckle male and female elements on instantiation
;=====
(define-class knuckle-constraint-class
  :inherit-from (master-joint-model)
  :properties ())
;=====
; END knuckle-constraint-class definitions
;=====
;=====
; Class: knuckle-female-element
; Defines geometry for the knuckle female element
;=====
(define-class knuckle-female-element
  :inherit-from (difference-object joint-element-model)
  :properties (
    main-frame-object (get-main-frame (the superior))
    inner-radius (/ ^max-width (expt ^scale-factor 2))
    outer-radius ^max-width
    union-list (when (and (plusp ^max-width) (plusp ^max-height))
      (list !superior))
    difference-list (when (and (plusp ^max-width) (plusp ^max-height))
      (list ^imprinted-pin-hole))
    object-list (list ^eye ^imprinted-pin-hole)
    (imprinted-pin-hole :class '(tagging-object geometry-with-split-periodic-faces-class)
      source-object ^^pin-hole
      tag-dimensions '(1 2 3)
      tag-attributes (list ^^max-element-size ^^min-element-size
        0 0.1 0 20.0 1.0e-5)
    )
    (pin-hole :class 'cylinder-object
      reference-object ^main-frame-object
      height ^^max-height
      diameter ^^inner-radius
    )
    (eye :class '(tagging-object cylinder-object)

```

```

tag-dimensions '(1 2 3)
tag-attributes (list ^^max-element-size ^^min-element-size
                0 0.1 0 20.0 1.0e-5)
reference-object ^sub-frame
height ^^max-height
diameter ^^outer-radius)
(sub-point-ref :class 'sub-point-data-model)
(rbe2-dependent-nodes :class 'mesh-nodes-query-class
  tagged-object-list (list ^^imprinted-pin-hole)
  mesh-object (get-surface-mesh (get-mesh-model-object
                                (get-incident-link ^superior)))
  color 'green
  line-width 3
  )))
;=====
; END knuckle-female-element definitions
;=====
;=====
; Class: knuckle-male-element
; Defines geometry for the knuckle male element
;=====
(define-class knuckle-male-element
  :inherit-from (union-object joint-element-model)
  :properties (
    main-frame-object (get-main-frame (the superior))
    ;; Helping variables
    outer-radius (* ^max-width ^scale-factor)
    inner-radius (* ^outer-radius 0.6)
    ;; Lists sent to link-geometry-class =====>
    union-list (when (and (plusp ^max-width) (plusp ^max-height))
                (list !superior))
    difference-list (when (and (plusp ^max-width) (plusp ^max-height))
                     (list ^subtracted-box))
    ;; <=====
    object-list (list ^pin ^fork)
    simplify? t
    (imprinted-pin :class '(tagging-object geometry-with-split-periodic-faces-class)
      source-object ^^pin
      tag-dimensions '(1 2 3)
      tag-attributes (list ^^max-element-size ^^min-element-size

```

```

0 0.1 0 20.0 1.0e-5)
;; Objects used to create the male joint geometry =====>
(fork :class 'difference-object
  object-list (list ^^box ^^subtracted-box ^^trim-object ^^sphere)
)
(subtracted-box :class 'box-object
  reference-object ^main-frame-object
  width (* ^^outer-radius 1.5)
  height ^^outer-radius
  depth ^^max-width
  orientation (list
    (translate (list (- (* 0.75 ^^outer-radius) ^^inner-radius) 0 0)))
)
(box :class 'box-object
  reference-object ^main-frame-object
  width (* ^^outer-radius 2)
  height ^^outer-radius
  depth (* ^^outer-radius 1.5)
  orientation (list
    (translate (list (- ^^outer-radius ^^inner-radius) 0 0)))
)
(pin :class 'cylinder-object
  reference-object ^main-frame-object
  diameter ^^inner-radius
  height (* ^^outer-radius 2))
(trim-box :class 'box-object
  reference-object ^main-frame-object
  width ^^inner-radius
  height ^^outer-radius
  depth (* ^^outer-radius 1.5)
  orientation (list
    (translate (list (* ^^inner-radius -0.5) 0 0)))
)
(trim-cyl :class 'cylinder-object
  reference-object ^main-frame-object
  diameter ^^outer-radius
  height (* ^^outer-radius 1.5))
(trim-object :class 'difference-object
  object-list (list ^^trim-box ^^trim-cyl))
(difference-sphere :class 'sphere-object
  reference-object ^main-frame-object
  diameter (* ^^outer-radius 3))
(trim-sphere :class 'sphere-object

```

```

reference-object ^main-frame-object
diameter (* ^^outer-radius 4)
(sphere :class 'difference-object
  object-list (list ^^trim-sphere ^^difference-sphere))
;;; <=====
;;; Used as the difference-list in link-geometry-class ==>
(difference-box :class 'difference-object
  object-list(list ^^subtracted-box ^^eye)
)
(eye :class '(cylinder-object)
  reference-object ^main-frame-object
  height ^^max-height
  diameter ^^outer-radius)
;;; <=====
;;; Defines where the sub-frame should be placed
(sub-point-ref :class 'sub-point-data-model
  orientation
    (list (translate (list (- (* ^^outer-radius 2) ^^inner-radius) 0 0)))
  (rbe2-dependent-nodes :class 'mesh-nodes-query-class
    tagged-object-list (list ^^imprinted-pin)
    mesh-object (get-surface-mesh (get-mesh-model-object
      (get-incident-link ^superior)))
    color 'green
    line-width 3
  )))
;=====
; END knuckle-male-element definitions
;=====

```

## E.9 Meshing.aml

```

;=====
; Class: link-mesh-class
; Class responsible for meshing
;=====
(define-class link-mesh-class
  :inherit-from (object)
  :properties (
    link-model          nil ;link-model-class

```

```

    geometry-model-object nil ;;link-geometry-class
    joint-elements        nil ;;joint-element-models
    mesh-object           (the (:from (the surface-tri-mesh)))
    rbe2-independent-node-list (loop for element in ^joint-elements
                                     collect (get-rbe2-independent-node element)
    )
    (node-set :class 'analysis-node-set-class
      query-objects-list (append (if (get-export-surface ^link-model)
                                   (list (the nodes-query (:from ^^mesh-object)))
                                   (list (the nodes-query (:from ^^tet-mesh))))
                                ^rbe2-independent-node-list
      )
    ) ;;Contains all the nodes used for the bdf file
  )
  :subobjects (
    (mesh-database :class 'mesh-database-class
      )
    (surface-tri-mesh :class 'paver-mesh-class
      object-to-mesh      ^^geometry-model-object
      mesh-database-object ^^mesh-database
      element-shape       :tri
      surface-element-size-method (nth 2 '(0 1 2))
      curvature-refinement-method :isotropic-relative
      )
    (tet-mesh :class 'tet-mesh-class
      mesh-database-object ^^mesh-database
      object-to-mesh      ( first (get-link-surface-mesh-elements-query-objects-list
                                   ^superior)))
    ;; Example subobjects for analysis in AML:----->
    (fixed-nodes :class 'mesh-nodes-query-class
      )
    (loaded-nodes :class 'mesh-query-nodes-from-interface-class
      )
    ; <-----
  ))
  (define-method get-surface-mesh link-mesh-class ()
    !mesh-object)
  (define-method get-mesh-database link-mesh-class ()
    !mesh-database)
  (define-method get-mesh-entities-list link-mesh-class ()

```



```

    (if (get-export-surface !link-model)
        (the nodes-query mesh-entities-list (:from !mesh-object))
        (the tet-mesh nodes-query mesh-entities-list)))
(define-method get-link-mesh-node-query-objects-list link-mesh-class ()
  (list (the nodes-query (:from !mesh-object))))
(define-method get-link-surface-mesh-elements-query-objects-list link-mesh-class ()
  (when (the surface-elements-query (:from !mesh-object :error nil :relation nil))
    (list (the surface-elements-query (:from !mesh-object)))))
(define-method get-link-solid-mesh-elements-query-objects-list link-mesh-class ()
  (when (the elements-query (:from !tet-mesh :error nil :relation nil))
    (list (the elements-query (:from !tet-mesh)))))
)
(define-method get-rbe2-constraints-list link-mesh-class ()
  !joint-elements)
(define-method get-rbe2-independent-node-list link-mesh-class ()
  !rbe2-independent-node-list)
(define-method get-all-node-objects-list link-mesh-class ()
  (list !node-set))
(define-method get-node-position-in-query-list link-mesh-class (node-query-object)
  (1+ (position node-query-object (the node-set query-objects-list))))
;=====
; END link-mesh-class definitions
;=====

```

## E.10 Analysis.aml

```

;=====
; Class: RBE2-collection
; Instantiates every RBE2 as a series-object
;=====
(define-class RBE2-collection
  :inherit-from (series-object)
  :properties (
    link-mesh-object nil ;;link-mesh-class
    constraints-list (get-rbe2-constraints-list ^link-mesh-object)
    class-expression 'analysis-rigid-body-element-type-1-class
    series-prefix 'rbe2
    quantity (length ^constraints-list)
    init-form '(

```

```

independent-node-query-object (get-rbe2-independent-node
                               (nth ^index ^constraints-list))
dependent-nodes-query-object (get-rbe2-dependent-nodes
                              (nth ^index ^constraints-list))
dependent-degrees-of-freedom-list '(1 2 3 4 5 6)
;; The node is added in the node-set manually
create-new-independent-node? nil
;; Have to set the id manually, as default is 1 for each instance
      id                (+ 1 ^index)
)))

=====
; END RBE2-collection definitions
=====
;=====  

; Class: analysis-link-model-class
; Used for creating objects for writing to the nastran bdf file.
; Can also be used to set up a load case for use in an AML analysis
;=====  

(define-class analysis-link-model-class
  :inherit-from (analysis-model-class)
  :properties (
    mesh-model-object      nil ;;This property is set as a link-mesh-class
    link-model             nil ;link-model-class
    mesh-database-object   (get-mesh-database ^mesh-model-object)
    export-surface?       (default t)
    material-catalog-object ^material-catalog
    materials-list         (list (the material-name (:from ^material-selection)))
    property-set-objects-list (progn
                               (change-aluminum-properties !material-catalog)
                               (change-steel-properties !material-catalog)
                               (if ^export-surface?
                                   (list ^link-2D-material-properties)
                                   (list ^link-3D-material-properties)
                               )
    )
    element-set-2d-objects-list (if ^export-surface?
                                     (list ^link-analysis-surface-elements) nil)
    element-set-3d-objects-list (if ^export-surface?
                                     nil (list ^link-analysis-solid-elements))
    mesh-object            (get-surface-mesh ^mesh-model-object)
  )

```

```

node-set-objects-list      (get-all-node-objects-list ^mesh-model-object)
rigid-body-element-objects-list (children ^rbe2-list
                                     :class 'analysis-rigid-body-element-type-1-class)
(material-selection :class 'option-property-class
  options-list (children ^^material-catalog)
  labels-list  (loop for material in (children ^^material-catalog)
                    collect (the material-name (:from material))
                      )
  mode        'menu
  formula     (default (the material-catalog steel))
  label       "Select Material"
)
)
)
:subobjects (
  (material-catalog :class 'material-catalog-class
    )
  (link-2D-material-properties :class 'analysis-property-set-2d-type-1-class
    material-catalog-object ^^material-catalog
    material-name      (write-to-string (the material-name (:from ^material-selection)))
    thickness          (/ (get-smallest-mesh-size ^^link-model) 2.5)
  )
  (link-3D-material-properties :class 'analysis-property-set-3d-type-1-class
    material-catalog-object ^^material-catalog
    material-name      (write-to-string (the material-name (:from ^material-selection)))
  )
  (link-analysis-surface-elements :class 'analysis-element-set-2d-type-1-class
    query-objects-list (get-link-surface-mesh-elements-query-objects-list
                        ^^mesh-model-object)
    property-set-object ^^link-2D-material-properties
  )
  (link-analysis-solid-elements :class 'analysis-element-set-3d-type-1-class
    query-objects-list (get-link-solid-mesh-elements-query-objects-list
                        ^^mesh-model-object)
    property-set-object ^^link-3D-material-properties
  )
  (rbe2-list :class 'RBE2-collection
    link-mesh-object ^^mesh-model-object
  )
)
;; Example subobjects for analysis in AML:----->
(fixed-nodes-constraint :class 'analysis-constraint-displacement-class

```

```

    )
  (nodal-load :class 'analysis-load-force-nodal-class
    )
  (load-case-1 :class 'analysis-load-case-class
    )
  ;; <-----
  (nastran-interface :class 'nastran-analysis-class
    analysis-model-object ^superior
    model-name (format nil "~a - ~a" (the folder (:from ^^mechanism-selection))
      (the version-name (:from ^^version-selection)))
    nastran-file-name (concatenate (write-to-string (object-name ^superior)) ".bdf")
    nastran-version (nth 2 '(:nei-nastran :msc-nastran :nx-nastran))
    ;; could write 'nastran-version :nx-nastran' but this also shows the available versions
  )))

(define-method get-material-type analysis-link-model-class ()
  !material-selection)

(define-method get-yield-strength material-class ()
  (if (nth 4 !material-properties)
      (nth 1 (nth 4 !material-properties))
      (progn
        (change-steel-properties !superior)
        (nth 1 (nth 4 !material-properties))
      )))

;=====
; Method for circumventing AML bug when writing to bdf.
; Manually overwriting all material-properties for steel,
; as well as adding the yield strength for Steel, API 5L X65.
;=====

(define-method change-steel-properties material-catalog-class ()
  (unless (nth 1 (nth 4 (the steel material-properties)))
    (change-value (the steel material-properties)
      (list (list 'elastic-modulus 9.9E7 (list 'N (list 'm -2)) "youngs modulus")
        (list 'poissons-ratio 0.29 'nil "")
        (list 'mass-density 7.85 (list 'kg (list 'cm -3)) "")
        (list 'shear-modulus 7.7E7 (list 'N (list 'm -2)) "")
        (list 'yield-strength 4.48E5 (list 'N (list 'm -2)) ""))
      )))

