Object-Oriented Mathematical Modelling — Applied to Rolling Bearings

Dag Fritzson

SKF Engineering & Research Centre B.V.
Postbus 2350, 3430 DT Nieuwegein, The Netherlands

Peter Fritzson, Lars Viklund, Johan Herber

Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden

August 31, 1992

Abstract

Machine element analysis has the goal of describing functionality and other important aspects of machine elements in theoretical form. A rolling bearing is a typical example of a machine element. The first issue covered in this paper is how ideas from object-oriented programming and modelling can be applied to machine element analysis. The models thus obtained are both easier to understand, better structured, and allow a higher degree of re-use than conventional models. Our conclusion is that an object-oriented model description is natural for the machine element analyst and is suitable for machine element analysis.

An equational model of rolling bearings is presented in this paper. The structure of the model is general, in three dimensions, and applies to many types of rolling bearings. Part of this generality is obtained by using parametric surfaces. In order to keep the presentation of the model understandable, we have simplified many details. Still, the model and one solution require approximately 200 + 200 equations. The model is also extensible, e.g. simple submodels of detailed properties can be made more complex without altering the overall structure.

The rolling bearing model has been implemented in our language, called ObjectMath (Object oriented Mathematical language for scientific computing). Using ObjectMath, it is possible to model classes of equation objects, to support inheritance of equations, and to solve systems of equations. This is the first programming environment and modelling language which integrates inheritance within a computer algebra language. Algebraic transformations can conveniently be supported since ObjectMath models are translated into the Mathematica computer algebra language. When necessary, equations can be transformed to C++ code for efficient numerical solution. The re-use of equations through inheritance reduced the size of the model by a factor of two, compared to a direct representation of the model in the Mathematica computer algebra language.

Keywords: machine element, mechanical analysis, modelling, computer algebra, object-oriented, scientific computing.

1 Introduction

In this paper we focus on how to improve the modelling of machine elements, by introducing methods and tools that provide better and clearer model structure, and enhanced re-usability. We investigate the use of object-oriented modelling techniques for machine element modelling. The investigated modelling technique supports programming in equations instead of low-level procedural programming, and permits better re-use of equations in mathematical models. Some questions are:

- How will an object-oriented description compare to and support a "natural" description of machine elements? Will it support the "thought process" of a mechanical analyst?
- Will it simplify the model definition and simulation phases?

To evaluate the technique, a prototype of a Mechanical Analyst's Workbench including a modelling language (ObjectMath Object oriented Mathematical language for scientific computing) has been designed and implemented, and used for a rather realistic rolling bearing application. ObjectMath is developed through cooperation between application experts and computer scientists.

The structure of the paper is as follows: First we review the current state of the art in scientific computing and the need for improvement. Then, the structure of

Figure 2: *A three-dimensional view of the rolling bearing, automatically generated from the model example equations.*

3 The structure of models in the machine element application domain

We need to describe the structure of machine elements, i.e. the parts, components and their substructure, together with material data and interaction between components. These together represent the total machine element. As an example of a machine element model, we will use a rolling bearing model throughout this paper.

Considering our machine element example, the rolling bearing has the following structure:

Parts:

- *Rings:*
  - *Inner ring.*
  - *Outer ring.*
- *Rolling element set(s):*
  - *Rolling element 1.*
  - ...
- *Guiding elements:*
  - *Cage(s).*

Figure 3: *Physical structure: the parts hierarchy for a simple version of the bearing model.*

  - *Guide ring.*
- *Lubricant.*

Material models/data: Here we are concerned with constitutive models and material data for materials and fluids used.

- *Material constitutive model(s):*
  - *Model 1.*
  - ...
- *Material data:*
  - *Material 1.*
  - ...

Relevant geometry description: For example, this can be a detailed parametric surface/volume model, or just a few parameters such as diameter and length.

Function description: The function is the most important aspect of a machine element model. This may include many aspects such as: a model for deformation under external loads and temperature, strain and stress inside the body, interaction with other machine element bodies including contacts and forces, interaction with a fluid, etc.

3.1 Analysis oriented modelling

So far we have discussed a descriptive and easily understood type of model exemplified by a rolling bearing. Such models describe what *parts* an object consists of and what *functions* they perform. In this section we concentrate on the analysis aspect, i.e. what equations we should solve and which quantities are relevant. The previous description must be modified to suit this purpose. This should be done without losing the "natural" decomposition and description of the machine element and its function.

We also introduce the term "*object*". An object can represent many different things and we can have different classes of objects. Some examples of what an object can be are:

- a real three dimensional body,
- an equation,
- a coordinate system,
- a set of material data,
- or a combination of those items.

Regarding the previous description of the rolling bearing example one can conclude that there are primarily two types of information that can be grouped into objects:

Self-contained objects: These are objects which are more or less self-contained, e.g. physical objects or abstract objects such as coordinate systems. They describe a certain object and not the relations between objects.

Connection objects: The main purpose of these objects is to describe relations and connections between other objects. A typical example is a contact between two physical bodies.

Another observation is that it is quite useful to use existing objects when defining new objects. New objects can be defined through composition and combination of existing objects, or as specialization of existing objects using inheritance. For example, a cylindrical roller element can inherit many properties from an existing general cylinder object, and add special information such as size and material properties.

