UPPSALA
UNIVERSITET

UPTEC F15 020
Examensarbete 30 hp
Maj 2015

Effective Finite Element Analysis Workflow for Structural Mechanics

André Hedlund

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Effective Finite Element Analysis Workflow for Structural Mechanics

André Hedlund

The Finite Element Method (FEM) is a technique for finding the approximate solution of differential equations. It is commonly used in structural analysis to evaluate the deformation and internal stresses of a structure that is subject to outer loads. This thesis investigates the Finite Element Analysis (FEA) workflow that is used at Andritz Hydro AB, with the objective to find solutions that make the workflow more time effective. The current workflow utilises Siemens NX and Salomé for pre- and post-processing, and Code Aster as the FEM solver. Two different approaches that improve the workflow are presented. The first suggest that the entire FEA workflow is migrated to NX using the built-in FEM package of NX called Advanced Simulation. The second approach utilises the Salomé API (Application Programming Interface) to create a customised toolbox (a script containing several functions) that automate several repetitive and cumbersome steps of the workflow, therefore effectively reducing the time that is required by the analyst to perform FEA. Due to the positive results and ease-of-use, the Salomé toolbox is preferred over the license cost and steep learning curve that is related to NX and Advanced Simulation.

Handledare: Mikael Helin
Ämnesgranskare: Per Isaksson
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F15 020

Populärvetenskaplig sammanfattning

Andritz Hydro AB utvecklar och tillverkar turbiner och generatorer till vattenkraftverk. Det ställs höga krav på ett vattenkraftverk, de ska under lång tid (ca 50 år) producera el - och helst utan avbrott. För att säkerställa att de turbiner och generatorer som tillverkas klarar dessa mål, krävs en grundlig analys av konstruktionen och de maskindelar som ingår. Maskindelarna analyseras med avseende på hållfasthet (dvs. hur stora krafter som uppstår i materialet) och utmattning. Finita elementmetoden (FEM) är en numerisk metod som bland annat används för att beräkna hållfastheten hos mekaniska konstruktioner. Metoden delar upp konstruktionen eller maskindelen i ett antal (finita) delområden (element) och approximerar sedan en lösning över dessa delområden.

Det här examensarbetet undersöker det nuvarande arbetsflödet hos Andritz Hydro AB för att göra FEM-analyser av mekaniska konstruktioner, samt kommer med förslag på hur arbetsflödet kan förbättras. Det nuvarande arbetsflödet involverar tre olika programvaror (Siemens NX, Salomé och Code Aster) som används för olika delar av analysen. Två olika förslag på hur arbetsflödet kan förbättras läggs fram. Det första föreslår ett förenklat arbetsflöde genom att använde en och samma programvara för hela analysen, detta program är tyvärr licensbelagt och kommer att tillföra stora kostnader för företaget. Det andra förslaget utgår från det nuvarande arbetsflödet, men där vissa steg har identifierats som kan förenklas. Ett program har därför utvecklats som automatiskt utför tre steg i analysen som tidigare utförts manuellt.

Rekommendationen är att det nuvarande arbetsflödet förenklas genom att använda det program som utvecklats, och därmed minska tidsåtgången för att göra FEM-analyser.

Contents

1 Introduction

Hydro-power is the worlds largest renewable energy source, and hydroelectric power plants have been developed and used since the nineteenth century. During the last decade hydro-power amount to 45 percent of the total electricity production in Sweden [1]. Many of the hydro-power plants in Sweden were built several decades ago, and they are therefore in need of refurbishment and modernisation. A simplified description of a conventional hydro power plant is that it consist of a dam, a turbine and a generator, where the dam creates a water reservoir that contains potential energy that drives the turbine which in turn generates electricity.

The Andritz Group provides services mainly for the hydro-power, pulp and paper, and metals industries, with headquarters in Graz, Austria and approximately 24 500 employees worldwide. Andritz Hydro is a supplier of electromechanical equipment for hydro-power plants, and the Swedish subsidiary Andritz Hydro AB (henceforth referred to as Andritz) has 160 employees and focuses mainly on refurbishment and optimisation of turbines and generators.

At Andritz the entire product development process is performed, including product design, analysis and manufacturing. The design process, usually referred to as Computer Aided Design (CAD), and the analysis process, usually referred to as Computer Aided Engineering (CAE), are the parts of the product development that is the focus of this thesis.

Turbines and generators are advanced electromechanical equipment, and since hydroelectric power plants need to reliably operate under long terms, the modernisation process is subject to rigorous analysis to ensure long term operation. Therefore, it is important to analyse the structure's capability to support the applied loads; *structural mechanics* focuses on the computation of deformations and stresses, to evaluate the performance of the structure. The physical phenomena that are investigated (stress, strain, deformation, etc.) are modelled by solving differential equations together with a set of boundary conditions. Generally, there exist no analytical solutions to differential equations that relate to complex structures, therefore, the only way to estimate solutions of the equations are with numerical methods.

The finite element method (FEM) is a numerical approach that can be used to approximate the solution of boundary value problems. The method has been used rigorously in different engineering disciplines for several decades, and its main advantage over other numerical methods for solving differential equations is its capability to handle complex geometries. The structure is divided into smaller parts, called elements, which together create a discrete mesh that represent the geometry. [2]

There exist several different software programs for CAD and CAE, at Andritz *Siemens NX* is used for CAD and *Salomé* together with *Code Aster* are used for the finite element analysis (FEA). The analysis process is comprised of three steps: pre-processing, solution and post-processing. The goal of the pre-processing is to from a CAD model develop an FE model containing a mesh, material definitions and boundary conditions. In general, a significant amount of time is spent on the pre-processing and a lot of manual work can be required by the analyst, it is therefore important to make the pre-processing as effective as possible.

At Andritz the analysis workflow is in some aspects cumbersome and convoluted, and a more streamlined workflow is desired. Some of the difficulties occur solely because of that two different software programs are used for CAD and CAE, and other difficulties are generically inherited from the software programs that are used.

The purpose of this thesis is to evaluate the current workflow at Andritz, based on time-efficiency and convenience. Based on what the evaluation manifest, suggestions to improve the workflow will be presented. The suggestions are concerned with both larger strategical improvements, such as evaluating other software programs, and smaller improvements that simplify time-consuming repetitive processes. The main aspects to consider are:

- time-efficiency,
- simple and convenient processes,
- cost-efficiency and
- easy and fast to learn.

Since the product development process is complex, it is not certain that there exist feasible solutions that improves the workflow. It is also not guaranteed that the software programs provides functionality such that processes can be simplified.

2 | Finite Element Analysis

This section gives a basic introduction to the Finite Element Method (FEM) for structural mechanics, and describes the different stages of finite element analysis (FEA). It also presents the software programs that are used in the current analysis workflow at Andritz.

2.1 Finite Element Method for Structural Mechanics

The finite element method is a technique for finding the approximate solution of differential equations. The differential equations are mathematical models that arise from various physical phenomena, such as heat conduction, electromagnetism and fluid dynamics, but the focus of this thesis is on structural mechanics. The structure, where the differential equations hold, is divided into elements, which in turn consist of N nodes. The nodes are the points where the solution is calculated, and at each node there are usually three spatial degrees of freedom, therefore the total number of degrees of freedom are $3 \times N$. [2, p. 1–4]

In structural mechanics the deformation and stresses are calculated for structural objects that are subject to loads (body and surface forces, thermal loads, etc.). Stresses are the internal forces in a structure that are reactions to the load that is applied to the structure, whereas strain is a measure of the deformation. [2, p. 235ff.]. The *constitutive relation* describes how stress and strain relate to each other. For linear elasticity Hooke's law, $\sigma = E\varepsilon$, describes the relation, where σ is the stress, ε is the strain and E is Young's modulus. Note that there exists several other constitutive relations which are not considered in this thesis [2, p. 248].