;=====
; Method for circumventing AML bug when writing to bdf.
; Manually overwriting all material-properties for aluminum,

```

```

; as well as adding the yield strength for Aluminum 6061-T6
;=====
(define-method change-aluminum-properties material-catalog-class ()
  (unless (nth 1 (nth 4 (the aluminum material-properties)))
    (change-value (the aluminum material-properties)
      (list (list 'elastic-modulus 6.89E7 (list 'N (list 'm -2)) "youngs modulus")
        (list 'poissons-ratio 0.33 'nil "")
        (list 'mass-density 2.72 (list 'kg (list 'cm -3)) "")
        (list 'shear-modulus 2.6E7 (list 'N (list 'm -2)) "")
        (list 'yield-strength 2.76E5 (list 'N (list 'm -2)) ""))
      )))
(define-method property-classification-list analysis-link-model-class ()
  (list
    (list "Input Properties"
      '(
        material-selection
      ) )))
;=====
; END analysis-link-model-class definitions
;=====

```

## F.11 Link-member-geometry.aml

```

;=====
; Class: connection-model
; Creates a NURBS curve between two endpoints, using Sheth-Uicker definitions
;=====
(define-class connection-model
  :inherit-from (nurb-curve-object)
  :properties (
    pij (convert-coords ^frame_D '(0 0 0) :from :local :to :global)
    wij (convert-vector ^frame_D '(0 0 1) :from :local :to :global)
    pj_k (convert-coords ^frame_A '(0 0 0) :from :local :to :global)
    wj_k (convert-vector ^frame_A '(0 0 1) :from :local :to :global)
    weight-points (append (the weight-list (:from ^shape-ref))
      (make-sequence 'list (- (length (the point-list (:from ^shape-ref)))
        (length (the weight-list (:from ^shape-ref)))) :initial-element 1)
      )
    start-point (the origin (:from ^frame_D))
  )

```

```

end-point (the origin (:from ^frame_A))
middle-points (case ^line-config
  ('paralell (let (
    (start-tangent (add-vectors ^start-point (multiply-vector-by-scalar
      (normalize ^perpendicular-dir) (half ^param_b))) ))
    (end-tangent (add-vectors ^end-point (multiply-vector-by-scalar
      (normalize ^perpendicular-dir) (- (half ^param_b)))) ))
    (start-weight (list 0.5))
    (end-weight (list 0.5))
  )
  (if (roughly-same-point start-tangent end-tangent)
    (list (append start-tangent start-weight))
    (list (append start-tangent start-weight) (append end-tangent end-weight)))
  )
))

('intersecting (let (
  (center (nth 0 ^inter_points))
  (middle-point (add-vectors center (multiply-vector-by-scalar (normalize
    (add-vectors (subtract-vectors ^start-point center)
      (subtract-vectors ^end-point center))) ^param_a)) )
  (angle-start-middle (/ (angle-between-2-vectors
    (subtract-vectors ^start-point center)
    (subtract-vectors middle-point center) ) 2))
  (start-tangent (add-vectors center (multiply-vector-by-scalar
    (normalize (add-vectors (subtract-vectors ^start-point center)
      (subtract-vectors middle-point center) )
    (/ ^param_a (cosd angle-start-middle))))))
  (angle-middle-end (/ (angle-between-2-vectors
    (subtract-vectors middle-point center)
    (subtract-vectors ^end-point center) ) 2))
  (end-tangent (add-vectors center (multiply-vector-by-scalar
    (normalize (add-vectors (subtract-vectors middle-point center)
      (subtract-vectors ^end-point center) )
    (/ ^param_a (cosd angle-middle-end))))))
  (start-weight (list (sind (/ (angle-between-2-vectors
    (subtract-vectors start-tangent ^start-point)
    (subtract-vectors start-tangent middle-point)
  ) 2))))
  (middle-weight (list 1))
  (end-weight (list (sind (/ (angle-between-2-vectors

```

```

        (subtract-vectors end-tangent middle-point)
        (subtract-vectors end-tangent ^end-point)
      ) 2))))
      (list (append start-tangent start-weight) (append middle-point middle-weight)
            (append end-tangent end-weight))))
start-weight (list (append ^start-point (list 1)))
end-weight (list (append ^end-point (list 1)))
points (let (
            (shape-points (loop for p-index in (the point-list (:from ^shape-ref))
                                for w in ^weight-points
                                collect (append (the coordinates (:from
                                                                    (nth p-index ^point-ref-list))) (list w))))
            (if shape-points
                (append ^start-weight shape-points ^end-weight)
                (append ^start-weight ^middle-points ^end-weight)))
          rational? t
          homogeneous? t
          ;degree 2
          line-config (line-pose (the superior))
          inter_points (inter_section (the superior) ^line-config)
          perpendicular-dir (perp-dir (the superior) ^line-config)
          param_a (vector-length (subtract-vectors (the origin (:from ^frame_A))
                                                    (the origin (:from ^frame_B))))
          param_b (vector-length (subtract-vectors (the origin (:from ^frame_B))
                                                    (the origin (:from ^frame_C))))
          param_c (vector-length (subtract-vectors (the origin (:from ^frame_C))
                                                    (the origin (:from ^frame_D))))
        )
:subjects (
  ;;cross section at start of spline
  (spline-frame_start :class 'frame-data-model
    point-ref ^point-ref_D
    z-vector-ref ^z-vector-ref_D
    (x-vector-ref :class 'vector-data-model
      direction (subtract-vectors (nth 1 ^points) (nth 0 ^points))
    ))
  ;;cross section at end of spline
  (spline-frame_end :class 'frame-data-model
    point-ref ^point-ref_A
    z-vector-ref ^z-vector-ref_A

```

```

(x-vector-ref:class 'vector-data-model
  direction (subtract-vectors (nth (1- (length ^points)) ^points)
    (nth (- (length ^points) 2) ^points))))
;;augumented frames from SU-convention
(frame_B :class 'frame-data-model
  (point-ref:class 'point-data-model
    coordinates (nth 0 ^^inter_points)
  )
  (z-vector-ref:class 'vector-data-model
    direction ^^wij
  )
  (x-vector-ref:class 'vector-data-model
    direction ^^perpendicular-dir
  ))
(frame_C :class 'frame-data-model
  (point-ref:class 'point-data-model
    coordinates (nth 1 ^^inter_points)
  )
  (z-vector-ref:class 'vector-data-model
    direction ^^wjk
  )
  (x-vector-ref:class 'vector-data-model
    direction ^^perpendicular-dir
  )))
(define-method get-spline-frame_start connection-model ()
  !spline-frame_start)
(define-method get-spline-frame_end connection-model ()
  !spline-frame_end)
(define-method get-start-weight connection-model ()
  !start-weight)
;=====
; Finds the middle point between p1 and p2, given the direction d1
;=====
(defun m-point (p1 d1 p2)
  (add-vectors p1 (proj_v d1 (multiply-vector-by-scalar (subtract-vectors p2 p1) 0.5))
  )
;=====
; Finds the closest point from p1
;=====
(defun cl-point (p1 d1 p2 d2)

```



```

(let (
  (n1x (cross-product d1 d2))
  (n1d (dot-product n1x (cross-product p2 d2)))
  (n2d (dot-product n1x (cross-product p1 d2)))
  (d1s (dot-product n1x n1x))
  (l1s (multiply-vector-by-scalar d1 (/ n1d d1s)))
  (l2s (multiply-vector-by-scalar d1 (/ n2d d1s)))
)
  (add-vectors p1 (subtract-vectors l1s l2s)))

;=====
;Determine configuration of two lines in relation to eachother
;=====

(define-method line-pose connection-model ()
  (let (
    (v0_1 (cross-product !p1j !w1j))
    (v0_2 (cross-product !p1j !w2j))
    (coplan (* 0.5 (+ (dot-product !w1j v0_2) (dot-product v0_1 !w2j))))
    (normal-mag (vector-length (cross-product !w1j !w2j)))
    (coincident (vector-length (cross-product (subtract-vectors !p1j !p2j) !w1j)))
  )
    (if (/= 0 coplan) 'skew (if (/= 0 normal-mag) 'intersecting (if (= 0 coincident) 'coincident 'parallell)))
  )
)

;=====
; Generalized closest points
; If lines Gij Gjk are intersecting or skew: closest point
; If lines Gij Gjk are coincident or parallel: mid-point
; Calculate intersection between lines
;=====

(define-method inter_section connection-model (line-config)
  (case line-config
    ('skew
     )
    ('list (cl-point !p1j !w1j !p2j !w2j) (cl-point !p2j !w2j !p1j !w1j))
    ('intersecting
     (list (cl-point !p1j !w1j !p2j !w2j) (cl-point !p2j !w2j !p1j !w1j))
     )
    ('coincident
     (list (m-point !p1j !w1j !p2j) (m-point !p2j !w2j !p1j))
     )
  )
)

```

```

    (' paralell
      (list (m-point !pij !wij !pjk) (m-point !pjk !wjk !pij))
    )
  )
)

;=====
; Returns generalized perpendicular direction
;=====

(define-method perp-dir connection-model (line-config)
  (let (
    (cross (cross-product !wij !wjk)
      (ortho-comp (orthogonal-projection-complement !wij (subtract-vectors !pjk !pij)))
    )
  (case line-config
    ('skew cross)
    ('intersecting cross)
    ('coincident (read-from-string (pop-up-text-prompt
      :nb-entries 1
      :title "Please specify direction"
      :prompt "Type in x-vector"
      :init-text "(1 0 0)"
      :x-offset (/ (nth 0 (get-screen-size)) 2)
      :y-offset (/ (nth 1 (get-screen-size)) 2))))
    ('paralell ortho-comp)
  )
)
)

;=====
; Returns the orthogonal projection of a vector b onto some vector a, pi_a(b)
;=====

(defun proj_v (a b)
  (multiply-vector-by-scalar a (/ (dot-product b a) (dot-product a a)))
)

;=====
; Returns the orthogonal projection of vector b into the orthogonal
; complement of vector a, tau_a(b)
;=====

(defun orthogonal-projection-complement (a b)
  (subtract-vectors b (proj_v a b))
)

```

```

;=====
; END connection-model definitions
;=====
;=====
; Class: member-solid-model
; Creates the link cross-section sweep for a link member
;=====
(define-class member-solid-model
  :inherit-from (tagging-object general-sweep-class)
  :properties(
    mesh-size-factor (default 0.25) ;;25% of the member's smallest dimension
    (mesh-element-size :class 'editable-data-property-class
      formula (calculate-mesh-size (the superior))
      label "Mesh element size"
    )
    tag-dimensions '(1 2 3)
    tag-attributes (list ^mesh-element-size .1
      0 0.1 0 20.0 1.0e-5)
    (display? :class 'flag-property-class
      formula (when (the cross-section-type (:from ^shape-ref)) t)
    )
    joints-on-member nil ;;List of the two joint elements on the member
    frame_D (the sub-frame (:from (nth 0 ^joints-on-member)))
    frame_A (the sub-frame (:from (nth 1 ^joints-on-member)))
    point-ref_D (the point-ref (:from ^frame_D))
    point-ref_A (the point-ref (:from ^frame_A))
    ;;test for "link twist"
    z-vector-ref_A (if (is-vectors-dependent (get-z-vector-ref_A ^frame_A)
      (get-z-vector-ref_D ^frame_D) (get-x-vector-ref_D ^connection))
      ;;test for opposite joint directions
      (if (is-joint-directions-opposite
        (get-z-vector-ref_A ^frame_A) (get-z-vector-ref_D ^frame_D)
        (get-x-vector-ref_A ^connection) (get-x-vector-ref_D ^connection))
        (the z-vector-ref (:from ^frame_D))
        (the z-vector-ref (:from ^frame_A))
      )
      (the z-vector-ref (:from ^frame_D))
    )
    z-vector-ref_D (the z-vector-ref (:from ^frame_D))
    ;;cross section dimension, width 0.04 / height 0.04

```

```

shape-ref nil
width (nth 0 (the solid-dimensions (:from ^shape-ref)))
height (if (< 1 (length (the solid-dimensions (:from ^shape-ref))))
          (nth 1 (the solid-dimensions (:from ^shape-ref)))
          (nth 0 (the solid-dimensions (:from ^shape-ref)))
        )
width-end (if (< 2 (length (the solid-dimensions (:from ^shape-ref))))
              (nth 2 (the solid-dimensions (:from ^shape-ref)))
              (nth 0 (the solid-dimensions (:from ^shape-ref)))
            )
height-end (if (< 3 (length (the solid-dimensions (:from ^shape-ref))))
               (nth 3 (the solid-dimensions (:from ^shape-ref)))
               (nth 1 (the solid-dimensions (:from ^shape-ref)))
             )
;; Sweep parameters
profile-objects-list (list
                      ^cross-section_D
                      ^cross-section_A
                      )
path-points-coords-list (list
                          (the origin (:from ^frame_D))
                          (the origin (:from ^frame_A))
                          )
profile-match-points-coords-list (list
                                  (vertex-of-object ^cross-section_D)
                                  (vertex-of-object ^cross-section_A)
                                  )
path-object ^connection
tangential-sweep? t
;; If two cross-sections, only nil works, with one cross-section t gives best mesh
simplify? nil
render 'shaded
;; cross-section selection
(cross-section-type :class 'option-property-class
  label "Cross-section Type"
  mode 'menu
  formula (if (the cross-section-type (:from ^shape-ref))
              (nth (position (write-to-string (the cross-section-type (:from ^shape-ref)))
                             !labels-list) !options-list)
              (nth (position (write-to-string (the cross-section-type (:from

```