3.2 Self-contained objects

In this class of objects there are several natural different object classes that describe various aspects of the studied problem. Some examples of basic objects:

- A geometrical description of a physical body.
- A coordinate system.
- A theory that describes the structural deformation of a body given loads or displacements on the body.
- A material model for a material.
- Material data for a certain solid or fluid.

When defining the *inner ring object*, it is natural to use and combine several of the basic objects defined above in order to give an appropriate description of a general *simple ring object*. We then create the *inner ring* machine element object, by inheriting from the general ring object (e.g. *simple ring*), and adding properties that are special for the inner ring. This is also done for the outer ring and the rollers, see figure 4.

Figure 5: *Inheritance and instantiation of connection objects for the rolling bearing.*

4 Basic definitions for the rolling bearing model

Here we start giving the theory of the rolling bearing example. This is in order to illustrate the reasoning in the previous sections. The aim is to retain the general structure of a machine element without getting lost in detailed analysis. Therefore, normal bearing theory is not used for the details of the model. Instead, a simplified version is used.

4.1 Notation

The notation used here is both the *direct matrix notation* of vectors and tensors, and the *vector-dyadic notation*. The right-handed rectangular Cartesian coordinate system is used.

a	a scalar
$\hat{\mathbf{e}}_i$	unit base vector
$\vec{\boldsymbol{a}}$	$a_i\hat{\mathbf{e}}_i$ a vector or first-order tensor
$\boldsymbol{a}, [a]$	$a_{ij}\hat{\mathbf{e}}_i\hat{\mathbf{e}}_j$ a second-order tensor or matrix
$a = \vec{\boldsymbol{b}} \cdot \vec{\boldsymbol{c}}$	$b_{ij}c_j$ (or $\vec{\boldsymbol{b}}^T \cdot \vec{\boldsymbol{c}}$) dot product of two vectors
$\vec{\boldsymbol{a}} = \vec{\boldsymbol{b}} \times \vec{\boldsymbol{c}}$	cross product of two vectors
$\boldsymbol{a}^{\mathrm{T}}$	transpose of $\boldsymbol{a}$
$\vec{\boldsymbol{a}} = \boldsymbol{b} \cdot \vec{\boldsymbol{c}}$	$b_{ij}c_j\hat{\mathbf{e}}_i$ dot product of a second-order tesnosr and a vector
$\boldsymbol{a} = \boldsymbol{b} \cdot \boldsymbol{C}$	$b_{ij}c_{jk}\hat{\mathbf{e}}_i\hat{\mathbf{e}}_k$ dot product of two second-order tensors

The first subscript mainly denotes what object the variable refers to. The second and third subscripts define additional attributes. If the superscript is enclosed in parenthesis, it defines in which coordinate system the

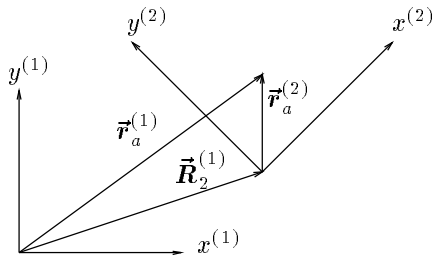


Figure 6: *The vectors describing a point a in the two coordinate systems 1 and 2.*

vector is expressed. For transformation matrices there are two superscripts enclosed in parentheses to define the two coordinate systems involved.

4.2 Transformations

The coordinate system 2 is formed from 1 by three successive rotations around the three axes (denoted 1, 2, and 3). After each rotation a new intermediate system is created in which the next rotation is done. The final system is achieved after the third rotation. This does not use the classical Euler angles. Thus, for pure rotation:

$$\vec{r}^{(2)} = \boldsymbol{A}^{(2,1)}\vec{r}^{(1)} \tag{1}$$

$$\boldsymbol{A}^{(2,1)} = \begin{bmatrix} c_{2,3}^{(1)} & s_{2,3}^{(1)} & 0 \\ -s_{2,3}^{(1)} & c_{2,3}^{(1)} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_{2,2}^{(1)} & 0 & -s_{2,2}^{(1)} \\ 0 & 1 & 0 \\ s_{2,2}^{(1)} & 0 & c_{2,2}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_{2,1}^{(1)} & s_{2,1}^{(1)} \\ 0 & -s_{2,1}^{(1)} & c_{2,1}^{(1)} \end{bmatrix} \tag{2}$$

where the notations c and s means $\cos\varphi$ and $\sin\varphi$, e.g. $c_{2,1}^{(1)} = \cos\varphi_{2,1}^{(1)}$. The matrix is orthogonal, therefore the transpose is the inverse.

$$\vec{r}^{(1)} = \boldsymbol{A}^{(1,2)}\vec{r}^{(2)} \tag{3}$$

$$\boldsymbol{A}^{(1,2)} = \boldsymbol{A}^{(2,1)-1} = \boldsymbol{A}^{(2,1)T} \tag{4}$$

We define the function:

$$\boldsymbol{A}^{(2,1)} = \text{rotmat}(\vec{\boldsymbol{\varphi}}_2^{(1)}) \tag{5}$$

$$\vec{\boldsymbol{\varphi}}_2^{(1)} = \begin{bmatrix} \varphi_{2,1}^{(1)} \\ \varphi_{2,2}^{(1)} \\ \varphi_{2,3}^{(1)} \end{bmatrix} \tag{6}$$

Vectors depending on the motion of coordinate systems

Some physical vectors depends on the motion of two coordinate systems, e.g. location and motion of a point, or time derivatives dependent on the coordinate systems.