2.1.1 Mathematical Description

The content of this section is based on the book Introduction to the Finite Element Method by Niels Ottossen and Hans Petersson [2], for a more detailed description of the material the reader is referred to that source.

This section describes the FE procedure: how to get from the mathematical model (ie. differential equations) to a system of equations that can be solved with a numerical approach.

Given the stress tensor

$$\mathbf{S} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix}, \tag{2.1}$$

the stress equilibrium is given by

$$\begin{cases} \dfrac{\partial \sigma_{xx}}{\partial x} + \dfrac{\partial \sigma_{xy}}{\partial y} + \dfrac{\partial \sigma_{xz}}{\partial z} + b_x = 0 \\ \dfrac{\partial \sigma_{yx}}{\partial x} + \dfrac{\partial \sigma_{yy}}{\partial y} + \dfrac{\partial \sigma_{yz}}{\partial z} + b_y = 0 \\ \dfrac{\partial \sigma_{zx}}{\partial x} + \dfrac{\partial \sigma_{zy}}{\partial y} + \dfrac{\partial \sigma_{zz}}{\partial z} + b_z = 0 \end{cases} \quad \Leftrightarrow \quad \widetilde{\nabla}^T \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \tag{2.2}$$

where $\widetilde{\nabla}^T$ is a differential operator and $\mathbf{b}$ is the body force acting on the structure. The strain kinematic relation is

$$\boldsymbol{\varepsilon} = \widetilde{\nabla}^T \mathbf{u}, \tag{2.3}$$

where $\mathbf{u}$ is the displacement vector.

The boundary conditions are given by the traction vector $\mathbf{t}$ or a prescribed displacement $\mathbf{u}$ on the surface of the structure:

$$\begin{cases} \mathbf{t} = \mathbf{S}\mathbf{n} = \mathbf{h} & \text{on } S_h \\ \mathbf{u} = \mathbf{g} & \text{on } S_g \end{cases} \tag{2.4}$$

where $\mathbf{n}$ is the normal vector of the surface. The *strong formulation* of elasticity is given by Equation (2.2) and (2.3) with boundary conditions given by Equation (2.4).

To obtain the weak formulation the standard procedure is to multiply the strong formulation with a weight function $\mathbf{v}$ and then integrate over the volume; the case with Equation (2.2) yields

$$\int_V \mathbf{v} \widetilde{\nabla}^T \boldsymbol{\sigma} \, \mathrm{d}V + \int_V \mathbf{v}\mathbf{b} \, \mathrm{d}V = \mathbf{0}. \tag{2.5}$$

Using Green's theorem and the boundary condition in Equation (2.4) the *weak formulation* of the stress equilibrium equation is obtained

$$\int_V \left(\widetilde{\nabla} \mathbf{v} \right)^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \mathbf{v}^T \mathbf{t} \, \mathrm{d}S + \int_V \mathbf{v}^T \mathbf{b} \, \mathrm{d}V. \tag{2.6}$$

The FE formulation is based on the weak formulation, where the continuous displacement vector in the weak formulation is approximated by

$$\mathbf{u} = \mathbf{N}\mathbf{a} \tag{2.7}$$

where $\mathbf{N}$ is a *shape function matrix* and $\mathbf{a}$ is the nodal displacement vector. Using the same approximation for the weight function yields

$$\mathbf{v} = \mathbf{N}\mathbf{c}, \tag{2.8}$$

where $\mathbf{c}$ is an arbitrary vector. If Equation (2.8) is used in the weak formulation (2.6), the following expression is obtained

$$\int_V \left(\widetilde{\nabla}\mathbf{N}\right)^T \boldsymbol{\sigma}\,\mathrm{d}V = \int_S \mathbf{N}^T\mathbf{t}\,\mathrm{d}S + \int_V \mathbf{N}^T\mathbf{b}\,\mathrm{d}V. \tag{2.9}$$

Using the constitutive relation between stress and strain (in this case Hooke's law)

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon}, \tag{2.10}$$

where $\mathbf{D}$ is the constitutive matrix and Equation 2.3, the final FE formulation is obtained

$$\mathbf{K}\mathbf{a} = \mathbf{f}. \tag{2.11}$$

$\mathbf{K}$ is the linear stiffness matrix given by

$$\mathbf{K} = \int_V \left(\widetilde{\nabla}\mathbf{N}\right)^T \mathbf{D}\left(\widetilde{\nabla}\mathbf{N}\right)\,\mathrm{d}V, \tag{2.12}$$

and $\mathbf{f}$ is the force vector that is composed of the body forces, boundary conditions and the initial strain.

The system of equations that (2.11) represent is the system that is solved and the solution $\mathbf{a}$ is the approximation of the displacement vector $\mathbf{u}$. The size (ie. the number of unknowns) of the system is $D \times N$, where N is the number of nodes in the mesh and D is the dimension of the problem.

2.1.2 Elements

A principal feature of FEM is that the unknown function is approximated by dividing the geometry into a finite number of elements, which enables the method to approximate the unknown function on complex geometries. How these elements are constructed depend on the geometry and the problem definition. This section presents simple two- and three-dimensional elements that are common in FEA.

The unknown function is approximated by a polynomial, and the simplest element is the linear one-dimensional element. If a one-dimensional geometry is divided into $N-1$ elements with N nodes, the unknown is approximated on element i by a *shape functions* given by

$$\varphi_i = \begin{cases} (x - x_{i-1})/h_i & \text{if } x_{i-1} \le x \le x_i \\ (x_{i+1} - x)/h_{i+1} & \text{if } x_i < x \le x_{i+1} \\ 0 & \text{else} \end{cases}, \tag{2.13}$$

where h_i is the length of the element. The approximation over the entire geometry is then given by

$$u = \sum_{i=0}^{n} \alpha_i \varphi_i(x), \tag{2.14}$$

where α_i is a constant parameter (see Fig. 2.1). Instead of linear interpolation between the nodes, a quadratic element can be defined by introducing a nodal point in the middle of the element. [3, p. 1ff.]

Two- and three-dimensional elements are constructed by the same principle as described above, although the mathematical description is more involved. A

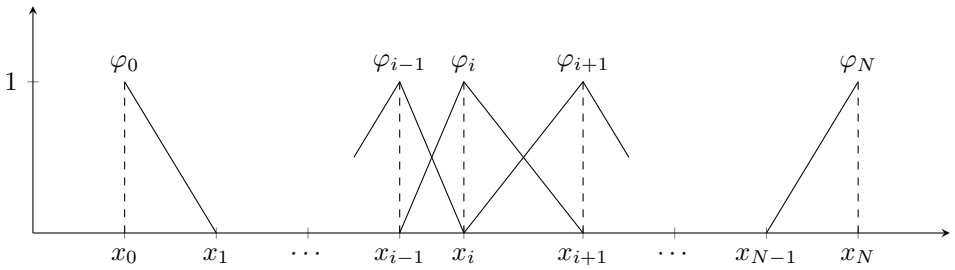


Figure 2.1: A one-dimensional mesh and the corresponding linear shape functions.

three-node triangular element is shown in Figure 2.2a, Figure 2.2b shows a four-node rectangular element, Figure 2.2c shows a four-node tetrahedral element and Figure 2.2d Higher order elements can be constructed by adding nodes on the mid-points of the elements, just as with the one-dimensional elements. [2, p. 118ff.]

2.1.3 Contacts

The differential equations presented in Section 2.1.1 are linear, and are thus easy to handle with the FEM. However, there exists many structural problems where the governing differential equations are non-linear. Non-linear differential equations in structural mechanics can relate to material, geometric, force and kinematic non-linearities; and the focus of this section is on kinematic non-linearities from contact problems. Since this thesis does not investigate the solution process, this section does not delve into to the topic, but merely mentions it and how the FE model should be prepared if contacts are present in the analysis.