```

        ^default-shape)))
    !labels-list) !options-list))
options-list (reverse (class-direct-defined-subclasses 'cross-section-model))
labels-list (loop for option in !options-list
                  collect (remove "-section" (write-to-string option))
                  )
)
)
optimization-object (default nil)
property-objects-list (list
                       (list (the superior cross-section-type self)
                              '(automatic-apply? t)
                              )
                       '("Set cross-section" (button1-parameters :set-c button3-parameters
                                                                :set-c)
                        ui-work-area-action-button-class)
                       (the superior width self)
                       (the superior height self)
                       (the superior width-end self)
                       (the superior height-end self)
                       (list (the superior display? self)
                              '(automatic-apply? t)
                              )
                       '("Draw..." (button1-parameters :draw button3-parameters :draw)
                        ui-work-area-action-button-class)
                       '("Undraw..." (button1-parameters :undraw button3-parameters
                                                            :undraw)
                        ui-work-area-action-button-class)
                       ""
                       (list (the superior mesh-element-size self)
                              '(automatic-apply? t)
                              )
                       )
)
:subobjects (
  (connection :class 'connection-model
             )
  (cross-section_D :class !cross-section-type
                  reference-object (the spline-frame_start (:from ^connection))
                  orientation      (list
                                   (rotate 90 :x-axis)
                                   (rotate 90 :z-axis)
                                   )
                  )
)

```

```

    optimization-object ^^optimization-object
  )
  (cross-section_A :class !cross-section-type
    width      (if ^optimization-object
                (get-width ^optimization-object)
                ^width-end)
    height     (if ^optimization-object
                (get-height ^optimization-object)
                ^height-end)
    reference-object (the spline-frame_end (:from ^connection))
    orientation    (list
                    (rotate 90 :x-axis)
                    (rotate 90 :z-axis)
                    )
    optimization-object ^^optimization-object
  )
)

(define-method get-z-vector-ref_A sub-frame-data-model ()
  (the direction (:from !z-vector-ref)))
(define-method get-z-vector-ref_D sub-frame-data-model ()
  (the direction (:from !z-vector-ref)))
(define-method get-x-vector-ref_D connection-model ()
  (the direction (:from (the x-vector-ref (:from !spline-frame_start))))))
(define-method get-x-vector-ref_A connection-model ()
  (the direction (:from (the x-vector-ref (:from !spline-frame_end))))))
(define-method get-cross-section_D member-solid-model ()
  !cross-section_D)
(define-method get-connection member-solid-model ()
  !connection)
(define-method is-displayed? member-solid-model ()
  !display?)
(define-method get-joints-on-member member-solid-model ()
  (when !joints-on-member
    !joints-on-member))
;=====
; Returns the smallest dimension of a member times a mesh scaling factor defined in
; member-solid-model
;=====
(define-method calculate-mesh-size member-solid-model ()

```

```

(* (min !width !height !width-end !height-end) !mesh-size-factor)
;=====
; Returns the members width and height for a given joint-element-model
;=====
(define-method get-max-dimensions member-solid-model (joint-element)
  (let (
    (pos (position joint-element (get-joints-on-member (the))))
    (width (if pos
      (get-width (get-cross-section-at-joint-element (the) joint-element)
        0));;A joint not on the member might be sent as input
      ;;In that case, return 0
    (height (if pos
      (get-height (get-cross-section-at-joint-element (the) joint-element)
        0))
    )
    (list width height)))
;=====
; Returns the member cross-section at a joint element
;=====
(define-method get-cross-section-at-joint-element member-solid-model (joint-element)
  (if (= 0 (position joint-element (get-joints-on-member (the))))
    !cross-section_D
    !cross-section_A
  ))
(define-method get-spline-frame-at-joint-element member-solid-model (joint-element)
  (if (= 0 (position joint-element (get-joints-on-member (the))))
    (get-spline-frame_start !connection)
    (get-spline-frame_end !connection)))
;=====
; Left-click button methods for member-solid-model
;=====
(define-method work-area-button1-action member-solid-model (params)
  (case params
    (:set-c
      ; (draw self :draw-subobjects? nil)
    )
    (:draw
      (draw self :draw-subobjects? nil)
    )
    (:undraw

```

```

    (undraw self :subobjects? nil)
  )
)
)
)
;=====
; Right-click button methods for member-solid-model
;=====
(define-method work-area-button3-action member-solid-model (params)
  (case params
    (:draw
      (draw self :draw-subobjects? t)
      )
    (:undraw
      (undraw self :subobjects? t)
      )
    )
  )
)
;=====
; Checks whether vector v1 is linear independent to,
; the plane defined by the vectors pv1 and pv2
;=====
(defun is-vectors-dependent (v1 pv1 pv2)
  (line-is-in-plane '(0 0 0) v1 '(0 0 0) (cross-product pv1 pv2)))
;=====
; Checks whether the joint directions z1 and z2 is
; opposite when the x-vector (sweeping direction)
; is the same.
;=====
(defun is-joint-directions-opposite (z1 z2 x1 x2)
  (and (equal (nth 2 z1) (- (nth 2 z2)))
    (equal (round-point (every-but-last x1) 3) (round-point (every-but-last x2) 3))))
;=====
; Returns every element of list except last
;=====
(defun every-but-last (list)
  (loop for l on list
    while (rest l)
    collect (first l)))
;=====
; END member-solid-model definitions

```



```

=====
;
;=====  

; Class: members-on-link-collection  

; Instantiates every member on a link as series-objects  

;=====  

(define-class members-on-link-collection
  :inherit-from (series-object)
  :properties (
    shapes-on-link (default nil)
    quantity      (length ^connection-between-2-constraints-combinations)
    class-expression 'member-solid-model
    series-prefix  'member
    optimization-object (default nil)
    init-form '(
      joints-on-member (nth ^index
        ^^connection-between-2-constraints-combinations)
      shape-ref        (nth ^index ^shapes-on-link)
      optimization-object ^^optimization-object
      )
    )
  )
)
;=====  

; END members-on-link-collection definitions  

;=====

```

## F.12 Link-surface-geometry.aml

```

=====
;
;=====  

; Class: surface-model  

; Model creating a thickened surface from three connected curves  

;=====  

(define-class surface-model
  :inherit-from (tagging-object surface-thickened-class)
  :properties (
    members      nil
    (mesh-element-size :class 'editable-data-property-class
      formula (default ^thickness)
      label "Mesh element size"
    )
  )
)

```

```

tag-dimensions      '(1 2 3)
tag-attributes      (list ^mesh-element-size .1
                      0 0.1 0 20.0 1.0e-5)
(display? :class 'flag-property-class
  formula (loop for i in ^edge-combination
                when (or (not (the display? (:from (nth i ^members))))
                          (not (the geom (:from (get-connection (nth i ^members))))))
                do (return nil)
                finally (return t) ))
edge-combination nil
;; The source-object has to be nil when it can't be displayed. Else there will be a null geom
source-object (if ^display? ^surface nil)
;; Surface thickness 40% of smallest cross-section height
thickness (* 0.4 (loop for i in ^edge-combination
                       minimize (get-height (get-cross-section_D (nth i ^members)))
                       ))
front-thickness (/ ^thickness 2)
back-thickness (/ ^thickness 2)
render 'shaded
(start-point :class 'point-object
  coordinates (nth 0 (get-start-weight (get-connection (nth (nth 0 ^edge-combination)
                                                           ^members))))
  property-objects-list (list
                        (list (the superior display? self)
                              '(automatic-apply? t)
                              (the superior mesh-element-size self)
                              ))
                        ))
:subobjects (
  (surface :class 'surface-from-uv-curves-class
    u-curves-objects-list (list ^start-point
                               (get-connection (nth (nth 2 ^edge-combination) ^members)))
    v-curves-objects-list (list (get-connection (nth (nth 0 ^edge-combination) ^members))
                                (get-connection (nth (nth 1 ^edge-combination) ^members)))
    )))
;=====
; END surface-model definitions
;=====
;=====
; Class: surfaces-on-link-collection
; Instantiates every surface-model on a link as series-objects

```

```

=====
(define-class surfaces-on-link-collection
  :inherit-from (series-object)
  :properties (
    members-list      nil
    closed-loops-combinations (sweep-loop-combinations
                               (length ^constraints-incident-on-link-list))
    visible-members-index (loop for mem in ^visible-members-ref-list
                               collect (the index (:from mem))
                               )
    valid-surface-loops (intersection ^closed-loops-combinations
                                       (list-3-subset-combinations ^visible-members-index))
    quantity            (length ^valid-surface-loops)
    class-expression    'surface-model
    series-prefix       'surface
    init-form '(
      edge-combination (nth ^index ^valid-surface-loops)
      members      ^members-list
      )))
(defun sweep-loop-combinations (n)
  (let (
    (c-loops (3-edge-loop-combinations n))
    (sweep-con (connection-combinations n))
    )
    (loop for ci from 0 to (1- (length c-loops))
      for list-com = (list-combinations (nth ci c-loops))
      collect (loop for si from 0 to (1- (length list-com))
                  collect (position (nth si list-com) sweep-con)
                  )))
=====
; C(n,3) = 3! / ( 3! (n - 3)!
; Output: List unique combinations of three, given n numbers
=====
(defun 3-edge-loop-combinations (n)
  (loop for i from 0 to (- n 3)
    append (loop for j from (1+ i) to (- n 2)
      append (loop for k from (1+ j) to (1- n)
        collect (list i j k)
        )))
(defun list-3-subset-combinations (p)

```

```

(let (
  (l (if (typep p 'list) p (if (typep p 'fixnum) (loop for i from 0 to (1- p) collect i) (list) )) )
  (n (length l))
  )
(loop for i from 0 to (- n 3)
  append (loop for j from (1+ i) to (- n 2)
    append (loop for k from (1+ j) to (1- n)
      collect (list (nth i 1) (nth j 1) (nth k 1))
    ))))
;=====
; END surfaces-on-link-collection definitions
;=====

```

## E.13 Links.aml

```

;=====
; Class: shape-model
; Defines the shape properties of a member
;=====
(define-class shape-model
  :inherit-from (object)
  :properties (
    label          nil
    link-ref        (default 'default)
    sweep-index     (default 'default)
    cross-section-type (read-from-string (remove "-section" (write-to-string
      (default 'circular-section))))
    solid-dimensions '(0.04 0.04)
    point-list      nil
    weight-list     nil
  )
)
(define-method get-cs-type shape-model ()
  !cross-section-type)
;=====
; END shape-model definitions
;=====
;=====
; Class: link-geometry-class

```

```

; Generates a link's total geometry, including joints
;=====
(define-class link-geometry-class
  :inherit-from (tagging-object geometry-with-split-periodic-faces-class)
  :properties (
    incident-constraints nil
    ;source-object ^difference-element ;; Used for
    ;geometry-with-split-periodic-faces-class
    source-object (if ^blend? ^blend-object ^difference-element)
    max-element-size 0.02
    default-shape (let(
      (def (loop for shape in (children ^^shapes :class 'shape-model)
        when (and
          (equal 'default (the sweep-index (:from shape)))
          (equal ^link-index (the link-ref (:from shape)))
        ) do
          (return shape))))
      (if def def ^^default-shape)
    )
    constraint-connection-combination (connection-combinations (length
      ^incident-constraints))
    surfaces-ref-list (children ^surfaces :class 'surface-model
      :test '(and !geom !display?))
    members-ref-list (children ^sweeps :class 'member-solid-model)
    visible-members-ref-list (children ^sweeps :class 'member-solid-model
      :test '!display?)
    union-list (loop for constraint in ^incident-constraints
      append (get-joint-union-list constraint)
    )
    difference-list (loop for constraint in ^incident-constraints
      append (get-joint-difference-list constraint)
    )
  )
  ;; List of the final geometry
  object-list (append
    (list ^imprint-union-element)
    ^difference-list
  )
  (imprint-union-element :class '(tagging-object
    geometry-with-split-periodic-faces-class)
  tag-dimensions '(1 2 3)

```

```

tag-attributes (list ^max-element-size .01
                  0 0.1 0 20.0 1.0e-5)
source-object ^^union-element
)
;; Imprints the union-element with the point reference
(imprint-constraint-points :class '(tagging-object imprint-class)
 target-object ^^imprint-union-element
 tool-object-list (loop for c in ^incident-constraints
                       collect (the point-ref (:from c))
                      )
)
;; The final sewn geometry of members, surfaces and joints
(difference-element :class '(tagging-object difference-object)
 tag-dimensions '(1 2 3)
 tag-attributes (list ^max-element-size .01
                    0 0.1 0 20.0 1.0e-5)
 object-list      (append (list ^union-element) ^difference-list)
 simplify?       t
)
;; Union of surfaces, members and joint geometry without difference objects
(union-element :class '(tagging-object union-object)
 tag-dimensions '(1 2 3)
 tag-attributes (list ^max-element-size .01
                    0 0.1 0 20.0 1.0e-5)
 object-list      (append (if ^surfaces-ref-list ^surfaces-ref-list nil)
                          ^visible-members-ref-list
                          ^^union-list)
 simplify?       t
)
render 'shaded
(blend? :class 'option-property-class
 mode      'radio
 options-list (list t nil)
 labels-list (list "Yes" "No")
 formula     (default t)
 label       "Automatic blending?"
)
(blend-mesh-element-size :class 'editable-data-property-class
 formula (default (/ (the max-element-size (:from (first ^incident-constraints))) 2))
 label "Blend mesh element size"
)

```