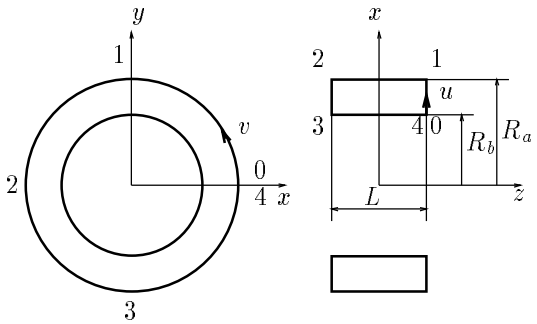


Figure 7: *A parametric body of revolution.*

In this rolling bearing example we perform only static analysis and have no time derivatives. Therefore we can use a simple rule, "point vectors are given relative to the origin of the coordinate system they are expressed in", and need not denote this explicitly. For a point we have:

$$\vec{r}_{\text{a}}^{(1)} = \vec{\boldsymbol{R}}_2^{(1)} + \boldsymbol{A}^{(1,2)}\vec{r}_{\text{a}}^{(2)} \tag{7}$$

$$\vec{r}_{\text{a}}^{(2)} = \boldsymbol{A}^{(2,1)}(\vec{r}_{\text{a}}^{(1)} - \vec{\boldsymbol{R}}_2^{(1)}) \tag{8}$$

where the vector $\vec{\boldsymbol{R}}_2^{(1)}$ is the location of the origin for the 2-system expressed in the 1-system. See figure 6.

Vectors independent on the motion of coordinate systems

A force does not depend on the translation of the coordinate systems.

$$\vec{\boldsymbol{F}}^{(1)} = \boldsymbol{A}^{(1,2)}\vec{\boldsymbol{F}}^{(2)} \tag{9}$$

Neither does a unit vector or a pure moment.

$$\vec{\boldsymbol{n}}^{(1)} = \boldsymbol{A}^{(1,2)}\vec{\boldsymbol{n}}^{(2)} \tag{10}$$

$$\vec{\boldsymbol{M}}^{(1)} = \boldsymbol{A}^{(1,2)}\vec{\boldsymbol{M}}^{(2)} \tag{11}$$

4.3 A body of revolution

We define a body of revolution as shown in figure 7. As seen it has a very simple geometric shape, however those simplifications are not utilized when analyzing the model. The theory is general but it is illustrated here with a very simple example.

The geometry is given by a parametric surface $\vec{\boldsymbol{r}}(u,v)$ expressed in the local Cartesian coordinate system of the body. A surface of revolution is used and the profile curve is defined in the $x-z$ plane as a function of the parameter u. Four curve segments are used. The couplings between the segments are C^0, i.e. equal position but not equal derivatives. However, the segments are of higher order ($\geq C^2$). The parameter interval is 1 for each segment, which means that the total parameter interval is: $u \in [0,4]$.

The profile curve is rotated with the parameter v. One segment is used with a parameter interval $v \in [0,4]$

which corresponds to $[0, 2\pi]$. The surface is closed with a positive normal vector directed outwards, so we have in fact a representation of a solid. The surface of a ring is:

$$\vec{\boldsymbol{r}}(u,v) = \begin{bmatrix} x(u)\cdot\cos(\frac{\pi v}{2}) \\ x(u)\cdot\sin(\frac{\pi v}{2}) \\ z(u) \end{bmatrix} \quad (12)$$

$$u \in [u_{\min}, u_{\max}] \quad u_{\min} = 0 \quad u_{\max} = 4 \quad (13)$$

$$v \in [v_{\min}, v_{\max}] \quad v_{\min} = 0 \quad v_{\max} = 4 \quad (14)$$

$$x(u) = \begin{cases} R_b + (R_a - R_b)u & 0 \le u < 1 \\ R_a & 1 \le u < 2 \\ R_b + (R_a - R_b)(3-u) & 2 \le u < 3 \\ R_b & 3 \le u < 4 \end{cases} \quad (15)$$

$$z(u) = \begin{cases} \frac{1}{2}L & 0 \le u < 1 \\ -L(u - \frac{3}{2}) & 1 \le u < 2 \\ -\frac{1}{2}L & 2 \le u < 3 \\ L(u - \frac{7}{2}) & 3 \le u < 4 \end{cases} \quad (16)$$

where R_a is the outer radius of the ring, R_b the inner radius, and L the axial length of the ring. Equation 12 is the definition of a parametric surface of revolution.

The normal $\vec{\boldsymbol{n}}(u,v)$ of the surface is needed. We have, using the partial derivatives $\frac{\partial \vec{\boldsymbol{r}}}{\partial u}$, $\frac{\partial \vec{\boldsymbol{r}}}{\partial v}$:

$$\vec{\boldsymbol{n}}(u,v) = \frac{\frac{\partial \vec{\boldsymbol{r}}}{\partial u} \times \frac{\partial \vec{\boldsymbol{r}}}{\partial v}}{|\frac{\partial \vec{\boldsymbol{r}}}{\partial u} \times \frac{\partial \vec{\boldsymbol{r}}}{\partial v}|} \quad (17)$$

4.4 Static equilibrium

The sum of forces acting on body a expressed in coordinate system 1 gives the force equilibrium.

$$\sum_{i=1}^{n} \vec{\boldsymbol{F}}_{a,i}^{(1)} = \boldsymbol{0} \quad (18)$$

The the sum of moments acting on body a expressed in coordinate system 1 gives the moment equilibrium.

$$\sum_{i=1}^{n} (\vec{\boldsymbol{M}}_{a,i}^{(1)} + \vec{\boldsymbol{p}}_{a,i}^{(1)} \times \vec{\boldsymbol{F}}_{a,i}^{(1)}) = \boldsymbol{0} \quad (19)$$

The moment point is the origin of system 1. The point $\vec{\boldsymbol{p}}_{a,i}^{(1)}$ is the location where the force $\vec{\boldsymbol{F}}_{a,i}^{(1)}$ is applied.