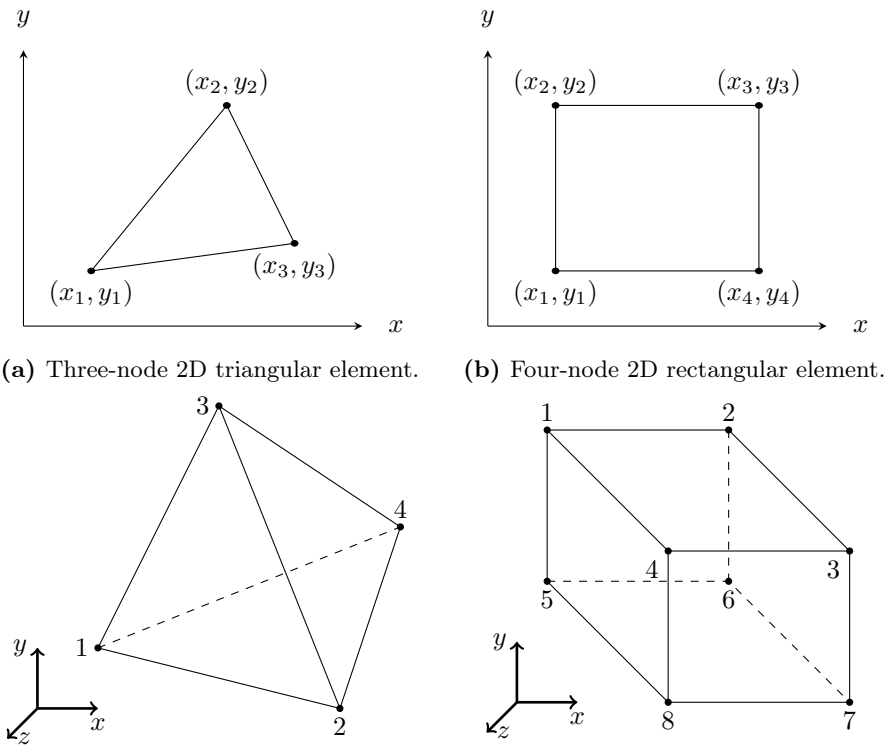
Contact problems occur when two or more structures come in contact with each other during the simulation. This means that the boundary condition prescribing the displacement depends on the deformations. This is handled in by introducing constraint equations to the system of equations. Generally, contacts can involve friction, large displacements and inelastic behaviour. For static analyses it is important to set boundary conditions such that no rigid-body motion is possible. [4, p. 549ff.]

2.2 FEA Workflow

The FEA consist of three phases: pre-processing, solution and post-processing; this section describes these phases.

2.2.1 Pre-Processing

When a structure has been designed, the CAD model needs to be analysed to determine if the desired performance qualifications are reached. Pre-processing is the first step of an FEA analysis where the CAD model is translated to an FE model. The FE model is a model that is meshed and have the appropriate properties assigned to it (i.e. material properties and boundary conditions).



(a) Three-node 2D triangular element.

(b) Four-node 2D rectangular element.

(c) Four-node 3D tetrahedral element.

(d) Eight-node 3D hexahedral element.

Figure 2.2: Two- and three-dimensional elements.

Ideally, the CAD model built by the design team is usable for finite element analysis without having to alter the model. However, this is often not the case, since it often exists features that are problematic from an analysis viewpoint. Therefore, it is necessary to make simplifications to the model in order to create a mesh of the model. Design features that complicate automatic meshing are short edges, sliver faces, small holes, fillets, chamfers, etc., and it is necessary to clean the model of these features to enable the automesher to mesh the model. This cleaning process can be both tedious and difficult since it may exist a lot of features, and even if a disadvantageous feature is found it may be difficult to remove it; the cleaning process can therefore be very time consuming. It is also important to mention that the simplifications that are performed should not influence the structural capabilities of the model, this could be difficult to assess, but as long as the simplifications are local and not in areas of interest the structural capabilities should not be affected. [5, p. 181–191]

When the model is clean, the next step is to mesh the model. Whether the model is meshed automatically or manually and with shell elements or tetrahedrons, the process is an essential step of the analysis. The focus of this thesis is on automeshing with tetrahedrons (for 3D models) and triangles (for 2D models). Automeshing is a simple and efficient technique to create a discrete FE model of the CAD model. Depending on which automesher that is used different settings are available, but the user should generally consider the maximum

and minimum elements size, element growth rate and local mesh refinements. The goal is to create a mesh that captures the structural features of the model with as few elements as possible. Even though the automesher could create the mesh in a fast and convenient way, it is no guarantee that the mesh is sufficiently refined. Therefore, considerable thought should be spent on where local mesh refinements are necessary, how small elements are needed and how large elements are excepted. It is also important to mention that it is an absolute necessity that the model is clean, otherwise the automesher will most likely fail to create a mesh. [5, p. 251-255]

Part of the pre-process is also to define the boundary conditions that influence the model. There exist two groups of boundary conditions: *constraints* that prohibits the model from moving in the specified spatial degrees of freedom, and *loads* such as forces, moments and temperature. How the boundary conditions are defined and which types that are used are often not straightforward, and an important part of the analysis. [5, p. 263]

Before the FE model can be solved the material properties of the model needs to be specified.

It is probably obvious that this part of the analysis can be very time consuming, and to obtain a solution it is of paramount importance that the FE model is created correctly, with sufficient details of the original model and that boundary conditions are properly defined.

2.2.2 Solution

When the FE model is created, the next step is to solve the model to obtain the results. This step is mainly executed by the computer, the only effort from the user is setting the correct solver parameters. Which parameters that can be specified is highly dependent on which solver is used. The execution time depends on how large the FE model is (number degrees of freedom) and what solution type is used (non-linear solutions are more demanding).

When the solver is finished it is important to evaluate the results and the solver output to establish if the solution is reasonable and if the results are accurate. To determine if the mesh is sufficiently refined the convergence should be checked, and to check that the boundary conditions are properly defined the resultant forces on the model could be compared with the specified loads. [5, p. 303-324]

2.2.3 Post-Processing

The first step of the post-processing is to visualise the results (displacement, stress, etc.) to determine if the solution is reasonable. If the solution is reasonable the FE model can be considered to be adequate. The next step is to visualise the specific results that are the goal of the analysis. Even if this step is not very technical, it is important to analyse the results to determine if the results can be trusted.

2.3 FEA Software Packages

During the development of the FEM, lots of different software programs have been developed. This section describes the software packages that are relevant for this thesis, based on the software packages that are used at Andritz.

2.3.1 Siemens NX

NX^{TM} is a software developed by Siemens PLM Software that supports every aspect of product development (i.e. design, analysis and manufacturing) [6]. NX is a licensed software where the features that are provided is based on which license the user has purchased. This section describes a subset of the functionality provided by NX that is relevant for this thesis, and the focus is on the tools to clean the model (called Synchronous Modelling tools) and the FEA application (called Advanced Simulation). [7, p. 36ff.]

The synchronous modelling tools that are provided by NX are very efficient and powerful to use during the cleaning process. Synchronous modelling extends the regular parametric modelling approach that is used in CAD with more intuitive and direct modelling tools. The tools that are most commonly used are: *Move Face*, *Delete Face*, *Make Coplanar*, *Pull Face*, etc., for a description of these tools see [7]. The main advantage of these tools is that the model can be changed without having access to the original history-tree of the commands that created the model.

NX Advanced Simulation provides the entire CAE workflow with pre- and post-processing and solver environment. It is also tightly integrated with the CAD application of NX, enabling a fast and efficient analysis process. The meshing capabilities that Advanced Simulation provides are the usual 3D and 2D automeshers as well as more exotic elements for specific applications. The simulation environment can be integrated with several of the most common solvers such as NX Nastran, Abaqus and Ansys.