```

)
(blend-object :class '(tagging-object blend-class)
  source-object ^^difference-element
  edge-ids (get-intersecting-edges-list ^superior)
  radii (loop for edge in ^edge-ids
           append (list (/ (the max-width (:from (first ^^incident-constraints)))
                          5)))
  tag-dimensions '(1 2 3)
  tag-attributes (list ^blend-mesh-element-size
                      0 0.1 0 20.0 1.0e-5)
)
;; This list collects the members that can be drawn.
;; If it is empty, this links will not be considered in the mesh or analysis
drawable-members-ref-list (loop for member in ^members-ref-list
                              if (the geom (:from member))
                              collect member
                              )
;; simplify? t removes common boundaries in the geometry
;; simplify? nil keeps them
simplify? nil
property-objects-list (list
  "Link sewn with incident joint element"
  (list (the superior cross-section-type self)
        '(automatic-apply? t)
        '("Set all cross-sections" (button1-parameters :set-c)
          ui-work-area-action-button-class)
        ""
        '("Draw" (button1-parameters :draw-sewn-geometry)
          ui-work-area-action-button-class)
        '("Draw with mesh" (button1-parameters :draw-with-mesh)
          ui-work-area-action-button-class)
        '("Undraw" (button1-parameters :undraw-sewn)
          ui-work-area-action-button-class)
        "")
  (list (the superior blend? self)
        '(automatic-apply? t)
        (list (the superior blend-mesh-element-size self)
              '(automatic-apply? t)
              ;'("Add/remove surface" (button1-parameters :set-surface)
                ui-work-area-action-button-class)

```

```

)
optimization-object nil)
:subobjects (
  (surfaces :class 'surfaces-on-link-collection
  members-list ^^members-ref-list
  )
  (sweeps :class 'members-on-link-collection
  shapes-on-link (get-shapes-on-link (the superior superior))
  optimization-object ^^optimization-object
  )))
(define-method get-intersecting-edges-list link-geometry-class ()
  (let* (
    (difference-edge-list (vgl::k-sub-geoms (the geom (:from !difference-element)) 1))
    (member-geom-list (loop for member in !visible-members-ref-list
      append (list (copy-geom (the geom (:from member))))))
    (members-diff-intersection (loop for geom in member-geom-list
      append (list (vgl::intersection-geoms (list (copy-geom (the geom
        (:from !difference-element)))
        geom))))))
    (non-free-joints (remove nil (loop for joint in !incident-constraints
      when (not (is-free-constraint joint)) collect joint)))
  )
  (remove nil (loop for edge in difference-edge-list
    append (loop for member in members-diff-intersection
      append (loop for joint in non-free-joints
        collect (when (and (vgl::intersect-geom-p member edge)
          (< (points-distance (get-sub-frame-coords joint) (geom-center edge))
            (* (the max-width (:from joint)) 0.75)))
          edge))))))
(define-method get-shapes-on-link link-geometry-class ()
  (let (
    (shape-list (make-sequence 'list (length !constraint-connection-combination)
      :initial-element !default-shape))
  )
  (loop for shape in (children ^^shapes :class 'shape-model)
    when (and (equal !link-index (the link-ref (:from shape)))
      (not (equal 'default (the sweep-index (:from shape)))))) do
    (replace shape-list (list shape) :start1 (the sweep-index (:from shape)))
  finally (return shape-list)))
(define-method get-surface link-geometry-class (index)

```



```

    (nth index !surfaces-ref-list)
(define-method has-line-cross-section link-geometry-class ()
  (loop for shape in (get-shapes-on-link (the)) do
    (if (equal 'line (the cross-section-type (:from shape)))
      (return t)
      finally (return nil)))
(define-method get-members link-geometry-class ()
  !members-ref-list)
(define-method get-visible-members link-geometry-class ()
  !visible-members-ref-list)
(define-method get-rbe2-dependent-nodes-list link-geometry-class ()
  (loop for c in !incident-constraints
    collect (get-rbe2-dependent-nodes c)))
;=====
; Returns all possible connections with all incident joints
; Input: Number of incident joints
;=====
(defun connection-combinations (n)
  (loop for j from 0 to (- n 2)
    append (loop for k from (1+ j) to (1- n)
      collect (list j k)))
;=====
; Left-click button methods for member-solid-model
;=====
(define-method work-area-button1-action link-geometry-class (params)
  (case params
    (:set-c
      (loop for m in (the members-ref-list (:from self)) do
        (change-value (the cross-section-type self (:from m)) !cross-section-type))
    ;; Sewn geometry refers to the union of a member and a joint
    (:draw-sewn-geometry
      (draw self :draw-subobjects? nil)
      )
    (:undraw-sewn
      (undraw self :subobjects? t)
      (undraw (the superior link-mesh-model)))
    (:draw-with-mesh
      (draw self :draw-subobjects? nil)
      (with-error-handler (:show-system-error? t)
        (draw (the superior link-mesh-model)))

```

```

))
(:set-surface
  (let (
    (display-value? (the display? (:from (nth 0 (the surfaces-ref-list))))))
    )
    (if display-value?
      (change-value (the display? (:from (get-surface (the) 0))) nil)
      (change-value (the display? (:from (get-surface (the) 0))) t)
    ))))
;=====
; END link-geometry-class definitions
;=====
;=====  

; Class: link-model-class
; Used to hold the link geometry, its mesh and the analysis as subobjects
;=====
(define-class link-model-class
  :inherit-from (object)
  :properties (
    ;; properties set from parent init-form
    label                nil
    constraints-incident-on-link-list    nil ;;List of joint-element-models
    connection-between-2-constraints-combinations
    (list-combinations ^constraints-incident-on-link-list)
    link-index           nil
    optimization-object  (default nil)
    has-line-cross-section? (has-line-cross-section ^link-geometry)
  )
  :subobjects (
    (link-geometry :class 'link-geometry-class
      incident-constraints ^^constraints-incident-on-link-list
      optimization-object ^^optimization-object
    )
    (link-mesh-model :class 'link-mesh-class
      geometry-model-object ^link-geometry
      link-model            ^superior
      joint-elements        ^^constraints-incident-on-link-list
    )
    (analysis :class 'analysis-link-model-class
      mesh-model-object ^^link-mesh-model

```

```

        link-model ^superior
    )))
(defun get-blend-mesh-size (dimensions-list)
  (loop for dim in dimensions-list do
    minimize dim into min-size
    finally (return (/ min-size 16))))
(define-method get-non-free-constraints-incident-on-link-list link-model-class ()
  (when !constraints-incident-on-link-list
    (loop for joint-element in !constraints-incident-on-link-list do
      if (not (string-equal "free" (get-constraint-type joint-element)))
        collect joint-element
    )))
(define-method get-shapes-on-link link-model-class ()
  (get-shapes-on-link !link-geometry))
(define-method get-smallest-mesh-size link-model-class ()
  (loop for member in (get-visible-members !link-geometry) do
    minimize (calculate-mesh-size member) into min-mesh
    finally (return min-mesh)))
(define-method get-link-geometry link-model-class ()
  !link-geometry)
(define-method get-joint-elements-on-link link-model-class ()
  !constraints-incident-on-link-list)
(define-method are-all-members-displayed? link-model-class ()
  (> (length (the visible-members-ref-list (:from !link-geometry))) 0))
(define-method get-mesh-model-object link-model-class ()
  !link-mesh-model)
(define-method get-mesh-database link-model-class ()
  (get-mesh-database !link-mesh-model))
(define-method get-material-type link-model-class ()
  (get-material-type !analysis))
(define-method get-export-surface link-model-class ()
  (the export-surface? (:from !analysis)))
;=====
; Returns all possible connections with all incident joints
; Input: List of incident joints, OR the number of incident joints
;=====
(defun list-combinations (p)
  (let (
    (l (if (typep p 'list) p (if (typep p 'fixnum) (loop for i from 0 to (1- p) collect i) (list))))
    (n (length l)))

```

```

)
(loop for j from 0 to (- n 2)
  append (loop for k from (1+ j) to (1- n)
    collect (list (nth j 1) (nth k 1))))))
;=====
; END link-model-class definitions
;=====

```

## E.14 Collections.aml

```

;=====
; Class: collection-class
; Superclass for the different collection types, to let them have access to
; the same read-from-file method
;=====
(define-class collection-class
  :inherit-from (object)
  :properties (
    collection-type nil
  ))
;=====
; Checks whether a specific file is accessible or not,
; if so then the corresponding read method is executed
;=====
(define-method read-from-file collection-class ()
  (let (
    (file-name (write-to-string !collection-type))
    (file-path (logical-path (the version-path (:from ^version-selection))
      (concatenate file-name ".txt")))
    (function-name (read-from-string (concatenate "read-" file-name "-from-file")))
  )
    (if (and
      file-path
      (stringp file-path)
      (probe-file file-path))
      (with-open-file (file file-path :direction :input)
        (apply function-name (list file)))
      (progn
        (message (format nil "\\~a\\" is not a valid file path." file-path) :append? t)

```

```

nil))))
;=====
; END collecion-class definitions
;=====
;=====  

;=====  

; Class: constraint-class
; Executes the read-from-file method and creates the corresponding subobjects for the constraints
;=====  

;=====  

(define-class constraint-collection
  :inherit-from (series-object collection-class)
  :properties (
    links-list      nil ;;Set as link-model-class objects
    collection-type 'constraints
    constraint-list  (cdr (read-from-file !superior))
    quantity        (length ^constraint-list)
    series-prefix   'c
    class-expression '(read-from-string (concatenate
      (nth 1 (nth !index !constraint-list)) "-constraint-class"))
      ;;The "-constraint-classes" are master-joint-models
    init-form '(
      point-ref      (nth (nth 0 (nth !index ^constraint-list)) ^point-ref-list)
      label          (concatenate (nth 1 (nth !index ^constraint-list)) " ")
      (write-to-string (nth 2 (nth !index ^constraint-list)))
      constraint-type (nth 1 (nth !index ^constraint-list))
      link-incidence (nth 2 (nth !index ^constraint-list))
      direction      (normalize (nth 3 (nth !index ^constraint-list)))
      degrees-of-freedom (nth 4 (nth !index ^constraint-list))
      constraint-variable (nth 5 (nth !index ^constraint-list))
      incident-links (let (
        (link1 (if (first (nth 2
          (nth !index ^constraint-list)))
          (nth (first (nth 2
            (nth !index ^constraint-list))) ^links-list)))
        (link2 (if (second (nth 2
          (nth !index ^constraint-list)))(nth (second (nth 2
            (nth !index ^constraint-list))) ^links-list)))
        (remove nil (list link1 link2))))
      property-objects-list (list
        '("Draw all joints" (button1-parameters :draw-joints)
          ui-work-area-action-button-class)

```

```

      '("Draw all RBE2 nodes" (button1-parameters :draw-rbe2)
        ui-work-area-action-button-class)
      ""
      '("Undraw all joints and nodes" (button1-parameters :undraw)
        ui-work-area-action-button-class)))

(define-method get-constraints constraint-collection ()
  (children (the) :class 'master-joint-model))
;=====
; Reads each line of the constraints file and adds this to a list
;=====
(defun read-constraints-from-file (stream)
  (when stream
    (loop for line = (read-line stream nil :eof)
      until (equal line :eof)
      for ls = (string-to-delimited-token-list line :delimiter #\tab :string-token? nil)
      for c-data = (list
        (nth 0 ls) (nth 1 ls) (nth 2 ls) (nth 3 ls) (nth 4 ls) (nth 5 ls)
        )collect c-data)))
;=====
; Left click button methods for constraint-collection
;=====
(define-method work-area-button1-action constraint-collection (params)
  (case params
    (:draw-joints
      (loop for constraint in (get-constraints (the)) do
        (with-error-handler (:error-message (concatenate "Error drawing " (write-to-string
          (object-name constraint))) :show-system-error? nil)
          (draw constraint))))
    (:draw-rbe2
      (loop for constraint in (get-constraints (the)) do
        (loop for joint-element in (children constraint) do
          (with-error-handler (:error-message (concatenate "Error drawing " (write-to-string
            (object-name constraint))) :show-system-error? t)
            (draw (get-rbe2-dependent-nodes joint-element))
            (draw (get-rbe2-independent-node joint-element))))))
    (:undraw
      (loop for constraint in (get-constraints (the)) do
        (loop for joint-element in (children constraint) do
          (undraw (get-rbe2-dependent-nodes joint-element))
          (undraw (get-rbe2-independent-node joint-element))))))
  )

```

```

        (undraw self)))
;=====
; END constraint-collection definitions
;=====
;=====
; Class: point-collection
; Executes the read-from-file and creates the corresponding subobjects for the points
;=====
(define-class point-collection
  :inherit-from (series-object collection-class)
  :properties (
    collection-type 'coordinates
    points-list      (cdr (read-from-file !superior))
    quantity         (length ^points-list)
    class-expression 'point-data-model
    series-prefix    'p
    init-form '(
      label          (nth 0 (nth ^index ^points-list))
                    coordinates (list (nth 1 (nth ^index ^points-list))
                                      (nth 2 (nth ^index ^points-list)) (nth 3 (nth ^index ^points-list)))
      id              ^index))
(define-method get-points point-collection ()
  (children (the) :class 'point-data-model))
;=====
; Reads each line of the coordinats file and adds this to a list
;=====
(defun read-coordinates-from-file (stream)
  (when stream
    (loop for line = (read-line stream nil :eof)
          until (equal line :eof)
          for ls = (string-to-delimited-token-list line :delimiter #\tab :string-token? nil)
          for coord-line = (list
            (read-from-string (nth 1 ls)) (nth 2 ls) (nth 3 ls) (nth 4 ls) (read-from-string (nth 5 ls))
            (read-from-string (nth 6 ls)) (nth 7 ls))
          collect coord-line)))
;=====
; END point-collection definitions
;=====
;=====
; Class: shape-collection

```