4.5 Contact between bodies

Consider a contact between two bodies 1 and 2, see figure 8. The gap/intersection $\Delta_{1,2}$ and the parametric location $u_{1,2}, v_{1,2}, u_{2,1}, v_{2,1}$ on the two contacting bodies are calculated. This is done in a common coordinate system c.

$$\vec{\boldsymbol{n}}_1^{(c)}(u_{1,2}, v_{1,2}) + \vec{\boldsymbol{n}}_2^{(c)}(u_{2,1}, v_{2,1}) = \boldsymbol{0} \quad (20)$$

$$\Delta_{1,2}\vec{\boldsymbol{n}}_1^{(c)}(u_{1,2}, v_{1,2}) = \vec{\boldsymbol{r}}_2^{(c)}(u_{2,1}, v_{2,1}) - \vec{\boldsymbol{r}}_1^{(c)}(u_{1,2}, v_{1,2}) \quad (21)$$

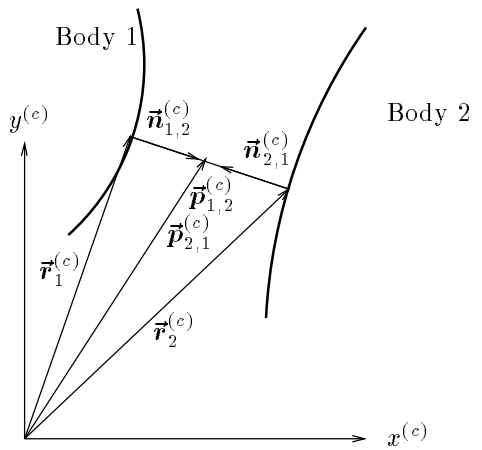


Figure 8: *A contact between two bodies. The gap is denoted* $\Delta_{1,2}$.

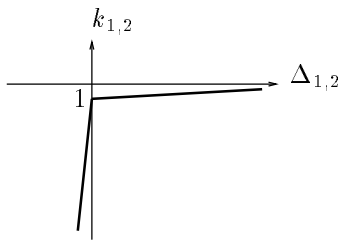


Figure 9: *The contact stiffness function* $k_{1,2}$. *The variable* $\Delta_{1,2}$ *is negative for intersection and positive for gap.*

The gap/intersection causes a normal force at the contact points $\vec{\boldsymbol{p}}_{1,2}^{(c)}$ and $\vec{\boldsymbol{p}}_{2,1}^{(c)}$. We assume no friction, therefore we have only a normal force.

$$\vec{\boldsymbol{F}}_{1,2}^{(c)} = k_{1,2}(\Delta_{1,2}) \cdot \vec{\boldsymbol{n}}_{1,2}^{(c)} \quad (22)$$

$$\vec{\boldsymbol{F}}_{2,1}^{(c)} = -\vec{\boldsymbol{F}}_{1,2}^{(c)} \quad (23)$$

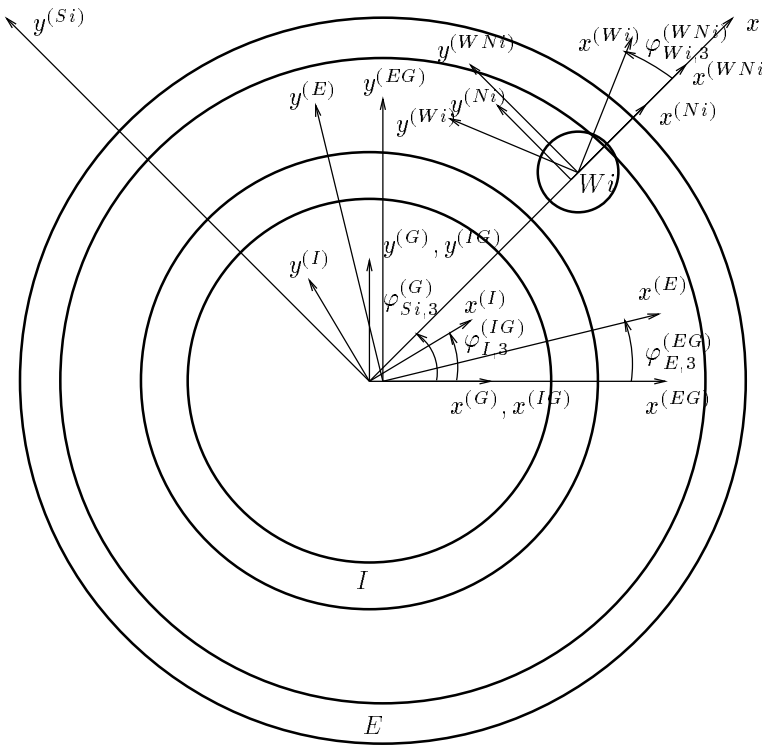
The contact stiffness function $k_{1,2}(\Delta_{1,2})$ is a function of the gap/intersection $\Delta_{1,2}$. The $\Delta_{1,2}$ is negative for an intersection and positive for a gap. Here we choose a simple linear model (see figure 9) but other more complex models could be used. Since we have no friction, there is no spin moment in the contact.