2.3.2 Salomé Platform

Salomé is a free software platform (distributed under GNU LGPL [8]) for numerical simulation. Salomé strives to provide a framework where the entire workflow of a numerical simulation (described in Sec. 2.2.1-2.2.3) can take place. The platform consists of different modules that can be used for pre- and post-processing, which are presented under the same GUI (graphical user interface). [9]

Geometry Module

The *Geometry* module provides basic functionality to import CAD models and prepare them for numerical simulation. Common tasks that are carried out in this module are to create *groups* and *partition* objects.

A group in Geometry is a collection of geometrical objects (solids, faces, edges or vertices), which is given a common name and can be referenced by that name from other modules. Groups are used for several different purposes including: local mesh refinement, boundary conditions, contact surfaces, material definition, etc.

To be able to control the automeshing it is useful to define patches on the model where local mesh refinements are necessary to get the solution to converge. A patch is a group of faces or edges that should be better resolved. Any geometrical object of the model that is the subject of a boundary condition should also be defined as a group. If two objects are in contact, and a contact simulation is required, the contact surfaces should be defined as groups.

The partition function connects solid objects and creates a single solid, but the contact face will still remain between the objects. This function is mainly used when a conformal mesh has to be created for a contact simulation. [10]

Mesh Module

When a model has been imported to Geometry and the groups are defined, the *Mesh* module can mesh the model. Mesh provides the most common elements and several different algorithms for mesh generation are available, resulting in a very extensive library of functions.

To mesh an object with the Mesh module first an object from Geometry is selected, then an algorithm is selected and finally an hypothesis is created. The hypothesis represent the parameters that the algorithm use to create the mesh, depending on the algorithm different parameter values are specified in the hypothesis. Even though there exist several algorithms in Mesh, this section only describes the *NETGEN* algorithm and the *Projection* algorithm.

The NETGEN algorithm [11] is an automesher that can be used for creating 3D tetrahedral meshes and 2D triangular and quadrilateral meshes. The hypothesis for the NETGEN algorithm specifies

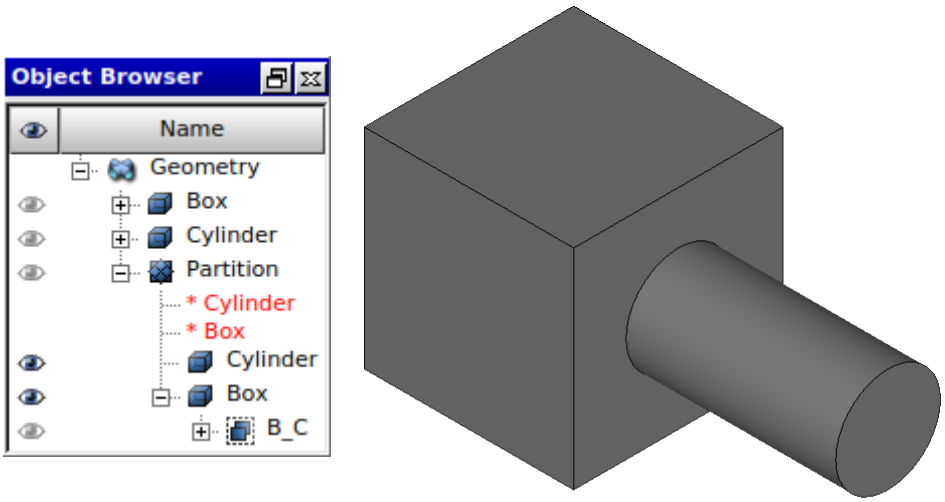
- the maximum and minimum element size,
- if second order elements should be used,
- growth rate,
- the possibility to specify the number of elements based on surface curvature, and
- local refinements on specified groups.

The projection algorithm creates a mesh of an object by projecting the mesh of another object. This algorithm is especially used when conformal meshes are desirable.

The mesh module supports, just as the Geometry module, the creation of groups. This feature is specifically useful when a specific part of the mesh needs to be referenced or exported.

Contact Mesh

If the model contains contacts, it can be an advantage to have matching meshes at the contact surface. A matching mesh means that the mesh of the two objects in contact have nodes at the same place at the contact surface. In Salomé there are several ways to obtain a mesh with matching element nodes, here follows a description using the Projection algorithm.



(a) Object browser

(b) A box and a cylinder in contact.

Figure 2.3: A simple example demonstrating how a contact mesh can be created in Salomé.

Consider the box and cylinder in Figure 2.3. To create a matching mesh at the contact surface between the box and the element the following steps are required:

1. Create a partition of the box and cylinder.
2. Under the partition create one group containing the box and one containing the cylinder.
3. Create a sub-group of the contact surface under either box or cylinder (see group `B_C` in Fig. 2.3a).
4. Mesh the box and cylinder group under the partition separately.
5. Create a sub-mesh under `Box` using the Projection algorithm on the contact surface.

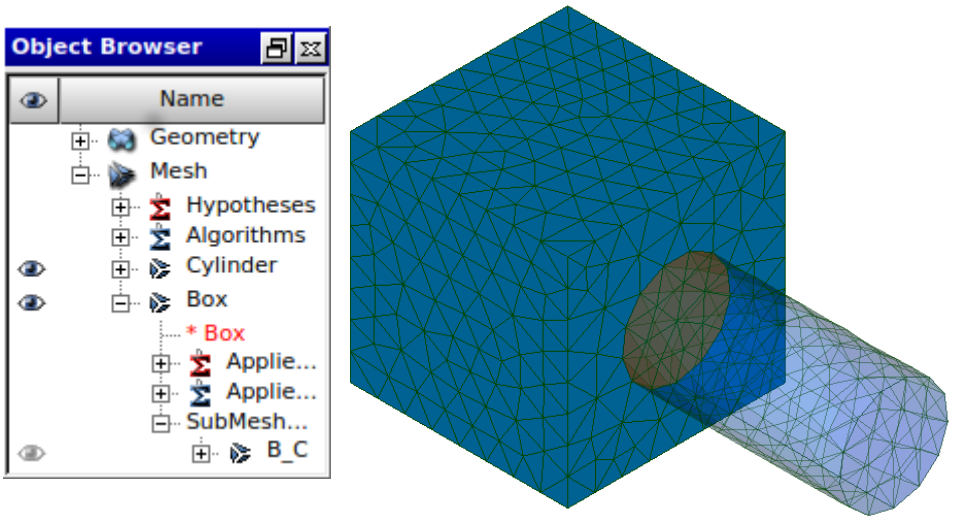
The resulting mesh and the Object Browser is depicted in Figure 2.4.

ParaVis Module

The ParaVis module is based on ParaView an open source platform for data analysis and visualisation. Suffice to say that all the functionality needed to visualise the results in a structural analysis is provided.

Application Programming Interface (API)

A very advantageous feature of Salomé and the modules that have been described is that the functionality provided by the GUI also is available through an API, enabling the user to write scripts that execute functions automatically.



(a) Object browser

(b) The contact mesh.

Figure 2.4: An example of a contact mesh created in Salomé.

An API is a framework of functions for building software applications. Using the API a developer can efficiently create software that perform specific tasks by using the functionality provided by the API. The documentation of the API describes which functions that are available and what each functions does.

Salomé's API is based on the Python programming language and the Salomé GUI includes a Python console, by which the user can execute scripts that utilise the API. There exist a specific API for each module and these can work together within the same script, therefore it is possible to write a script that first import a model to Geometry where groups are created, then the script can select an algorithm and create a hypothesis, and finally the script can update the GUI.

Figure 2.5 gives a simplified view of how the different modules of the Salomé platform interact with each other and with other software programs.

2.3.3 Code Aster

Code Aster [12] is an open source FEM software for structural analysis which is developed by EDF (Électricité de France) and distributed under GPL (General Public License) [13]. The software is driven by an input file with commands that that describe the simulation (a COMM file), and a text file describing the mesh (usually a MAIL file). The information the MAIL file needs to contain is the mesh data, that is the nodes and the elements of the mesh, but the file also needs to describe the patches on which the boundary conditions are applied, these patches can be created in the Mesh module as described in Section 2.3.2. The output is text files describing general information about the simulation (errors, warnings, convergence, etc.) and a file containing the results which can be imported to Salomé and visualised with the ParaVis module.