```

; Executes the read-from-file method and creates the corresponding subobjects for the shapes
;=====
(define-class shape-collection
  :inherit-from (series-object collection-class)
  :properties (
    collection-type 'shapes
    shapes-list (cdr (read-from-file !superior))
    quantity (length ^shapes-list)
    class-expression 'shape-model
    series-prefix 'shape
    init-form '(
      label          (nth 0 (nth ^index ^shapes-list))
      link-ref       (nth 1 (nth ^index ^shapes-list))
      sweep-index    (nth 2 (nth ^index ^shapes-list))
      cross-section-type (nth 3 (nth ^index ^shapes-list))
      solid-dimensions (if (not (nth 4 (nth ^index ^shapes-list)))
        (list 0 0)
        (nth 4 (nth ^index ^shapes-list)))
      )
      point-list     (nth 5 (nth ^index ^shapes-list))
      weight-list    (nth 6 (nth ^index ^shapes-list))))
)
(define-method get-input-shapes shape-collection ()
  (children (the) :class 'shape-model))
;=====
; Reads each line of the shapes file and adds this to a list
;=====
(defun read-shapes-from-file (stream)
  (when stream
    (loop for line = (read-line stream nil :eof)
      until (equal line :eof)
      for ls = (string-to-delimited-token-list line :delimiter #\tab :string-token? nil)
      for shape-data = (list
        (read-from-string (nth 0 ls)) (nth 1 ls) (nth 2 ls) (read-from-string (nth 3 ls))
        (nth 4 ls) (nth 5 ls) (nth 6 ls))
      collect shape-data)))
;=====
; END shape-collection definitions
;=====
; Class: link-collection

```



```

; Creates all the links in the object tree
;=====
(define-class link-collection
  :inherit-from (series-object)
  :properties (
    constraints-list nil ;;List of all master-joint-models in the mechanism
    input-shapes-list nil ;;List of all initial shape-models
    (cross-section-type :class 'option-property-class
      label "Cross-section Type"
      mode 'menu
      formula (nth 0 !options-list)
      ;Returns all classes that inherits from cross-section-model, i.e. all cross-sections
      options-list (reverse (class-direct-defined-subclasses 'cross-section-model))
      labels-list (loop for option in !options-list
        collect (remove "-section" (write-to-string option))))
    common-width 0.04
    common-height 0.04
    property-objects-list (list
      (list (the superior cross-section-type self)
        '(automatic-apply? t)
        '("Set all cross-sections" (button1-parameters :set-c button3-parameters :unset)
          ui-work-area-action-button-class)
        (list (the superior common-width self)
          '(automatic-apply? t)
          (list (the superior common-height self)
            '(automatic-apply? t)
            '("Set all dimensions" (button1-parameters :set-d button3-parameters :unset)
              ui-work-area-action-button-class)
            ""
            '("Draw all link members" (button1-parameters :draw-without-mesh
              button3-parameters :draw-without-mesh)
              ui-work-area-action-button-class)
            ""
            '("Undraw" (button1-parameters :undraw button3-parameters :unset)
              ui-work-area-action-button-class)
            )
      )
    ;Returns a sorted list of all link numbers in the mechanism. I.e: (0 1 2 3 4)
    link-list (sort (remove nil (copy-seq (remove-duplicates (append-list
      (loop for constraint in ^constraints-list
        collect (the link-incidence (:from constraint))))))))) '<

```

```

    (init-default-shape :class 'shape-model
      )
  default-shape (let(
    (def (loop for shape in ^input-shapes-list
      when (equal 'default (the link-ref (:from shape)))
      do (return shape))))
    (if (def def ^init-default-shape)
      shape-list (remove-duplicates (loop for kid in (the shapes-ref-list)
        collect (list (the label (:from kid)) (the link-ref (:from kid)))))
      opt-object (default nil)
      quantity (length ^link-list)
      class-expression 'link-model-class
      series-prefix 'link
      init-form '(
        link-index (nth ^index ^^link-list)
        label (let(
          (shape-name (loop for shape in ^shape-list
            if (equal ^index (nth 1 shape))
            do (return (nth 0 shape)))))
          (if (equal shape-name nil)
            (concatenate "Link-" (write-to-string !index))
            (concatenate (write-to-string !index) " "
              (write-to-string shape-name))))
          constraints-incident-on-link-list (loop for constraint in ^constraints-list
            for con = (get-constraint-incidence constraint
              (the superior))
            when con collect con)
          optimization-object (if ^opt-object
            (loop for link-index in (get-affected-links ^opt-object)
              do
                (if (= link-index ^index)
                  (return ^opt-object))))))
        (define-method get-links link-collection ()
          (children (the) :class 'link-model-class))
          ;=====
          ; Gets the link reference from a link-collection
          ;=====
          (define-method get-link-ref link-collection (link-index)
            (nth (position link-index !link-list) ^link-ref-list)
            )

```

```

(define-method get-all-surface-meshes-list link-collection ()
  (loop for link in (get-links (the))
        append (get-link-surface-mesh-elements-query-objects-list (get-mesh-model-object link))))
;=====
; Button actions for drawing/undrawing links (with and without mesh),
; setting cross-section-type and setting dimensions
;=====
(define-method work-area-button1-action link-collection (params)
  (case params
    (:set-c
     (loop for l in ^link-ref-list do
           (loop for s in (the members-ref-list (:from (the link-geometry (:from l)))) do
                 (change-value (the cross-section-type self (:from s)) !cross-section-type)))
    (:set-d
     (loop for l in ^link-ref-list
           do (loop for s in (the members-ref-list (:from (the link-geometry (:from l))))
                   do (progn
                       (change-value (the width self (:from s)) !common-width)
                       (change-value (the height self (:from s)) !common-height)
                       (change-value (the width-end self (:from s)) !common-width)
                       (change-value (the height-end self (:from s)) !common-height))))
    (:draw-without-mesh
     (loop for link in ^link-ref-list
           do (loop for member in (the visible-members-ref-list (:from (the link-geometry (:from link))))
                   do (draw member :draw-subobjects? nil))))
    (:undraw
     (undraw self))))
;=====
; END link-collection definitions
;=====
; Class: spring-damper-collection
;=====
(define-class spring-damper-collection
  :inherit-from (series-object collection-class)
  :properties (
    points-list    nil
    links-list     nil
    collection-type 'spring-damper
    sd-list        (cdr (read-from-file !superior)) ;Called on the collection-class

```

```

quantity      (length ^sd-list)
class-expression '(read-from-string (format nil "~a-model"
                                       (nth 0 (nth !index !sd-list))))

init-form '(
  label      (format nil "~a" (nth 0 (nth ^index ^sd-list)))
  start-point-data-model (nth (nth 1 (nth ^index ^sd-list)) ^points-list)
  end-point-data-model (nth (nth 2 (nth ^index ^sd-list)) ^points-list)
  ground-point  (if (nth 3 (nth ^index ^sd-list))
                   (nth (nth 3 (nth ^index ^sd-list)) ^points-list)
                   nil)
  incident-links (let (
                    link1 (if (first (nth 3 (nth ^index ^sd-list)))
                              (nth (first (nth 3 (nth ^index ^sd-list)))
                                    ^links-list))
                    link2 (if (second (nth 3 (nth ^index ^sd-list)))
                              (nth (second (nth 3 (nth ^index ^sd-list)))
                                    ^links-list)))
                    (remove nil (list link1 link2)))
  stiffness-damping (nth 4 (nth ^index ^sd-list))))

(define-method get-springs spring-damper-collection ()
  (children (the :class 'spring-model))
)
(define-method get-dampers spring-damper-collection ()
  (children (the :class 'damper-model))
)
;=====
; Reads each line of the spring-damper file and adds this to a list
;=====
(defun read-spring-damper-from-file (stream)
  (when stream
    (loop for line = (read-line stream nil :eof)
          until (equal line :eof)
          for ls = (string-to-delimited-token-list line :delimiter #\tab :string-token? nil)
          for data = (list (read-from-string (nth 0 ls)) (nth 1 ls) (nth 2 ls) (nth 3 ls))
          collect data)))
;=====
; END spring-damper-collection definitions
;=====
; Class: load-collection
; Class to add loads as subobjects (load-model)
;=====

```

```

(define-class load-collection
  :inherit-from (series-object collection-class)
  :properties (
    mech-size      (default nil)
    points-list    (default nil)
    links-list     (default nil)
    collection-type 'loads
    loads-list     (cdr (read-from-file !superior)) ;;Called on the collecion-class
    quantity       (length ^loads-list)
    class-expression 'load-model
    series-prefix  'load
    init-form '(
      type          (nth 0 (nth ^index ^loads-list))
      load-point-object (nth (nth 1 (nth ^index ^loads-list)) ^points-list)
      direction      (nth 2 (nth ^index ^loads-list))
      magnitude      (if (numberp (nth 3 (nth ^index ^loads-list)))
                          (nth 3 (nth ^index ^loads-list))
                          nil)
      scale-load     (unless ^magnitude (read-from-string (remove "scale"
                                                                    (format nil "~a" (nth 3 (nth ^index ^loads-list))))))
      loaded-link    (nth (nth 4 (nth ^index ^loads-list)) ^^links-list)
      mechanism-size ^mech-size)))

(define-method get-loads load-collection ()
  (children (the :class 'load-model)))

;=====
; Reads each line of the loads file and adds this to a list
;=====

(defun read-loads-from-file (stream)
  (when stream
    (loop for line = (read-line stream nil :eof)
          until (equal line :eof)
          for ls = (string-to-delimited-token-list line :delimiter #\tab :string-token? nil)
          for data = (list (nth 0 ls) (nth 1 ls) (nth 2 ls) (nth 3 ls) (nth 4 ls))
          collect data)))

;=====
; END load-collection definitions
;=====

; Class: folder-collection
; Reads all subfolders of the library folder in the file system and

```

```

; creates a subobject for each folder as a series-object of the type folder-info-model
;=====
(define-class folder-collection
  :inherit-from (series-object)
  :properties (
    ;; Removing ../ and ./
    ;; Returns a list of the contents in the library directory
    library-subfolder-list (rest (rest (directory #mechanism-library#)))
    quantity (length ^library-subfolder-list)
    class-expression 'folder-info-model
    series-prefix 'folder
    init-form '(
      path (nth ^index ^^library-subfolder-list)
      )))
;=====
; END folder-collection definitions
;=====
; Class: folder-info-model
; Defines the data-model for folders
;=====
(define-class folder-info-model
  :inherit-from (series-object)
  :properties (
    path nil
    folder (subseq (remove #MECHANISM-LIBRARY# ^path) 1)
    label (replace (copy-seq ^folder) " " :start1 (position "-" ^folder :test 'string-equal))
    class-name (let (
      (name (read-from-string (concatenate ^folder "-class" )))
      (when (find-class name)
        name)
      )
      )
    version-list (rest (rest (directory ^path)))
    class-expression 'mechanism-version-info
    series-prefix 'version
    quantity (length ^version-list)
    init-form '(
      version-path (nth ^index ^^version-list)
      version-name (subseq (remove ^^path ^version-path) 1)))
;=====

```

```

; END folder-info-model definitions
;=====
;=====
; Class: mechanism-version-info
;=====
(define-class mechanism-version-info
  :inherit-from (object)
  :properties (
    version-path nil
    version-name nil
  ))
(define-method get-version-path mechanism-version-info ()
  !version-path)
;=====
; END mechanism-version-info definitions
;=====
(defun get-new-dv-coords-from-line (fixed-point dv1 dv2 direction)
  (if (= (nth 0 direction) 1)
    (list
      (+ (nth 0 dv1) (* (/ (- (nth 0 fixed-point) (nth 0 dv1)) (- (nth 1 fixed-point) (nth 1 dv1)))
        (- (nth 1 dv2) (nth 1 dv1))))
      (nth 1 dv2)
      (nth 2 dv2))
    (list
      (nth 0 dv2)
      (+ (nth 1 dv1) (* (/ (- (nth 1 fixed-point) (nth 1 dv1)) (- (nth 0 fixed-point) (nth 0 dv1)))
        (- (nth 0 dv2) (nth 0 dv1))))
      (nth 2 dv2))))
;=====
; Class: main-mechanism-class
; Initiates the system by creating all the collections as subobjects
;=====
(define-class main-mechanism-class
  :inherit-from (series-object)
  :properties (
    point-ref-list (children ^points :class 'point-data-model)
    constraints-ref-list (children ^constraints :class 'master-joint-model)
    shapes-ref-list (children ^shapes :class 'shape-model)
    link-ref-list (children ^links :class 'link-model-class)
    final-link-ref-list (loop for link in ^link-ref-list

```

```

        if (are-all-members-displayed? link)
            collect link)
; mechanism-selection contains a list of folder-info-models to choose from
(mechanism-selection :class 'option-property-class
  labels-list (loop for subfolder in (children ^folders :class 'folder-info-model)
    when (the path (:from subfolder))
    collect (the label (:from subfolder)))
  options-list (children ^folders :class 'folder-info-model)
  mode 'menu
  formula (nth (position "four bar" !labels-list) !options-list)
  label "Select Mechanism")
(version-selection :class 'option-property-class
  options-list (children ^^mechanism-selection :class 'mechanism-version-info)
  labels-list (loop for version in !options-list
    collect (the version-name (:from version)))
  mode 'menu
  formula (nth 0 !options-list)
  label "Select Version")
property-objects-list (list
  (list (the superior mechanism-selection self)
    '(automatic-apply? t))
  (list (the superior version-selection self)
    '(automatic-apply? t))
  ""
  '("Draw mechanism" (button1-parameters :draw-sewn-wo-mesh)
  ui-work-area-action-button-class)
  ""
  '("Draw mechanism with 3D mesh"
    (button1-parameters :draw-sewn-with-tet-mesh)
    ui-work-area-action-button-class)
  '("Draw mechanism with 2D mesh"
    (button1-parameters :draw-sewn-with-tri-mesh)
    ui-work-area-action-button-class)
  ""
  '("Undraw" (button1-parameters :undraw button3-parameters :unset)
    ui-work-area-action-button-class)
  ""
  '("Create .bdf-files" (button1-parameters :create-bdf)
    ui-work-area-action-button-class)
  '("Create .stl-files" (button1-parameters :create-stl)

```