$$\vec{\boldsymbol{M}}_{1,2}^{(c)} = \boldsymbol{0} \quad (24)$$

$$\vec{\boldsymbol{M}}_{2,1}^{(c)} = -\vec{\boldsymbol{M}}_{1,2}^{(c)} \quad (25)$$

5 Model definition of a rolling bearing

We define a model of a "cylindrical roller bearing" without a cage, see figure 10, using the basic definitions from the previous section. The aim is to make as simple a model as possible while still keeping a general and complete structure of the machine element. In other words:



The coordinate systems:

G	Global, inertial
E	Outer ring, body-fixed
EG	Outer ring, no rotation relative G
I	Inner ring, body-fixed
IG	Inner ring, no rotation relative G
Wi	Roller i, body-fixed
WNi	Roller i, no rotation relative Ni
Ni	Nominal radial position of roller i
Si	Circumferential position of roller i

Figure 10: *The cylindrical roller bearing model with coordinate systems.*

details can be simple, but the structure should be realistic. The complete model and set of equations are given in [6].

5.1 Outer ring denoted E

The body of the outer ring is defined in the local body fixed coordinate system using equation 12:

$$\vec{\boldsymbol{r}}^{(E)}_E(u,v) = \vec{\boldsymbol{r}}(u,v,u_{max,E},v_{max,E},R_{a,E},R_{b,E},L_E) \tag{26}$$

where we set $u_{max,E} = 4$, $v_{max,E} = 4$, and $L_E = L$.

The normal $\vec{\boldsymbol{n}}^{(E)}_E(u,v)$ to the surface is defined using equations 26 and 17.

The coordinate systems are defined in a chain starting at the local system E and ending at the inertial system G. Definition of coordinate system E in coordinate system EG:

$$\vec{\boldsymbol{\varphi}}^{(EG)}_E = \begin{bmatrix} 0 \\ 0 \\ \varphi^{(EG)}_{E,3} \end{bmatrix} \tag{27}$$

$$\vec{\boldsymbol{R}}^{(EG)}_E = \mathbf{0} \tag{28}$$

$$\boldsymbol{A}^{(E,EG)} = \mathrm{rotmat}(\vec{\boldsymbol{\varphi}}^{(EG)}_E) \tag{29}$$

System E does rotate relative to system EG.

Definition of coordinate system EG in coordinate system G:

$$\vec{\boldsymbol{\varphi}}^{(G)}_{EG} = \mathbf{0} \tag{30}$$

$$\vec{\boldsymbol{R}}^{(G)}_{EG} = \begin{bmatrix} R^{(G)}_{EG,1} \\ 0 \\ 0 \end{bmatrix} \tag{31}$$

$$\boldsymbol{A}^{(EG,G)} = \mathrm{rotmat}(\vec{\boldsymbol{\varphi}}^{(G)}_{EG}) \tag{32}$$

System EG does not rotate relative to system G. The only displacement is in the x-direction. There is no misalignment.

5.2 The inner ring denoted I

The definition of the inner ring is done analogously to the outer ring.

$$\vec{\boldsymbol{r}}^{(I)}_I(u,v) = \vec{\boldsymbol{r}}(u,v,u_{max,I},v_{max,I},R_{a,I},R_{b,I},L_I) \tag{33}$$

where $u_{max,I} = 4$, $v_{max,I} = 4$, and $L_I = L$.

The definition of coordinate system I in coordinate system IG:

$$\vec{\boldsymbol{\varphi}}^{(IG)}_I = \begin{bmatrix} 0 \\ 0 \\ \varphi^{(IG)}_{I,3} \end{bmatrix} \tag{34}$$

$$\vec{\boldsymbol{R}}^{(IG)}_I = \mathbf{0} \tag{35}$$

System I rotates relative to system IG.

Definition of coordinate system IG in coordinate system G:

$$\vec{\boldsymbol{\varphi}}^{(G)}_{IG} = \mathbf{0} \tag{36}$$

$$\vec{\boldsymbol{R}}^{(G)}_{IG} = \mathbf{0} \tag{37}$$

System IG does not rotate relative to system G. There is no displacement or misalignment.

5.3 Rolling elements

We have n rolling elements. For rolling element i denoted Wi, we have:

$$\vec{\boldsymbol{r}}_{Wi}^{(Wi)}(u,v) = \vec{\boldsymbol{r}}(u,v,u_{max,Wi},v_{max,Wi},R_{a,Wi},R_{b,Wi},L_{Wi}) \tag{38}$$

where $u_{max,Wi} = 3$, $v_{max,Wi} = 4$, $L_{Wi} = L$, and $R_{b,Wi} = 0$.

Definition of the coordinate system Wi in coordinate system WNi:

$$\vec{\boldsymbol{\varphi}}_{Wi}^{(WNi)} = \begin{bmatrix} 0 \\ 0 \\ \varphi_{Wi,3}^{(WNi)} \end{bmatrix} \tag{39}$$

$$\vec{\boldsymbol{R}}_{Wi}^{(WNi)} = \mathbf{0} \tag{40}$$

There is no skew or tilt.