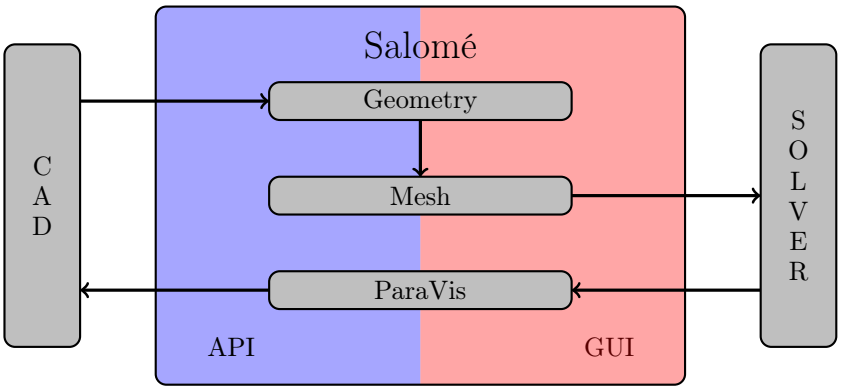


Figure 2.5: The Salomé platform provides three modules that are used in the FEA workflow (Geometry, Mesh and ParaVis). These modules can be accessed either by the GUI or the API. The output from the CAD software is read by the Geometry module and the solver reads output from the Mesh module and creates output to the ParaVis module.

3 Analysis

The previous section described structural analysis and FEA in general and presented different software programs that can be used in the analysis. This section presents how FEA is performed at Andritz based on the investigation made by the author of this thesis. The investigation is the basis of the improvement suggestions that are given in this section.

3.1 Analysis Workflow at Andritz

The structural analysis starts with a CAD model created by the design team. The design engineers at Andritz use Siemens NX to create CAD models and internally in NX these models are called *parts*. The parts are given a reference number to a database where they are stored, and this reference number is passed on to the calculation team when an analysis of the part is necessary. As described in Section 2.2.1, the CAD model needs to be cleaned of features that are unnecessary to the analysis. Since the cleaning process can change the model significantly it is not appreciated by the design engineers that the model they created is altered, therefore, a copy of this model is created which can be prepared for analysis. A copy is created by creating a new part in the database that is assigned to the calculation engineer and then using a synchronous modelling tool called *WAVE Geometry Linker* to link the original CAD model.

The cleaning process is done in NX with the synchronous modelling tools that is described in Section 2.3.1. Depending on the level of detail of the model that is provided, the amount of work that is needed to clean the CAD model can range from very little to a significant part of the entire analysis process.

When the part is ready to be meshed, it is exported from NX. There exists several formats to export a CAD model such that the details of the model are preserved, and at Andritz the STEP (standard for the exchange of product model data) format is used. A STEP file describes a CAD model according to an ISO standard which is used for file exchange [14].

The rest of the pre-processing is carried out in the Salomé platform described in Section 2.3.2. The STEP file is imported and the Geometry module is activated. As described in Section 2.3.2, groups are created in the Geometry module to define where local mesh refinement is needed, boundary conditions are applied and contact surfaces exists.

After the groups are defined the analyst change to the Mesh module where the model is meshed. The meshing process in Salomé is described in Section 2.3.2. 3D models are usually meshed with tetrahedrons (see Sec. 2.1.2)

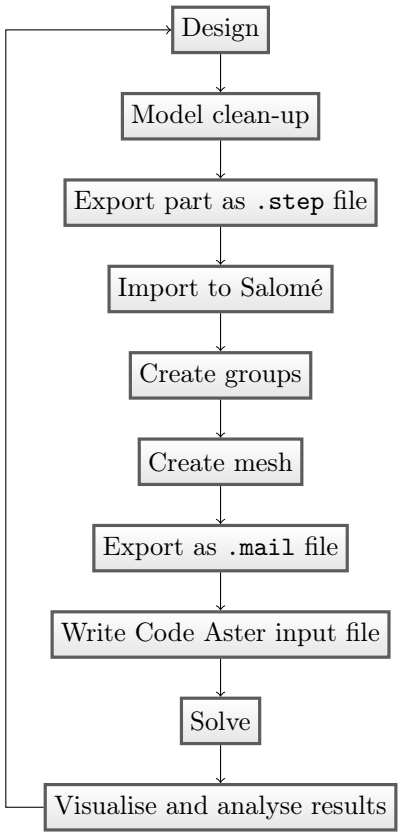


Figure 3.1: Current analysis workflow at Andritz.

and 2D models are usually meshed with triangles, in both cases the NETGEN algorithm is used.

To solve the model Code Aster is used and, as is described in Section 2.3.3, Code Aster requires the mesh to be described in a text file where patches, on which boundary conditions are applied, are defined. Therefore, mesh groups are defined in the Mesh module before the mesh is exported.

The final steps of the analysis (solution and post-processing) has not been investigated to the same level of detail, and are therefore summarised just to give a complete description of the analysis workflow.

Given a file describing the mesh and a COMM file that describes the simulation, Code Aster can run the simulation. The COMM file describes the type of simulation, boundary conditions and other parameters depending on the type of simulation. The output of the simulation is the results that can be visualised and analysed with the ParaVis module.

The entire workflow described above is presented in Figure 3.1.

If the results are feasible the development of the CAD model can continue, and preparations for the manufacturing process can begin. However, if the results show that the model does not fulfil the requirements, the model needs to be redesigned and the analysis repeated on the updated model. This iteration

procedure is standard in product development. Since the analysis of the updated model is likely to be very similar to the original model some of the steps described in Figure 3.1 are reusable, however, some steps have to be performed again. During the analysis of a redesigned model the following steps cannot be reused:

- Model clean-up
- Create geometric groups
- Create mesh
- Create mesh groups

Obviously the visualisation and analysis of the results need to be repeated, but that is unavoidable. The four steps mentioned above are all part of the pre-process of the analysis, and they could be a very time consuming part of the analysis, so it would be favourable to find a solution that could reuse the fact that these steps already have been performed on a very similar model. The reason why these steps have to be repeated is because the link between the CAD model and the analysis model is broken during the first step when the CAD model is copied. Therefore, all succeeding steps of the pre-process needs to be repeated.

3.2 Using NX Advanced Simulation

The previous section described the structural analysis workflow at Andritz and parts of the workflow that are inefficient where highlighted (especially if a new iteration is started). This section describes a possible solution to some of the problems that are discussed. NX Advanced Simulation (described in Sec. 2.3.1) supports the entire FEA process, and since the CAD model is created in NX it can seem like a good fit to analyse the model in the same software.

3.2.1 NX Licenses

Currently Andritz has one license for the Advanced Simulation application, this license is called *MasterFEM (A026)* and includes pre- and post-processing capabilities and the NX Nastran Basic solver. MasterFEM is a legacy license package that is no longer provided by Siemens, it has been replaced by a license called *NX Advanced Simulation (NX13500)* which has similar capabilities. If Advanced Simulation should be used at Andritz it is necessary to purchase additional licenses and, therefore, the cost must be considered.

Previous NX licenses where purchased from Conex Software, and the price information they have provided is presented in Table 3.1. The difference between the two licenses is that Advanced FEM does not include the NX Nastran Basic solver.

Note that these licenses include a variety of features (for a complete description see [6]), but the functionality they provide is sufficient for the analyses that are performed at Andritz.

Table 3.1: Price information for NX pre- and post-processing and solver features.

License	**Lic. No.**	**Price** (SEK/license)
NX Advanced Simulation	NX13500	137 900
NX Advanced FEM	NX12500	101 920

3.2.2 Advanced Simulation Workflow

To perform a FEA in Advanced Simulation the starting point is a `part` file describing the CAD model. When a new simulation is started three files are created.