```

        ui-work-area-action-button-class
        '( "Export to FEDEM" (button1-parameters :export-fedem)
          ui-work-area-action-button-class
          )
;;Property storing folders from library
(folders :class 'folder-collection)
mechanism-type (the folder (:from ^mechanism-selection))
mechanism-version (format nil "~a--a" ^mechanism-type (the version-name
  (:from ^version-selection)))
(shapes :class 'shape-collection)
;; Optimization defintions. If the optimization.txt file is empty, no optimization models
;; will be added
opt-file-path (concatenate (get-version-path ^version-selection) "\\optimization.txt")
opt-list      (cdr (read-opt-from-file !superior))
class-expression '(read-from-string (format nil "~a-optimization-model"
  (nth 0 (nth !index !opt-list))))
quantity      (length ^opt-list)
init-form '(
  affected-links      (nth 1 (nth ^index ^opt-list))
  init-values         (nth 2 (nth ^index ^opt-list))
  constraint-type     (nth 3 (nth ^index ^opt-list))
  max-allowed-deformation (nth 4 (nth ^index ^opt-list))
  load-objects        (get-loads ^^loads)
  links               (get-links ^^links)
  current-path        (get-version-path ^^version-selection)
  main-mech-ref       (the superior superior)
  label               (format nil "~a-optimization"
    (nth 0 (nth ^index ^opt-list))))
:subjects (
  (constraints :class 'constraint-collection)
  links-list (get-links ^links)
  (points :class 'point-collection)
  (links :class 'link-collection
    constraints-list (get-constraints ^constraints)
    input-shapes-list (get-input-shapes ^shapes)
    opt-object (get-opt-object (the superior superior))
    (spring-dampers :class 'spring-damper-collection)
    points-list (get-points ^^points)
    links-list (get-links ^^links))
  (loads :class 'load-collection)

```

```

    mech-size (calculate-mechanism-size ^superior)
    points-list (get-points ^^points)
    links-list (get-links ^^links)))

;=====
; Reads optimization definitions from optimization.txt and stores the data in a list
;=====

(define-method read-opt-from-file main-mechanism-class ()
  (if (probe-file !opt-file-path)
      (with-open-file (file !opt-file-path :direction :input)
        (loop for line = (read-line file nil :eof)
              until (equal line :eof)
              for ls = (string-to-delimited-token-list line :delimiter #\tab :string-token? nil)
              for data = (list (nth 0 ls) (nth 1 ls) (nth 2 ls) (nth 3 ls) (nth 4 ls))
              collect data))
      (progn
        (message (format nil "~a" is not a valid file path." !opt-file-path) :append? t)
        nil ;;Return value if the file does not exist)))

;=====
; Returns a general-optimization-model, if there is any. Assumes max one optimization object
;=====

(define-method get-opt-object main-mechanism-class ()
  (first (children (the :class 'general-optimization-model))))

;=====
; Returns the greatest distance within the mechanism
;=====

(define-method calculate-mechanism-size main-mechanism-class ()
  (let (
    (max-x 0)
    (max-y 0)
    (max-z 0))
    (loop for point in (get-points !points) do
      (progn
        (if (> (abs (nth 0 (get-coordinates point))) max-x)
            (setf max-x (abs (nth 0 (get-coordinates point))))
        )
        (if (> (abs (nth 1 (get-coordinates point))) max-y)
            (setf max-y (abs (nth 1 (get-coordinates point))))
        )
        (if (> (abs (nth 2 (get-coordinates point))) max-z)
            (setf max-z (abs (nth 2 (get-coordinates point))))))))))

```

```

    finally (return (sqrt (+ (expt max-x 2) (expt max-y 2) (expt max-z 2))))))
(define-method reset-all-database-values main-mechanism-class ()
  (loop for link in (get-links !links) do
    (smash-value (the db-id (:from (get-mesh-database link))))))
(define-method get-all-joint-elements main-mechanism-class ()
  (remove nil
    (loop for constraint in (get-constraints !constraints)
      append (list (get-male-element constraint) (get-female-element constraint))))))
;=====
; Assigns a unique ID (starting from 1) to each joint element in the mechanism
;=====
(define-method get-joint-ID main-mechanism-class (joint-element)
  (1+ (position joint-element (get-all-joint-elements (the))))))
(define-method get-constraint main-mechanism-class (constraint-number)
  (nth constraint-number !constraints-ref-list))
(define-method get-mechanism-type main-mechanism-class ()
  (the mechanism-selection label))
(define-method draw-sewn-wo-mesh main-mechanism-class ()
  (loop for link in !final-link-ref-list do
    (with-error-handler (:error-message (concatenate "Error drawing "
      (write-to-string (object-name link))) :show-system-error? t)
      (draw (the link-geometry (:from link) :draw-subobjects? nil)) t))
(define-method draw-sewn-with-tet-mesh main-mechanism-class ()
  (loop for link in !final-link-ref-list do
    (with-error-handler (:error-message (concatenate "Error drawing "
      (write-to-string (object-name link))) :show-system-error? t)
      (draw (first (get-link-solid-mesh-elements-query-objects-list
        (get-mesh-model-object link))))))
(define-method draw-sewn-with-tri-mesh main-mechanism-class ()
  (loop for link in !final-link-ref-list do
    (with-error-handler (:error-message (concatenate "Error drawing "
      (write-to-string (object-name link))) :show-system-error? t)
      (draw (first (get-link-surface-mesh-elements-query-objects-list
        (get-mesh-model-object link))))))
;=====
; Creates buttons to draw and export the mechanism
;=====
(define-method work-area-button1-action main-mechanism-class (params)
  (case params
    (:draw-sewn-wo-mesh

```

```

    (draw-sewn-wo-mesh (the)))
  (:draw-sewn-with-tet-mesh
    (draw-sewn-with-tet-mesh (the)))
  (:draw-sewn-with-tri-mesh
    (draw-sewn-with-tri-mesh (the)))
  (:undraw
    (undraw self))
  (:create-bdf
    (if (equal "2D" (pop-up-message (format nil "Export to .bdf")
      :width 200 :done-label "2D" :cancel-label "3D")); The pop-up-message returns the label
      (loop for link in !final-link-ref-list do
        (change-value (the analysis export-surface? (:from link)) t)
        (loop for link in !final-link-ref-list do
          (change-value (the analysis export-surface? (:from link)) nil)))
      (write-nastran-bdf-files (the)))
    (:create-stl
      (write-stl-files (the) (get-version-path !version-selection)))
    (:export-fedem
      (write-nastran-bdf-files (the))
      (write-fmm-model-file (the))))
;=====
; END main-mechanism-class definitions
;=====

```

## F.15 Geometry-export.aml

```

;=====
; Writes each link to a nastran .bdf file, stored in :nastran-data
;=====
(define-method write-nastran-bdf-files main-mechanism-class ()
  (loop for link in !final-link-ref-list do
    (write-nastran-bdf-file (the) link)))
;=====
; Writes each link to a .stl file
;=====
(define-method write-stl-files main-mechanism-class (path)
  (let (
    (corrected-path (concatenate (replace-all-in-string "/" "\\\" path) "/Stl-files/"))
  )

```

```

    (if (not (probe-file corrected-path))
        (create-directory corrected-path)
    )
    (loop for link in !final-link-ref-list do
        (write-stl-file (the link-geometry (:from link)) (concatenate corrected-path
            (the label (:from link)) ".stl"))
    )))
;=====
; Replaces all specified characters in string
;=====
(defun replace-all-in-string (new old string)
    (if (equal string (replace-in-string new old string))
        string
        (replace-all-in-string new old (replace-in-string new old string))))
;=====
; Writes one link to a nastran .bdf file , stored in :nastran-data
;=====
(define-method write-nastran-bdf-file main-mechanism-class (link)
    (the analysis nastran-interface run-nastran@ (:from link)))
;=====
; Writes a fedem fmm file, stored in library\'mechanism-type\'"version"
;=====
(define-method write-fmm-model-file main-mechanism-class (&key from-opt?)
    (let (
        (baseID      2)
        (linkID      0)
        (conID       0)
        (triads-written 1)
        (triad-fe-node-list nil)
    )
        (with-open-file (stream (logical-path (the version-path (:from !version-selection))
            (concatenate "model" ".fmm")))
            :direction :output
            :if-exists :overwrite
        )
        (progn
            (write-static-top-part-to-fmm stream)
            (loop for link in !final-link-ref-list do
                (setf baseID (1+ baseID))
                (setf linkID (1+ linkID))
            )
        )
    )

```

```

(format stream "LINK~%")
(format stream "{~%")
(format stream "BASE_ID = ~d;~%" baseID)
(format stream "COORDINATE_SYSTEM = ~%")
(format stream "1.00000000 0.00000000 0.00000000 0.00000000~%")
(format stream "0.00000000 1.00000000 0.00000000 0.00000000~%")
(format stream "0.00000000 0.00000000 1.00000000 0.00000000;~%")
(format stream "ID = ~d;~%" linkID)
(format stream "LINE_COLOR = 1 1 1;~%")
(format stream "MASS_PROP_DAMP = 0;~%")
(format stream "ORIGINAL_FE_FILE = ~a;~%" (concatenate ""\
    (logical-path :nastran-data)
    (the analysis nastran-interface model-name (:from link)) "\\"
    (the analysis nastran-interface nastran-file-name (:from link)) "\\"))
)
(format stream "POLYS_ON_POINTS_OFF = true;~%")
(format stream "STIF_PROP_DAMP = 0;~%")
(format stream "USE_MASS_CALCULATION = true;~%")
(format stream "}~3%")
(let (
    (number-of-nodes (length (get-mesh-entities-list (get-mesh-model-object link))))
    (counter 0)
)
    ;; For each non-free joint element on a link, we need to write a triad connected to
    ;; the joint element's RBE2 FE node.
    (loop for joint-element in (get-joint-elements-on-link link) do
        (setf baseID (1+ baseID))
        (setf counter (1+ counter))
        (write-triad-to-fmm stream baseID (get-joint-ID (the joint-element)
            (get-rbe2-independent-node-coordinates joint-element)
            linkID (+ number-of-nodes counter)))
        (setf triad-fe-node-list (append triad-fe-node-list (list (list
            (get-joint-ID (the joint-element) (+ number-of-nodes counter) linkID))))))
        (setf triads-written (1+ triads-written))))))
(loop for constraint in !constraints-ref-list do
    (setf baseID (1+ baseID))
    (setf conID (1+ conID))
    (let* (
        (male-element (get-male-element constraint))
        (female-element (get-female-element constraint))

```

```

)
;;Case 1: male element exists, female does not.
;;Have to write a ground triad to connect to the male element
;;Master triad is the ground, slave triad is the male element
(if (and male-element (not female-element))
  (progn
    (write-joint-type-to-fmm stream constraint baseID conID triads-written
      (get-joint-ID (the) male-element))
    (setf baseID (1+ baseID))
    (write-triad-to-fmm stream baseID triads-written
      (get-rbe2-independent-node-coordinates male-element) -1 -1) ;;In FEDEM,
      ;;-1 means that it's fixed to the ground
    (setf triads-written (1+ triads-written))
  )
)
;;Case 2: female element exists, male does not
;;Have to write a ground triad to connect to the female element
;;Master triad is the ground, slave triad is the female element
(if (and female-element (not male-element))
  (progn
    (write-joint-type-to-fmm stream constraint baseID conID triads-written
      (get-joint-ID (the) female-element))
    (setf baseID (1+ baseID))
    (write-triad-to-fmm stream baseID triads-written
      (get-rbe2-independent-node-coordinates female-element) -1 -1)
    (setf triads-written (1+ triads-written))
  )
)
;;Case 3: both elements exists
;;Master triad is the female element, slave triad is the male element (ofc)
;;No extra ground triad has to be created
(write-joint-type-to-fmm stream constraint baseID conID
  (get-joint-ID (the) female-element) (get-joint-ID (the) male-element))
))))
(loop for spring-damper in (children !spring-dampers) do
  (let (
    (counter 0)
    (sd-triad-fe-node-list nil)
  )
    (progn
      ;; Mesh-node is the node the spring/damper is connected to.
      (loop for mesh-node in (children spring-damper) do

```

```

(setf counter (1+ counter))
(let (
  (owner-linkID (1+ (the index (:from (get-owner-link mesh-node))))))
  (found-node? nil))
  (progn
    ;; Have to check if a triad for the given fe node has been written
    ;; already. If it has, no extra triad is needed.
    (loop for tuple in triad-fe-node-list do
      (when (and (= (get-node-position mesh-node) (nth 1 tuple))
                 (= owner-linkID (nth 2 tuple)))
        (setf sd-triad-fe-node-list (append sd-triad-fe-node-list
                                             (list (nth 0 tuple))))
        (setf found-node? t))
      (when (not found-node?)
        (setf baseID (1+ baseID))
        (write-triad-to-fmm stream baseID triads-written
                           (get-coordinates-for-node-id (get-mesh-database
                                                         (get-owner-link mesh-node) (get-node-position mesh-node))
                                                         (1+ (the index (:from (get-owner-link mesh-node))))
                                                         (get-node-position mesh-node))
                           (setf triads-written (1+ triads-written))))))
    (if (= 1 counter)
        ;; Happens if spring/damper only has one subobject. This means that it is
        ;; supposed to be connected to the ground. Have to create this ground triad
        (progn
          (setf baseID (1+ baseID))
          (write-triad-to-fmm stream baseID triads-written
                               (get-end-point spring-damper) -1 -1)
          (setf triads-written (1+ triads-written))))
        (setf baseID (1+ baseID))
        (let (
          (connecting-triads (if (= 2 (length sd-triad-fe-node-list))
                                sd-triad-fe-node-list
                                (if (= 1 (length sd-triad-fe-node-list))
                                    (list (- triads-written 1) (first sd-triad-fe-node-list))
                                    (list (- triads-written 1) (- triads-written 2))))))
            (case (get-type spring-damper)
              ('spring
               (write-spring-properties-to-fmm stream baseID
                                                (1+ (the index (:from spring-damper))) (get-stiffness

```