Definition of the coordinate system WNi in coordinate system Ni:

$$\vec{\boldsymbol{\varphi}}_{WNi}^{(Ni)} = \mathbf{0} \tag{41}$$

$$\vec{\boldsymbol{R}}_{WNi}^{(Ni)} = \begin{bmatrix} R_{WNi,1}^{(Ni)} \\ 0 \\ 0 \end{bmatrix} \tag{42}$$

Definition of the *nominal position* coordinate system Ni in the coordinate system Si. Here we simply chose to have equal clearance in both contacts between the roller and the rings.

$$\vec{\boldsymbol{\varphi}}_{Ni}^{(Si)} = \mathbf{0} \tag{43}$$

$$\vec{\boldsymbol{R}}_{Ni}^{(Si)} = \begin{bmatrix} R_{Ni,1}^{(Si)} \\ 0 \\ 0 \end{bmatrix} \tag{44}$$

$$R_{Ni,1}^{(Si)} = \frac{1}{2}(R_{a,I} + R_{b,E}) \tag{45}$$

Definition of the coordinate system Si in the coordinate system G:

$$\vec{\boldsymbol{\varphi}}_{Si}^{(G)} = \begin{bmatrix} 0 \\ 0 \\ \varphi_{Si,3}^{(G)} \end{bmatrix} \tag{46}$$

$$\vec{\boldsymbol{R}}_{Si}^{(G)} = \mathbf{0} \tag{47}$$

$$\varphi_{Si,3}^{(G)} = \varphi_{S1,3}^{(G)} + \frac{i-1}{n}2\pi \tag{48}$$

As seen, equal spacing of the n rolling elements is specified.

5.4 The contacts between the rolling element Wi and the rings

The location of the contact $(u_{E,Wi}, v_{E,Wi}, u_{Wi,E}, v_{Wi,E})$ and the intersection/gap $(\Delta_{Wi,E})$ between the roller Wi and the outer ring E is found applying equations 20–21.

$$\vec{\boldsymbol{n}}_{E,Wi}^{(WNi)} + \vec{\boldsymbol{n}}_{Wi,E}^{(WNi)} = \mathbf{0} \tag{49}$$

$$\Delta_{Wi,E}\vec{\boldsymbol{n}}_{Wi,E}^{(WNi)} = \vec{\boldsymbol{r}}_{E,Wi}^{(WNi)} - \vec{\boldsymbol{r}}_{Wi,E}^{(WNi)} \tag{50}$$

Assuming equal elasticity of the two contacting surfaces, the contact point is:

$$\vec{\boldsymbol{p}}_{Wi,E}^{(WNi)} = \vec{\boldsymbol{r}}_{Wi,E}^{(WNi)} + \frac{1}{2}\Delta_{Wi,E}\vec{\boldsymbol{n}}_{Wi,E}^{(WNi)} \tag{51}$$

The contact with the inner ring I is treated analogously.

$$\vec{\boldsymbol{n}}_{I,Wi}^{(WNi)} + \vec{\boldsymbol{n}}_{Wi,I}^{(WNi)} = \mathbf{0} \tag{52}$$

$$\Delta_{Wi,I}\vec{\boldsymbol{n}}_{Wi,I}^{(WNi)} = \vec{\boldsymbol{r}}_{I,Wi}^{(WNi)} - \vec{\boldsymbol{r}}_{Wi,I}^{(WNi)} \tag{53}$$

$$\vec{\boldsymbol{p}}_{Wi,I}^{(WNi)} = \vec{\boldsymbol{r}}_{Wi,I}^{(WNi)} + \frac{1}{2}\Delta_{Wi,I}\vec{\boldsymbol{n}}_{Wi,I}^{(WNi)} \tag{54}$$

The contact forces and moments

Absence of friction gives only normal forces on all the contacts between rings and roller. The outer ring contact:

$$\vec{\boldsymbol{F}}_{Wi,E}^{(WNi)} = k_{Wi,E}\cdot\vec{\boldsymbol{n}}_{Wi,E}^{(WNi)} \tag{55}$$

$$\vec{\boldsymbol{F}}_{Wi,E}^{(WNi)} + \vec{\boldsymbol{F}}_{E,Wi}^{(WNi)} = \mathbf{0} \tag{56}$$

$$\vec{\boldsymbol{M}}_{Wi,E}^{(WNi)} = \mathbf{0} \tag{57}$$

$$\vec{\boldsymbol{M}}_{Wi,E}^{(WNi)} + \vec{\boldsymbol{M}}_{E,Wi}^{(WNi)} = \mathbf{0} \tag{58}$$

and the inner ring contact:

$$\vec{\boldsymbol{F}}_{Wi,I}^{(WNi)} = k_{Wi,I}\cdot\vec{\boldsymbol{n}}_{Wi,I}^{(WNi)} \tag{59}$$

$$\vec{\boldsymbol{F}}_{Wi,I}^{(WNi)} + \vec{\boldsymbol{F}}_{I,Wi}^{(WNi)} = \mathbf{0} \tag{60}$$

$$\vec{\boldsymbol{M}}_{Wi,I}^{(WNi)} = \mathbf{0} \tag{61}$$

$$\vec{\boldsymbol{M}}_{Wi,I}^{(WNi)} + \vec{\boldsymbol{M}}_{I,Wi}^{(WNi)} = \mathbf{0} \tag{62}$$