- The *idealised* `part` file is associatively connected (optionally) to the original CAD model, and the purpose of this file is to create a simplified model of the original CAD model.
- The `fem` file represents the FE model, where material properties are defined and a mesh is created.
- The `sim` file where boundary conditions and solver settings are specified.

These three files together define the simulation and they are tightly connected such that if there is a change in the idealised `part` the `fem` and `sim` file is automatically updated and the mesh and boundary conditions are adapted to the change. Since the idealised `part` file is associatively connected to the CAD model this feature is very efficient when a new design-analysis iteration is started.

3.3 Salomé API Script

Salomé provides a Python API for every module, which allows the user to write Python scripts to automate processes that are often repeated. As is described in Section 3.1, there are steps in the pre-process of the FEA workflow that can be time consuming to perform manually, and it is especially inefficient if a new design-analysis iteration is started since the steps that already have been done must be repeated. This section presents a solution that efficiently automate some parts of the pre-processing steps and at the same time reduces the amount of work that have to be repeated if a new iteration is started.

The main issue with the workflow in Figure 3.1 is that the connection between the CAD model and the FE model is lost when the part is exported from NX and imported to Salomé. All the steps of the pre-processing that is done in Salomé needs to be repeated with the current workflow. Therefore, it would be better to move as many of the steps as possible from Salomé to NX, since these steps would not have to be repeated. Within the current NX license at Andritz (MasterFEM), the step that can be moved is the definition of the groups.

The main idea of the new workflow is to clean the model and define the groups in NX, and when the model is exported the group definitions are included in the STEP file. In Salomé the model is imported and the groups defined in

NX are automatically created. The model is meshed and the groups that should be included in the text file describing the mesh are automatically created in the Mesh module.

3.3.1 Defining Groups in NX

In NX any geometrical object (solid, face, edge, etc.) or collection of objects can be given a name. The name and which geometrical object it is associated with is included in the STEP file when the model is exported. The import utility in Salomé does not recognise these names as groups when the STEP file is imported, therefore these names needs to be translated to group objects. The Geometry API provides several functions to automate this process, and the solution is a function named `CreateGroups` which is presented in Appendix 6.1. `CreateGroups` creates the groups that are defined in NX of the object that is selected in the object browser.

3.3.2 Creating a Mesh

The calculation team at Andritz usually use the NETGEN algorithm with similar settings every time a mesh is created. Since the process of creating meshes in the Mesh module can be repetitive, a function has been developed that creates a mesh using predefined settings and automatically adds the groups that should be locally refined. The function is called `CreateMesh` (see App. 6.2), and works by selecting an object in the Object Browser and then executing the function. If the object that is selected has any group with a name starting with `FIN`, that group is automatically added to the local sizes.

3.3.3 Creating a Contact Mesh

As described in Section 2.3.2 it is complicated to create a mesh with matching nodes at contact surfaces. Therefore, two functions have been developed to automate the process of creating a contact mesh: `PartitionShapes` and `CreateSubMesh`.

`PartitionShapes` partitions several objects and identifies contact surfaces based on a naming convention for groups, the groups are placed in a hierarchical tree that is required by the Projection algorithm (see App. 6.3). `CreateSubMesh` creates a sub-mesh using the Projection algorithm to ensure that the contact is meshed with matching nodes.

The naming conventions under which these functions operate are:

- The objects that are in contact have names consisting of two upper case characters (eg. `CY` for `Cylinder` and `BO` for `Box`).
- The contact surface is named by the two objects that are in contact, separated by an underscore (eg. `CY_BO`).

If the objects are named by these conventions, the workflow for creating a contact mesh is:

1. Select the objects that are in contact and execute `PartitionShapes`.
2. Select the contact group and execute `CreateSubMesh`.

3.3.4 Workflow with Salomé Script

In this section a new, more efficient, workflow is proposed that utilise the functions that are described in Sections 3.3.1-3.3.3, see Figure 3.2. There are several advantages of this workflow and the main features are summarised below:

- The synchronous tool WAVE Geometry Linker creates a copy of the CAD model that is associative, therefore, any change to the CAD model can update the FE model in NX. If groups are defined in NX, it is only necessary to define them at the initial stage of the analysis since they continue to exist even if the CAD model is changed (the groups will adapt to the changes). This is very efficient if there are several iterations of the product development.
- The groups are imported to Salomé with the function `CreateGroups`, this is a simple and time-efficient process that requires very little from the user. Creating a mesh, whether it contains contacts or not, is automated by the functions `CreateMesh`, `PartitionShape` and `CreateSubMesh`, the only manual work is to adjust the mesh settings.
- The mesh definitions, and the groups that are required by Code Aster, are automatically created by `CreateMesh` and `CreateSubMesh`.

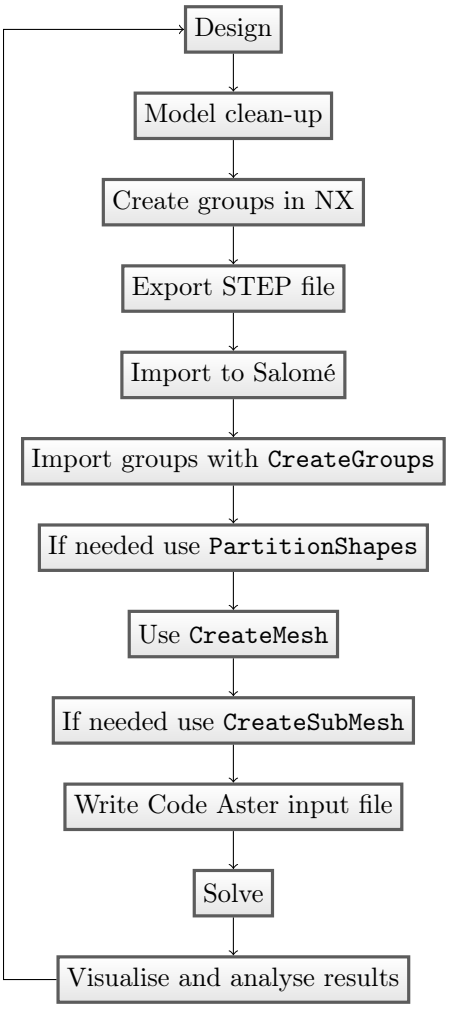


Figure 3.2: Proposed analysis workflow at Andritz using the functions developed with the Salomé API.

4 Conclusions

This chapter discuss the different solutions presented in Chapter 3, and recommends the – according to the author – preferred workflow for FEA at Andritz. The two different approaches (NX Advanced Simulation and Salomé API Script) are first discussed separately and in the last section the final recommendation is presented.

4.1 NX Advanced Simulation

With NX Advanced Simulation the entire FEA workflow can be performed within the same software, therefore eliminating the time spent on transferring files between different software programs. The environment that Advanced Simulation provides with the idealised part and the associativity to the original CAD model is very efficient for the design-analysis iterations. Advanced Simulation can therefore be expected to increase efficiency for the calculation engineers at Andritz.

There is currently only one Advanced Simulation license at Andritz (Master-FEM). There are at least three calculation engineers that need a license, therefore, additional licenses need to be purchased if Advanced Simulation should be used as CAE software. Since the solver is used in the background, whereas the pre- and post-processing features are used in the foreground by the user, it is possible to manage the workflow with a separate pre- and post-processing license for each calculation engineer and a shared solver license between the calculation engineers. Therefore it would suffice if the calculation team purchased three NX Advanced FEM licenses and kept the current NX Nastran solver, but since it might be too limited to only have one solver another possibility is to purchase one NX Advanced FEM and two NX Advanced Simulation licenses. These two different configurations represent a price range of 305 760 – 377 720 SEK; this cost is an important aspect to consider with this particular workflow.