```

        spring-damper)
      (first connecting-triads) (second connecting-triads))
    ('damper
      (write-damper-properties-to-fmm stream baseID
        (1+ (the index (:from spring-damper))) (get-damping
          spring-damper)
        (first connecting-triads) (second connecting-triads))))))
;; Loads
(loop for load in (get-loads !loads) do
  (setf baseID (1+ baseID))
  (let (
    (loaded-node (get-loaded-node load))
    (load-triad-fe-node nil)
  )
    (progn
      (loop for tuple in triad-fe-node-list do
        (if (and (= (get-node-position loaded-node) (nth 1 tuple))
          (= (1+ (the index (:from (get-owner-link loaded-node)))
            (nth 2 tuple)))
          (setf load-triad-fe-node (nth 0 tuple))))
        (if load-triad-fe-node
          (write-loads-to-fmm stream baseID (get-load-ID load) load-triad-fe-node
            (get-load-type load) (get-magnitude load) (get-scale-load load)
            (get-direction load))
          (progn
            (write-triad-to-fmm stream baseID triads-written
              (get-coordinates-for-node-id (get-mesh-database
                (get-owner-link loaded-node)) (get-node-position loaded-node))
              (1+ (the index (:from (get-owner-link loaded-node)))
                (get-node-position loaded-node))
              )
            (setf triads-written (1+ triads-written))
            (setf baseID (1+ baseID))
            (write-loads-to-fmm stream baseID (get-load-ID load) (1- triads-written)
              (get-load-type load) (get-magnitude load) (get-scale-load load)
              (get-direction load))))))
    (when (get-scale-load load)
      (progn
        (setf baseID (1+ baseID))
        (write-engines-to-fmm stream baseID 1 "1 FcSCALE" "1 FcTIME_SENSOR")

```

```

      (setf baseID (1+ baseID))
      (write-function-to-fmm stream baseID 1 (get-scale-load load))))))
(if (equal "four-bar-i-beam" !mechanism-version)
    (progn
      (let (
          (x-result "<\\"SCALAR\\",\\"Physical time\\">")
          (x-oper "\\None\\"")
          (y-result "<\\"Triad\\",8,5,\\\"VEC3\\",\\"Deformational displacement\\">")
          (y-object "5 FcTRIAD")
          (y-oper "\\Length\\"")
          (legend "\\Triad [5], Deformational displacement, Length vs Time\\""))
        )
      (progn
        (setf baseID (1+ baseID))
        (write-curve-sets-to-fmm stream baseID 1 1 x-result x-oper y-result
          y-object y-oper :legend legend)
        ))
      (setf baseID (1+ baseID))
      (write-graph-definitions-to-fmm stream baseID 1 "\\Deformation\\""))
    ))
(format stream "END {FEDEMMODELFILE}")
(unless from-opt?
  (message-box "Success!" (format nil "Writing to fmm successful!") :mode :ok)
  ))))
;=====
; Below are self-explanatory helping functions for writing to fmm
;=====
(defun write-static-top-part-to-fmm (stream)
  (progn
    (format stream "FEDEMMODELFILE {R7.0.4 ASCII}~2%")
    (format stream "GLOBAL_VIEW_SETTINGS~%{~%}")
    (format stream "ID = 1;~%")
    (format stream "SYMBOL_SCALE = 0.1;~%")
    (format stream "SYMBOL_LINE_WIDTH = 1;~%")
    (format stream "BACKGROUND_COLOR = 0.098039 0.305882 0.458823;~%")
    (format stream "CAMERA_FOCAL_DIST = 0.707107;~%")
    (format stream "CAMERA_HEIGHT = 1.41421;~%")
    (format stream "CAMERA_ORIENTATION =~%")
    (format stream "1.00000000 0.00000000 0.00000000 0.00000000~%")
    (format stream "0.00000000 1.00000000 0.00000000 0.00000000~%")
  ))

```

```

(format stream "0.00000000 0.00000000 1.00000000 0.70710678;~%}-2%")
(format stream "MECHANISM~%{~%}")
(format stream "ID = 1;~%")
(format stream "BASE_ID = 1;~%") ;;Manually setting "static" base ID
(format stream "GRAVITY = 0 0 -9.81;~%") ;;When gravity changes, let me know. Also update this
    value.
(format stream "POSITION_TOLERANCE = 0.0001;~%}-2%")
(format stream "REF_PLANE~%{~%}")
(format stream "ID = 1;~%")
(format stream "BASE_ID = 2;~%") ;;Manually setting "static" base ID
(format stream "HEIGHT = 0.1;~%")
(format stream "WIDTH = 0.1;~%")
(format stream "COLOR = 1 1 1;~%")
(format stream "TRANSPARENCY = 0.65;~%}-3%"))
(defun write-triad-to-fmm (stream baseID ID coords linkID node-number)
  (format stream "TRIAD~%{~%}")
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "ID = ~d;~%" ID)
  (format stream "COORDINATE_SYSTEM = ~%")
  (format stream "1.0 0.0 0.0 ~d~%" (nth 0 coords))
  (format stream "0.0 1.0 0.0 ~d~%" (nth 1 coords))
  (format stream "0.0 0.0 1.0 ~d;~%" (nth 2 coords))
  (format stream "LOCAL_DIRECTIONS = GLOBAL;~%")
  (format stream "LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;~%")
  (format stream "OWNER_LINK = ~d;~%" linkID)
  (format stream "FE_NODE_NO = ~d;~%" node-number)
  (format stream "NDOFS = 6;~%}-3%"))
(defun write-joint-type-to-fmm (stream constraint baseID conID masterID slaveID)
  (if (equal "revolute" (get-constraint-type constraint))
      (let (
        (unit-vector (get-unit-vector constraint))
        (rot-angle (get-rot-angle constraint))
        (cos-rot-angle (cosd rot-angle))
        (sin-rot-angle (sind rot-angle))
        )
        (progn
          (format stream "~a~%{~%" "REV_JOINT")
          (format stream "BASE_ID = ~d;~%" baseID)
          ;;Calculating rotation matrix with respect to rotation angle and rotation axis
          (format stream "COORDINATE_SYSTEM = ~%")

```

```

(format stream "~d ~d ~d 0.0~%"
  (+ cos-rot-angle (* (expt (nth 0 unit-vector) 2) (- 1 cos-rot-angle)))
  (- (* (nth 0 unit-vector) (nth 1 unit-vector) (- 1 cos-rot-angle))
    (* (nth 2 unit-vector) sin-rot-angle))
  (+ (* (nth 0 unit-vector) (nth 2 unit-vector) (- 1 cos-rot-angle))
    (* (nth 1 unit-vector) sin-rot-angle)))
(format stream "~d ~d ~d 0.0~%"
  (+ (* (nth 1 unit-vector) (nth 0 unit-vector) (- 1 cos-rot-angle))
    (* (nth 2 unit-vector) sin-rot-angle))
  (+ cos-rot-angle
    (* (expt (nth 1 unit-vector) 2) (- 1 cos-rot-angle)))
  (- (* (nth 1 unit-vector) (nth 2 unit-vector) (- 1 cos-rot-angle))
    (* (nth 0 unit-vector) sin-rot-angle)))
(format stream "~d ~d ~d 0.0;~%"
  (- (* (nth 2 unit-vector) (nth 1 unit-vector) (- 1 cos-rot-angle))
    (* (nth 1 unit-vector) sin-rot-angle))
  (+ (* (nth 2 unit-vector) (nth 1 unit-vector) (- 1 cos-rot-angle))
    (* (nth 0 unit-vector) sin-rot-angle))
  (+ cos-rot-angle (* (expt (nth 2 unit-vector) 2) (- 1 cos-rot-angle))))
(format stream "HAS_Z_TRANS_DOF = false;~%")
(format stream "ID = ~a;~%" conID)
(format stream "LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;~%")
(format stream "MASTER_TRIAD = ~d;~%" masterID)
(format stream "MOVE_MASTER_TRIAD_ALONG = false;~%")
(format stream "MOVE_SLAVE_TRIAD_ALONG = false;~%")
(format stream "ROT_FORMULATION = FOLLOWER_AXIS;~%")
(format stream "ROT_SEQUENCE = ZYX;~%")
(format stream "ROT_SPRING_CPL = NONE;~%")
(format stream "SLAVE_TRIAD = ~d;~%" slaveID)
(format stream "TRAN_SPRING_CPL = NONE;~%")
(format stream "VAR_QUADRANTS = 0 0 0;~%")
(format stream "Z_ROT_STATUS = FREE;~%")
(format stream "Z_TRANS_STATUS = FREE;~%}~3%")
)
)
(if (equal "ball" (get-constraint-type constraint))
  (let (
    (unit-vector (get-unit-vector constraint))
    (rot-angle (get-rot-angle constraint))
    (cos-rot-angle (cosd rot-angle))

```

```

(sin-rot-angle (sind rot-angle))
)
(progn
  (format stream "~a-%{~%" "BALL_JOINT")
  (format stream "BASE_ID = ~d;~%" baseID)
  ;; Calculating rotation matrix with respect to rotation angle and rotation axis
  (format stream "COORDINATE_SYSTEM = ~%")
  (format stream "~d ~d ~d 0.0~%"
    (+ cos-rot-angle (* (expt (nth 0 unit-vector) 2) (- 1 cos-rot-angle)))
    (- (* (nth 0 unit-vector) (nth 1 unit-vector) (- 1 cos-rot-angle)
        (* (nth 2 unit-vector) sin-rot-angle))
    (+ (* (nth 0 unit-vector) (nth 2 unit-vector) (- 1 cos-rot-angle)
        (* (nth 1 unit-vector) sin-rot-angle)))
  (format stream "~d ~d ~d 0.0~%"
    (+ (* (nth 1 unit-vector) (nth 0 unit-vector) (- 1 cos-rot-angle)
        (* (nth 2 unit-vector) sin-rot-angle))
    (+ cos-rot-angle
        (* (expt (nth 1 unit-vector) 2) (- 1 cos-rot-angle)))
    (- (* (nth 1 unit-vector) (nth 2 unit-vector) (- 1 cos-rot-angle)
        (* (nth 0 unit-vector) sin-rot-angle)))
  (format stream "~d ~d ~d 0.0;~%"
    (- (* (nth 2 unit-vector) (nth 1 unit-vector) (- 1 cos-rot-angle)
        (* (nth 1 unit-vector) sin-rot-angle))
    (+ (* (nth 2 unit-vector) (nth 1 unit-vector) (- 1 cos-rot-angle)
        (* (nth 0 unit-vector) sin-rot-angle))
    (+ cos-rot-angle (* (expt (nth 2 unit-vector) 2) (- 1 cos-rot-angle))))
  (format stream "FRICTION_DOF = 3;~%")
  (format stream "ID = ~d;~%" conID)
  (format stream "LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;~%")
  (format stream "MASTER_TRIAD = ~d;~%" masterID)
  (format stream "MOVE_MASTER_TRIAD_ALONG = false;~%")
  (format stream "MOVE_SLAVE_TRIAD_ALONG = false;~%")
  (format stream "ROT_FORMULATION = FOLLOWER_AXIS;~%")
  (format stream "ROT_SEQUENCE = ZYX;~%")
  (format stream "ROT_SPRING_CPL = NONE;~%")
  (format stream "SLAVE_TRIAD = ~d;~%" slaveID)
  (format stream "TRAN_SPRING_CPL = NONE;~%")
  (format stream "VAR_QUADRANTS = 0 0 0;~%")
  (format stream "X_ROT_STATUS = FREE;~%")
  (format stream "Y_ROT_STATUS = FREE;~%")

```

```

    (format stream "Z_ROT_STATUS = FREE;~%}-3%")
  )
)
(if (equal "free" (get-constraint-type constraint))
  (let* (
    (dofs (get-degrees-of-freedom constraint))
    (dof-list (list (if (position 1 dofs) "FIXED" "FREE")
      (if (position 2 dofs) "FIXED" "FREE")
      (if (position 3 dofs) "FIXED" "FREE")
      (if (position 4 dofs) "FIXED" "FREE")
      (if (position 5 dofs) "FIXED" "FREE")
      (if (position 6 dofs) "FIXED" "FREE"))))
    )
  (progn
    (format stream "FREE_JOINT~%{-%")
    (format stream "BASE_ID = ~a;~%" baseID)
    (format stream "COORDINATE_SYSTEM = ~%")
    (format stream "1.00000000 0.00000000 0.00000000 0.00000000~%")
    (format stream "0.00000000 1.00000000 0.00000000 0.00000000~%")
    (format stream "0.00000000 0.00000000 1.00000000 0.00000000;~%")
    (format stream "FRICTION_DOF = 0;~%")
    (format stream "ID = ~a;~%" conID)
    (format stream "LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;~%")
    (format stream "MASTER_TRIAD = ~d;~%" masterID)
    (format stream "MOVE_MASTER_TRIAD_ALONG = false;~%")
    (format stream "MOVE_SLAVE_TRIAD_ALONG = false;~%")
    (format stream "ROT_FORMULATION = FOLLOWER_AXIS;~%")
    (format stream "ROT_SEQUENCE = ZYX;~%")
    (format stream "ROT_SPRING_CPL = NONE;~%")
    (format stream "SLAVE_TRIAD = ~d;~%" slaveID)
    (format stream "TRAN_SPRING_CPL = NONE;~%")
    (format stream "X_TRANS_STATUS = ~d;~%" (nth 0 dof-list))
    (format stream "Y_TRANS_STATUS = ~d;~%" (nth 1 dof-list))
    (format stream "Z_TRANS_STATUS = ~d;~%" (nth 2 dof-list))
    (format stream "X_ROT_STATUS = ~d;~%" (nth 3 dof-list))
    (format stream "Y_ROT_STATUS = ~d;~%" (nth 4 dof-list))
    (format stream "Z_ROT_STATUS = ~d;~%}-3%" (nth 5 dof-list))
  ))
)
(if (equal "fixed" (get-constraint-type constraint))
  (progn

```