5.5 Equilibrium of ring E

The equlibrium is taken in coordinate system EG which is not rotating relative to system G and is situated at the centre of the body E. The force equilibrium is:

$$\sum_{i=1}^{n}\vec{\boldsymbol{F}}_{E,Wi}^{(EG)} + \vec{\boldsymbol{F}}_{E,ext}^{(EG)} = \mathbf{0} \tag{63}$$

The moment equilibrium taken at the origin of system EG is:

$$\sum_{i=1}^{n}\left(\vec{\boldsymbol{M}}_{E,Wi}^{(EG)} + \vec{\boldsymbol{p}}_{E,Wi}^{(EG)}\times\vec{\boldsymbol{F}}_{E,Wi}^{(EG)}\right) + \vec{\boldsymbol{M}}_{E,ext}^{(EG)} + \vec{\boldsymbol{p}}_{E,ext}^{(EG)}\times\vec{\boldsymbol{F}}_{E,ext}^{(EG)} = \mathbf{0} \tag{64}$$

Loading from the contacts with rolling elements

For each contact point there is a force and a moment.

$$\left\{ \begin{array}{l} \vec{\boldsymbol{F}}_{E,Wi}^{(WNi)} \\ \vec{\boldsymbol{M}}_{E,Wi}^{(WNi)} \\ \vec{\boldsymbol{p}}_{E,Wi}^{(WNi)} \end{array} \right. \qquad i \in [1,n] \tag{65}$$

This must be transformed to the coordinate system EG used for the equilibrium equations. It is accomplished by traversing a chain of coordinate systems. For the force we have:

$$\vec{\boldsymbol{F}}_{E,Wi}^{(EG)} = \boldsymbol{A}^{(EG,G)} \vec{\boldsymbol{F}}_{E,Wi}^{(G)} \tag{66}$$

$$\vec{\boldsymbol{F}}_{E,Wi}^{(G)} = \boldsymbol{A}^{(G,Si)} \vec{\boldsymbol{F}}_{E,Wi}^{(Si)} \tag{67}$$

$$\vec{\boldsymbol{F}}_{E,Wi}^{(Si)} = \boldsymbol{A}^{(Si,Ni)} \vec{\boldsymbol{F}}_{E,Wi}^{(Ni)} \tag{68}$$

$$\vec{\boldsymbol{F}}_{E,Wi}^{(Ni)} = \boldsymbol{A}^{(Ni,WNi)} \vec{\boldsymbol{F}}_{E,Wi}^{(WNi)} \tag{69}$$

which gives:

$$\vec{\boldsymbol{F}}_{E,Wi}^{(EG)} = \boldsymbol{A}^{(EG,WNi)} \vec{\boldsymbol{F}}_{E,Wi}^{(WNi)} \tag{70}$$

Transforming the contact point into system EG:

$$\vec{\boldsymbol{p}}_{E,Wi}^{(EG)} = \boldsymbol{A}^{(EG,G)} (\vec{\boldsymbol{p}}_{E,Wi}^{(G)} - \vec{\boldsymbol{R}}_{EG}^{(G)}) \tag{71}$$

$$\vec{\boldsymbol{p}}_{E,Wi}^{(G)} = \vec{\boldsymbol{R}}_{Si}^{(G)} + \boldsymbol{A}^{(G,Si)} \vec{\boldsymbol{p}}_{E,Wi}^{(Si)} \tag{72}$$

$$\vec{\boldsymbol{p}}_{E,Wi}^{(Si)} = \vec{\boldsymbol{R}}_{Ni}^{(Si)} + \boldsymbol{A}^{(Si,Ni)} \vec{\boldsymbol{p}}_{E,Wi}^{(Ni)} \tag{73}$$

$$\vec{\boldsymbol{p}}_{E,Wi}^{(Ni)} = \vec{\boldsymbol{R}}_{WNi}^{(Ni)} + \boldsymbol{A}^{(Ni,WNi)} \vec{\boldsymbol{p}}_{E,Wi}^{(WNi)} \tag{74}$$

Finally, expressing a pure moment in system EG is similar to expressing the force:

$$\vec{\boldsymbol{M}}_{E,Wi}^{(EG)} = \boldsymbol{A}^{(EG,WNi)} \vec{\boldsymbol{M}}_{E,Wi}^{(WNi)} \tag{75}$$

We have now expressed the contact forces, moments and contact points in the system EG.

$$\left\{ \begin{array}{l} \vec{\boldsymbol{F}}_{E,Wi}^{(EG)} \\ \vec{\boldsymbol{M}}_{E,Wi}^{(EG)} \\ \vec{\boldsymbol{p}}_{E,Wi}^{(EG)} \end{array} \right. \qquad i \in [1,n] \tag{76}$$

External loads and moments

External forces and moments are applied at a certain point.

$$\left\{ \begin{array}{l} \vec{\boldsymbol{F}}_{E,ext}^{(EG)} \\ \vec{\boldsymbol{M}}_{E,ext}^{(EG)} \\ \vec{\boldsymbol{p}}_{E,ext}^{(EG)} \end{array} \right. \tag{77}$$