Since the calculation engineers at Andritz have been working with the Salomé Platform and Code Aster in their FEA, it is also expected that the introduction of a new solver and GUI will demand a learning curve that can be costly depending on the usability of Advanced Simulation. Even though the GUI is relatively easy to use and a comprehensible documentation exists, the time that is lost on learning the new software can be significant and it might even be necessary with training courses for the employees (that comes at an additional cost).

4.2 Salomé API Script

The current workflow (see Sec. 3.1) uses NX for cleaning, Salomé for meshing and defining where boundary conditions, etc. are applied, Code Aster for solving, and Salomé to visualise the results. To make this workflow more efficient a script, that utilises the API that Salomé provides, has been developed (see Sec. 3.3). This script enables the user to automate some of the procedures that were previously performed manually.

The main change in the workflow with the script is that the groups are defined in NX instead of in Salomé. The most important aspects of this workflow is that the script is stable (ie. it works seamlessly regardless of the type of analysis). If the script would fail, and the groups that were defined in NX has to be redefined in Salomé, the script would only cause annoyance. During the testing of the script problems concerning the definition of groups have been encountered (though they have been sparse), for example have the groups been translated to a format in the STEP file that the script cannot identify, therefore the groups must be redefined again.

4.3 Summary

The fact that the workflow that is proposed with NX Advanced simulation is related with an increase in cost and a learning period for the users is a very big disadvantage. Advanced Simulation might solve the problems that exist in the current workflow, but since it is related to a significant cost increase it is not a feasible solution.

The workflow that is related with the script is very similar to the current workflow, therefore no significant learning time is expected. The advantage is that some repetitive processes are automatically performed with the script and the amount of work that has to be repeated in a new design-analysis iteration might be significantly reduced.

On these grounds the recommendation is to continue with the workflow described in Section 3.3.4 that uses the script that has been developed.

5 | Acknowledgements

I would like to thank my supervisor at Andritz Hydro AB Mikael Helin for providing me with the subject of this thesis, and for his guidance and experience to help me develop the final product. I would also like to thank Tobias Marten at Andritz Hydro AB for taking his time to answer my questions and guiding me in the right direction. Finally, I would like to thank Per Isaksson at Uppsala University for taking his time to read my thesis and quickly providing me with feedback.

Bibliography

[1] *SCB. Statistics Sweden*. 2014. URL: `http://scb.se/` (visited on 12/04/2014).

[2] N. Ottossen and H. Petersson. *Introduction to the finite element method*. Prentice Hall Europe, 1992.

[3] M. G. Larsson and F. Bengzon. *The Finite Element Method: Theory, Implementation and Applications*. Springer Verlag, 2013.

[4] M. A. Bhatti. *Advanced Topics in Finite Element Analysis of Structures*. John Wiley & Sons, Inc., 2006.

[5] V. Adams and A. Askenazi. *Building Better Products with Finite Element Analysis*. OnWord Press, 1999.

[6] *Siemens - NX*. 2015. URL: `http://siemens.com/nx` (visited on 03/24/2015).

[7] P. Goncharov, I. Artamonov, and T. Khalitov. *Engineering Analysis with NX Advanced Simulation*. Lulu Publishing Services, 2014.

[8] *GNU Lesser General Public License*. 2015. URL: `http://gnu.org/copyleft/lesser.html` (visited on 03/23/2015).

[9] A. Ribes and C. Caremoli. "Salomé platform component model for numerical simulation". In: *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*. Vol. 2. July 2007, pp. 553–564.

[10] *Salome Platform Documentation. Geometry*. 7.5.1. Open Cascade. 2015. URL: `http://docs.salome-platform.org/7/gui/GEOM/index.html`.

[11] *NETGEN - automatic mesh generator*. 2015. URL: `www.hpfem.jku.at/netgen/` (visited on 03/23/2015).

[12] *Code Aster*. 2015. URL: `www.code-aster.org/` (visited on 03/25/2015).

[13] *GNU General Public License*. 2015. URL: `http://gnu.org/copyleft/gpl.html` (visited on 03/25/2015).

[14] ISO. *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*. ISO 10303-21:2002. Internal Organization for Standardization, 2002.

6 | Appendix

6.1 CreateGroups

```
def CreateGroups(shape=[], edge=True, vertex=False):
    """
    Create groups that were defined in NX.

    Groups defined in NX are included in the STEP file.
    This function identifies these groups and adds them
    to the Object Browser.

    Attributes:
        shape -- name of the shape in a list, e.g. ["
    nameOfShape"]
        vertex -- True if vertex groups should be created
    """
    if not isinstance(shape, list):
        raise InputError("Name of shape should be passes as
    a list, e.g. ['nameOfShape']")
    shape = GetShape(shape)
    if not shape:
        raise BrowserError("Select an object in Object
    Browser")

    # Create groups of different types
    try:
        GetNXGroups(shape[0], "SOLID")
        GetNXGroups(shape[0], "FACE")
        if edge:
            GetNXGroups(shape[0], "EDGE")
        if vertex:
            GetNXGroups(shape[0], "VERTEX")
    except IndexError:
        print "Select or specify a shape."
        return
cg = CreateGroups
```

6.1.1 GetNXGroups

```
def GetNXGroups(shape, groupType):
    """
    Return groups that are defined in NX.
```

```
4
5       Attributes:
6           shape -- object to get groups from (not a list of
    objects)
7           groupType -- SOLID, FACE, EDGE or VERTEX
8       """
9       # Extract all groups from shape
10      try:
11          objects = geompy.ExtractShapes(shape, geompy.
    ShapeType[groupType], True)
12      except:
13          print "Shape could not be extracted.", shape
14          raise
15
16      for obj in objects[:]:
17          # Add objects to study to get NX names
18          geompy.addToStudyInFather(shape, obj, "")
19          # Delete objects with lower case letters
20          if obj.GetName() != obj.GetName().upper():
21              salome.geom.geomtools.GeomStudyTools().
    deleteShape(salome.ObjectToID(obj))
22              salome.sg.updateObjBrowser(1)
23              objects.remove(obj)
24
25      # Sort the remaining objects based on name
26      objects.sort(key=lambda obj: obj.GetName())
27
28      # For objects with equal names create a group
29      group = []
30      for obj in objects:
31          # Add object to 'group' if list is empty or the name
32          # of the object is the same as the previous object
33          if not group or obj.GetName()==group[-1].GetName():
34              group.append(obj)
35          else:
36              AddGroup(shape, group, groupType, group[-1].
    GetName())
37              group = [obj]
38      if group:
39          AddGroup(shape, group, groupType, group[-1].GetName
    ())
40
41      # Delete the objects from the object browser
42      for obj in objects:
43          salome.geom.geomtools.GeomStudyTools().deleteShape(
    salome.ObjectToID(obj))
44      salome.sg.updateObjBrowser(1)
```

6.2 CreateMesh

```
1   def CreateMesh(shape=[], minSize=10, maxSize=50):
2       """
3       Mesh the selected or specified object.
```

```
4
5       This function creates a mesh for a given shape. The
        shape can either be
6       selected in the Object Browser or given as a parameter.
7
8       Attributes:
9           shape -- name of the shape in a list, e.g. ["
        nameOfShape"]
10      """
11      if not isinstance(shape, list):
12          raise InputError("Name of shape should be passes as
        a list, e.g. ['shape1','shape2']")
13      shape = GetShape(shape)[0]
14      if not shape:
15          raise BrowserError("Select an object in Object
        Browser")
16
17      groups = geompy.GetGroups(shape)
18      # Create mesh
19      mesh = smesh.Mesh(shape)
20      smesh.SetName(mesh.GetMesh(), shape.GetName())
21      # Create algorithm and hypothesis
22      is3D = IsShape3D(shape)
23      [NETGEN, parameters] = CreateAlgorithm(mesh, groups,
        is3D, minSize, maxSize)
24
25      salome.sg.updateObjBrowser(1)
26  cm = CreateMesh
```