```

(format stream "FREE_JOINT~%{~%}")
(format stream "BASE_ID = ~a;~%" baseID)
(format stream "COORDINATE_SYSTEM = ~%")
(format stream "1.00000000 0.00000000 0.00000000 0.00000000~%")
(format stream "0.00000000 1.00000000 0.00000000 0.00000000~%")
(format stream "0.00000000 0.00000000 1.00000000 0.00000000;~%")
(format stream "FRICTION_DOF = 0;~%")
(format stream "ID = ~a;~%" conID)
(format stream "LOCATION3D_DATA = CART_X_Y_Z EUL_Z_Y_X;~%")
(format stream "MASTER_TRIAD = ~d;~%" masterID)
(format stream "MOVE_MASTER_TRIAD_ALONG = false;~%")
(format stream "MOVE_SLAVE_TRIAD_ALONG = false;~%")
(format stream "ROT_FORMULATION = FOLLOWER_AXIS;~%")
(format stream "ROT_SEQUENCE = ZYX;~%")
(format stream "ROT_SPRING_CPL = NONE;~%")
(format stream "SLAVE_TRIAD = ~d;~%" slaveID)
(format stream "TRAN_SPRING_CPL = NONE;~%")
(format stream "X_ROT_STATUS = FIXED;~%")
(format stream "Y_ROT_STATUS = FIXED;~%")
(format stream "X_TRANS_STATUS = FIXED;~%")
(format stream "Y_TRANS_STATUS = FIXED;~%")
(format stream "Z_TRANS_STATUS = FIXED;~%")
(format stream "Z_ROT_STATUS = FIXED;~%}{~3%")
))))))
(defun write-spring-properties-to-fmm (stream baseID ID stiffness triad1 triad2)
  (format stream "AXIAL_SPRING~%{~%}")
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "ID = ~d;~%" ID)
  (format stream "INIT_LENGTH = 0;~%")
  (format stream "INIT_STIFFNESS = ~d;~%" stiffness)
  (format stream "TRIAD_CONNECTIONS = ~d ~d;~%" triad1 triad2)
  (format stream "USE_INIT_DEFLECTION = true;~%}{~2%")
)
(defun write-damper-properties-to-fmm (stream baseID ID damping triad1 triad2)
  (format stream "AXIAL_DAMPER~%{~%}")
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "ID = ~d;~%" ID)
  (format stream "INIT_DAMPING = ~d;~%" damping)
  (format stream "TRIAD_CONNECTIONS = ~d ~d;~%}{~2%"} triad1 triad2)
)
(defun write-analysis-properties-to-fmm (mechanism-type stream baseID loads-list)
)
(defun write-control-system-properties-to-fmm (stream baseID)
)

```

```

;;EnginesX
(let (
  (descr (list "\Torque\" "\Reference\" "\Joint Velocity\"))
  (sensor (list "3 FcSIMPLE_SENSOR" "1 FcSIMPLE_SENSOR" "2 FcSIMPLE_SENSOR"))
  (math-func (list "1 FcSCALE" "1 FcLIM_RAMP" "1 FcSCALE"))
  (entity (list -1 -1 6))
  (dof (list -1 -1 5))
  (loop for i from 0 to 2 do
    (setf baseID (1+ baseID))
    (write-engines-to-fmm stream baseID (1+ i) (nth i descr) (nth i sensor)
      (nth i math-func) (nth i entity) (nth i dof))
  ))
;;SensorsX
(let (
  (descr (list "\Time sensor\" "\\" "\Control output sensor\""))
  (measured (list "1 FcTIME" "1 FcREV_JOINT" "1 FcOUTPUT"))
  )
  (loop for i from 0 to 2 do
    (setf baseID (1+ baseID))
    (write-sensors-to-fmm stream baseID (1+ i) (nth i descr) (nth i measured))
  ))
;;Function definitionsX
(setf baseID (1+ baseID))
(format stream "FUNC_LIM_RAMP~%{~%}")
(format stream "BASE_ID = ~d;~%" baseID)
(format stream "ID = 1;~%")
(format stream "AMPLITUDE_DISPLACEMENT = 0;~%")
(format stream "SLOPE_OF_RAMP = 12.5664;~%")
(format stream "DELAY_OF_RAMP = 0;~%")
(format stream "END_OF_RAMP = 0.5;~%}~3%")
;;Control linesX
(let (
  (ownerStart (list "1 FcINPUT" "2 FcINPUT" "1 FcCOMPARATOR" "1 FcAMPLIFIER"))
  (ownerEnd (list "1 FcCOMPARATOR 1" "1 FcCOMPARATOR 2" "1 FcAMPLIFIER 1" "1
    FcOUTPUT 1"))
  (ful (list 2 1 2 2))
  (controlNo (list 1 2 4 3))
  )
  (loop for i from 0 to 3 do
    (setf baseID (1+ baseID))
  ))

```



```

    (write-control-lines-to-fmm stream baseID (1+ i)
      (nth i ownerStart) (nth i ownerEnd) (nth i ful) (nth i controlNo))
  ))
;;Control i/oX
(let (
  (name (list "CONTROL_INPUT" "CONTROL_INPUT" "CONTROL_OUTPUT"
    "CONTROL_AMPLIFIER"
    "CONTROL_COMPARATOR"))
  (IDs (list 1 2 1 1 1))
  (engine (list "2 FcENGINE" "3 FcENGINE" "0 0" "0 0" "0 0"))
  (position (list "-3.5 1 0" "-3.5 0 0" "2.5 0.5 0" "0.5 0.5 0" "-1.5 0.5 0"))
  )
  (loop for i from 0 to 4 do
    (setf baseID (1+ baseID))
    (write-control-io-to-fmm stream (nth i name) baseID (nth i IDs)
      (nth i engine) (nth i position) (if (= i 3) 1))
  )))
(defun write-engines-to-fmm (stream baseID ID math-func sensor)
  (format stream "ENGINE-%{~%}"
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "ID = ~d;~%" ID)
  (format stream "MATH_FUNC = ~a;~%" math-func)
  (format stream "SENSOR = ~a;~%~2%" sensor))
(defun write-function-to-fmm (stream baseID ID scale)
  (format stream "FUNC_SCALE-%{~%}"
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "FUNC_USE = GENERAL;~%"
  (format stream "ID = ~d;~%" ID)
  (format stream "SCALE = ~d;~%~2%" scale))
(defun write-sensors-to-fmm (stream baseID ID descr measured)
  (format stream "SENSOR-%{~%}"
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "ID = ~d;~%" ID)
  (format stream "DESCR = ~a;~%" descr)
  (format stream "MEASURED = ~a;~%~2%" measured))
(defun write-loads-to-fmm (stream baseID ID owner-triad load-type magnitude scale-load direction)
  (format stream "LOAD~%{~%}"
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "ID = ~d;~%" ID)
  (if magnitude

```

```

(format stream "INIT_LOAD = ~d;~%" magnitude)
(progn
  (format stream "ENGINE = 1 FcENGINE;~%")
  (format stream "INIT_LOAD = 0;~%")
))
(format stream "OWNER_TRIAD = ~d;~%" owner-triad)
(format stream "LOAD_TYPE = ~d;~%" (case load-type ('force 0) ('torque 1)))
(format stream "SCALE_LOAD = 1;~%")
(format stream "FROM_OBJECT = -1 FcLINK;~%")
(format stream "FROM_POINT = 0 0 0;~%")
(format stream "TO_OBJECT = -1 FcLINK;~%")
(format stream "TO_POINT = ~d ~d ~d;~%~2%" (nth 0 direction) (nth 1 direction) (nth 2 direction)))
(defun write-control-lines-to-fmm (stream baseID controlID ownerStart ownerEnd ful controlNo)
  (format stream "CONTROL_LINE~%{~%}")
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "ID = ~d;~%" controlID)
  (format stream "OWNER_START = ~a;~%" ownerStart)
  (format stream "OWNER_END = ~a;~%" ownerEnd)
  (format stream "FIRST_LINE_VERTICAL = 0;~%")
  (format stream "FIRST_UNDEF_LINE = ~d;~%" ful)
  (format stream "SEGMENT_LENGTHS = 0.25;~%")
  (format stream "CONTROL_VAR_NO = ~d;~%~2%" controlNo))
(defun write-control-io-to-fmm (stream name baseID ID engine position &optional (rate nil))
  (format stream "~a~%{~%}" name)
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "ID = ~d;~%" ID)
  (format stream "ENGINE = ~a;~%" engine)
  (format stream "POSITION = ~a;~%" position)
  (format stream "LEFT_ORIENTATED = 0;~%")
  (if rate
    (format stream "RATE = ~a;~%" rate))
  (format stream "}~3%"))
(defun write-graph-definitions-to-fmm (stream baseID ID descr)
  (format stream "GRAPH~%{~%}")
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "AUTO_SCALE = true;~%")
  (format stream "BEAM_DIAGRAM = false;~%")
  (format stream "DESCR = \"~a\";~%" descr)
  (format stream "GRID_TTYPE = 2;~%")
  (format stream "ID = ~d;~%" ID))

```

```

(format stream "SHOW_LEGEND = false;~%")
(format stream "TIME_RANGE = 0 1;~%")
(format stream "USE_TIME_RANGE = false;~%")
(format stream "X_AXIS_RANGE = 0 1;~%")
(format stream "Y_AXIS_RANGE = 0 0.001;~%}~2%"))
(defun write-curve-sets-to-fmm (stream baseID ID owner x-result x-oper y-result y-object y-oper
  &key legend x-object)
  (format stream "CURVE_SET~%{~%")
  (format stream "BASE_ID = ~d;~%" baseID)
  (format stream "EXPORT_AUTOMATICALLY = true;~%")
  (format stream "FATIGUE_DOMAIN_START = 0;~%")
  (format stream "FATIGUE_DOMAIN_STOP = 1;~%")
  (format stream "FATIGUE_GATE_VALUE = 1;~%")
  (format stream "FATIGUE_LIFE_UNIT = REPEATS;~%")
  (format stream "FATIGUE_SN_CURVE = 0;~%")
  (format stream "FATIGUE_SN_STD = 0;~%")
  (format stream "FATIGUE_USING_ENTIRE_DOMAIN = true;~%")
  (format stream "ID = ~d;~%" ID)
  (format stream "INPUT_MODE = TEMPORAL_RESULT;~%")
  (if legend
    (format stream "LEGEND = ~a;~%" legend))
  (format stream "OWNER_GRAPH = ~d;~%" owner)
  (format stream "X_AXIS_RESULT = ~a;~%" x-result)
  (format stream "X_AXIS_RESULT_OPER = ~a;~%" x-oper)
  (if x-object
    (format stream "X_AXIS_RESULT_OBJECT = ~a;~%" x-object))
  (format stream "Y_AXIS_RESULT = ~a;~%" y-result)
  (format stream "Y_AXIS_RESULT_OBJECT = ~a;~%" y-object)
  (format stream "Y_AXIS_RESULT_OPER = ~a;~%}~2%" y-oper))
(defun get-rot-axis (constraint)
  (cross-product '(0 0 1) (get-joint-direction-vector constraint)))
(defun get-rot-angle (constraint)
  (angle-between-2-vectors '(0 0 1) (get-joint-direction-vector constraint)))
(defun get-unit-vector (constraint)
  (loop for element in (get-rot-axis constraint)
    collect (/ element (vector-length (get-joint-direction-vector constraint))))))

```

# Appendix G

## Risk Assessment

|   |                                     |              |           |            |   |
|---|-------------------------------------|--------------|-----------|------------|---|
| NTNU  | Kartlegging av risikofylt aktivitet | Uarbeldet av | Nummer    | Dato       |  |
|  |                                     | HMS-avd.     | HMSRV2601 | 22.03.2011 |   |
| HMS   |                                     | Godkjent av  |           | Erstatler  |   |
|   |                                     | Rektor       |           | 01.12.2006 |   |

Enhet: IPM

Dato: 3/2/2016

Linjeleder: Torgeir Welo

Deltakere ved kartleggingen (m/ funksjon): Anders Kristiansen (AK) (student), Eivind Kristoffersen (EK) (student), Bjørn Haugen (veileder)

Kort beskrivelse av hovedaktivitet/hovedprosess: Masteroppgave vår 2016 for Anders Kristiansen og Eivind Kristoffersen.  
AUTOMATISERING AV OPPGAVER VED MEKANISMEDESIGN

Er oppgaven rent teoretisk? (JA/NEI): JA

Signaturer: Ansvarlig veileder: *Bjørn Haugen*

Student: *Anders Kristiansen, Eivind Kristoffersen*

| ID nr. | Aktivitet/prosess            | Ansvarlig | Eksisterende dokumentasjon | Eksisterende sikringstiltak | Lov, forskrift o.l. | Kommentar |
|--------|------------------------------|-----------|----------------------------|-----------------------------|---------------------|-----------|
| 1      | Utvikling av AML-applikasjon | AK & EK   |                            |                             |                     |           |
|        |                              |           |                            |                             |                     |           |
|        |                              |           |                            |                             |                     |           |
|        |                              |           |                            |                             |                     |           |
|        |                              |           |                            |                             |                     |           |
|        |                              |           |                            |                             |                     |           |
|        |                              |           |                            |                             |                     |           |