6.2.1 CreateAlgorithm

```
1   def CreateAlgorithm(mesh, groups, is3D, minSize, maxSize):
2       if is3D:
3           NETGEN = mesh.Tetrahedron(algo=smeshBuilder.
        NETGEN_1D2D3D)
4       else:
5           NETGEN = mesh.Triangle(algo=smeshBuilder.NETGEN_1D2D
        )
6
7       parameters = NETGEN.Parameters()
8       parameters.SetMaxSize(maxSize)
9       parameters.SetMinSize(minSize)
10      parameters.SetSecondOrder(1) # Second order elements
11      parameters.SetOptimize(1)    # Optimize activated
12      parameters.SetFineness(5)    # Set finess to Custom
13      parameters.SetFuseEdges(1)   # Fuse activated
14      parameters.SetQuadAllowed(0) # Quadrangles deacativated
15      parameters.SetGrowthRate(0.3)
16      parameters.SetUseSurfaceCurvature(0)
17      parameters.SetNbSegPerEdge(0.2)
18      parameters.SetNbSegPerRadius(0.2)
19      # Add local sizes
20      AddLocalSize(parameters, groups)
21      # Set name of algorithm and hypothesis
```

```
    smesh.SetName(parameters, "NETGEN")
    smesh.SetName(NETGEN.GetAlgorithm(), "NETGEN")
    return [NETGEN, parameters]
```

6.3 PartitionShapes

```
def PartitionShapes(shapes=[]):
    """
    Partition shapes, identify contact groups and create group tree.

    Attributes:
        shapes -- list of shapes that should be partitioned
    """
    if not isinstance(shapes, list):
        raise InputError("Name of shape should be passes as a list, e.g. ['shape1','shape2']")
    shapes = GetShape(shapes)
    if not shapes:
        raise BrowserError("Select an object in Object Browser")

    # Make a partition of the imported object
    if IsShape3D(shapes[0]):
        partition = geompy.MakePartition(shapes, [], [], [], geompy.ShapeType["SOLID"], 0, [], 0, "partition")
    else:
        partition = geompy.MakePartition(shapes, [], [], [], geompy.ShapeType["FACE"], 0, [], 0, "partition")
    # Add all groups of shapes to partition
    for shape in shapes:
        groups = geompy.GetGroups(shape)
        groups.append(shape)
        allGroups = geompy.RestoreGivenSubShapes(partition, groups, GEOM.FSM_GetInPlace, True, False)
        # Delete groups that where copied to partition
        groups.pop()
        for group in groups:
            salome.geom.geomtools.GeomStudyTools().deleteShape(salome.ObjectToID(group))
    groups = geompy.GetGroups(partition)

    CreateTreeStructure(groups)
    RemoveContactGroups(partition)

    salome.sg.updateObjBrowser(1)
ps = PartitionShapes
```

6.3.1 CreateTreeStructure

```
def CreateTreeStructure(groups):
    """
    Create a tree structure where contact groups are
```

```
    sub-groups of parents.
    """
    for group in groups[:]:
        MoveContactGroup(group, groups)
        oldGroups = MoveOtherGroups(group, groups)
        RemoveOtherGroups(oldGroups)
    return groups
```

6.3.2 MoveContactGroups

```
def MoveContactGroup(contact, groups):
    name = contact.GetName()
    # Check if objects's name has format: XX_YY###
    try:
        isContact = name[2]
    except IndexError:
        # Ignore names shorter than three characters
        isContact = ""
    if isContact == "_":
        # Get the type of contact
        groupType = geompy.GetType(contact)
        # 'contact' is a group, get the shapes of this group
        contactShapes = geompy.SubShapeAllSortedCentres(
    contact, groupType)
        for parent in groups[:]:
            try:
                parentName = parent.GetName()
            except:
                print "Couldn't get name of GEOM object."
                parentName = None
            if name[0:2] == parentName:
                # Add a sub-group to parent, containing the
                # shapes in contact
                newGroup = AddGroup(parent, contactShapes,
    groupType, name)
                groups.append(newGroup)
                # Remove old contact group
                groups.remove(contact)
            if name[3:5] == parentName:
                # New contact name
                contactName = name[3:5] + "_" + name[0:2] +
    name[5:8]
                # Add a sub-group to parent, containing the
                # shapes in contact
                newGroup = AddGroup(parent, contactShapes,
    groupType, contactName)
                groups.append(newGroup)
```

6.3.3 MoveOtherGroups

```
def MoveOtherGroups(group, groups):
    oldGroups = []
    for parent in groups:
        name = group.GetName()
```

```
        # Check if objects's name has format: XX_YY###
        try:
            isContact = name[2]
        except IndexError:
            # Ignore names shorter than three characters
            isContact = ""
        # Move groups XXGROUP to XX, and ignore XX_YY and XX
        if name[0:2] == parent.GetName() and isContact != "_
    " and name != parent.GetName():
            # Get the type of group
            groupType = geompy.GetType(group)
            # Get the shapes of 'group'
            subShapes = geompy.SubShapeAllSortedCentres(
    group, groupType)
            # Add a sub-group to parent, containing the
            # shapes in group
            newGroup = AddGroup(parent, subShapes, groupType
    , name)
            groups.append(newGroup)
            # Remove old group group
            oldGroups.append(group)
            groups.remove(group)
    return oldGroups
```

6.4 CreateSubMesh

```
def CreateSubMesh(sourceFace=[]):
    """
    Create a conformal contact mesh.

    Select a contact face from Geometry or specify the name
    as an input argument. The name of the contact face is
    XX_YY###, therefore the submesh is created under mesh XX
    with mesh YY as source mesh.
    """
    sourceFace = GetShape(sourceFace)[0]
    if not sourceFace:
        raise BrowserError("Select an object in Object
    Browser")

    contactName = sourceFace.GetName()
    # Get SMESH instances
    parentPath = "/Mesh/" + contactName[0:2]
    sourcePath = "/Mesh/" + contactName[3:5]
    parentMeshRef = salome.myStudy.FindObjectByPath(
    parentPath).GetObject()
    sourceMeshRef = salome.myStudy.FindObjectByPath(
    sourcePath).GetObject()
    # Get smeshBuilder object
    parentMesh = smesh.Mesh(parentMeshRef)
    sourceMesh = smesh.Mesh(sourceMeshRef)

    parentShape = GetObject(contactName[0:2], "GEOM")
```

```
    is3D = IsShape3D(parentShape)
    # Create Projection algorithm
    if is3D:
        projection = parentMesh.Projection1D2D(geom=sourceFace)
        sourceFaceHyp = projection.SourceFace(sourceFace, sourceMesh, None, None, None, None)
        smesh.SetName(sourceFaceHyp, 'Source Face')
    else:
        projection =parentMesh.Projection1D(geom=sourceFace)
        sourceEdgeHyp = projection.SourceEdge(sourceFace, sourceMesh, None, None)
        smesh.SetName(sourceEdgeHyp, 'Source Edge')
    contactMesh = projection.GetSubMesh()
    smesh.SetName(contactMesh, contactName)
    smesh.SetName(projection.GetAlgorithm(), 'Projection')

    # Add mesh groups for Code Aster
    if is3D:
        meshType = SMESH.FACE
    else:
        meshType = SMESH.EDGE
    # Add group to the mesh application
    try:
        meshGroup = parentMesh.GroupOnGeom(sourceFace, contactName, meshType)
        oppositeName = contactName[3:5] + "_" + contactName[0:2] + contactName[5:8]
        opposite = GetObject(oppositeName, "GEOM")
        meshGroup = sourceMesh.GroupOnGeom(opposite, opposite.GetName(), meshType)
    except:
        print "Add group to mesh failed!"

    salome.sg.updateObjBrowser(1)
csm = CreateSubMesh
```