We choose to apply the external support at the centre of the ring:

$$\vec{\boldsymbol{p}}_{E,ext}^{(EG)} = \boldsymbol{0} \tag{78}$$

5.6 Equilibrium of ring I

The method of taking equilibrium and transforming it between coordinate systems is applied analogously with the case for the outer ring. Here we just give the equilibrium equations. The equlibrium is taken in coordinate system IG which is not rotating relative to system G and is situated at the centre of the body I.

$$\sum_{i=1}^{n} \vec{\boldsymbol{F}}_{I,Wi}^{(IG)} + \vec{\boldsymbol{F}}_{I,ext}^{(IG)} = \boldsymbol{0} \tag{79}$$

$$\sum_{i=1}^{n} \left(\vec{\boldsymbol{M}}_{I,Wi}^{(IG)} + \vec{\boldsymbol{p}}_{I,Wi}^{(IG)} \times \vec{\boldsymbol{F}}_{I,Wi}^{(IG)} \right) + \vec{\boldsymbol{M}}_{E,ext}^{(IG)} + \vec{\boldsymbol{p}}_{I,ext}^{(IG)} \times \vec{\boldsymbol{F}}_{I,ext}^{(IG)} = \boldsymbol{0} \tag{80}$$

5.7 Equilibrium of of rolling element Wi

The equilibrium is taken in coordinate system WNi. The force equilibrium is:

$$\vec{\boldsymbol{F}}_{Wi,I}^{(WNi)} + \vec{\boldsymbol{F}}_{Wi,E}^{(WNi)} + \vec{\boldsymbol{F}}_{Wi,ext}^{(WNi)} = \boldsymbol{0} \tag{81}$$

and the moment equilibrium at the origin of system WNi is:

$$\vec{\boldsymbol{M}}_{Wi,I}^{(WNi)} + \vec{\boldsymbol{M}}_{Wi,E}^{(WNi)} + \vec{\boldsymbol{M}}_{Wi,ext}^{(WNi)} + \vec{\boldsymbol{p}}_{Wi,I}^{(WNi)} \times \vec{\boldsymbol{F}}_{Wi,I}^{(WNi)} \vec{\boldsymbol{p}}_{Wi,E}^{(WNi)} \times \vec{\boldsymbol{F}}_{Wi,E}^{(WNi)} + \vec{\boldsymbol{p}}_{Wi,ext}^{(WNi)} \times \vec{\boldsymbol{F}}_{Wi,ext}^{(WNi)} = \boldsymbol{0} \tag{82}$$

The forces and moments caused by contacts with the rings and acting on the rolling element Wi are:

$$\left\{ \begin{array}{l} \vec{\boldsymbol{F}}_{Wi,I}^{(WNi)} \\ \vec{\boldsymbol{M}}_{Wi,I}^{(WNi)} \\ \vec{\boldsymbol{p}}_{Wi,I}^{(WNi)} \end{array} \right. \tag{83}$$

$$\left\{ \begin{array}{l} \vec{\boldsymbol{F}}_{Wi,E}^{(WNi)} \\ \vec{\boldsymbol{M}}_{Wi,E}^{(WNi)} \\ \vec{\boldsymbol{p}}_{Wi,E}^{(WNi)} \end{array} \right. \tag{84}$$

Since we prescribe the motion in the circumferential direction, external loads are introduced. In a bearing with cage, the forces and moments will come from the interaction with the cage.

$$\left\{ \begin{array}{l} \vec{\boldsymbol{F}}_{Wi,ext}^{(WNi)} \\ \vec{\boldsymbol{M}}_{Wi,ext}^{(WNi)} \\ \vec{\boldsymbol{p}}_{Wi,ext}^{(WNi)} \end{array} \right. \tag{85}$$

We choose to apply the external support at the centre of the rolling element:

$$\vec{\boldsymbol{p}}_{Wi,ext}^{(WNi)} = \boldsymbol{0} \tag{86}$$

and the radial motion is not constrained by external forces.

$$\vec{\boldsymbol{F}}_{Wi,ext}^{(WNi)} = \left[\begin{array}{c} 0 \\ F_{Wi,ext,2}^{(WNi)} \\ F_{Wi,ext,3}^{(WNi)} \end{array} \right] \tag{87}$$

6 Solving the equation system

In this rolling bearing example many degrees of freedom have been eliminated in the model definition. Of the remaining variables, there are many valid combinations of input and output variables. Each combination yields a different final equation system. It is important that the modelling environment have good support for this variation in the solution process.

The input variables here chosen are displacements and rotations of the rings, rotations and circumferential position of the rolling elements. Then, the output variables becomes all the forces, moments, and radial position of the rolling elements. The analysis steps are:

1. Solve the equilibrium of the rolling elements.

   This will give the radial position of the rolling elements, the contact forces and moments, the external (from the cage) forces and moments acting on the rollers, the deformation in the contacts, and the location of the contacts.

2. Solve the equilibrium of the rings.

   This will give the external support loads and moments of the rings.

3. Calculate any other variables of interest.

Using a valid small angle approximation, a complete symbolic solution can be found for the equation system [6]. Not using this simplification, a much reduced equation system is still obtained through symbolic solution and simplification. This remaining equation system is then solved with numerical methods.

Given the mathematics for the rolling bearing model, the ObjectMath programming environment has been used for model definition, both symbolic and numeric solutions, and visualization of the rolling bearing.

7 Requirements on the ideal programming environment

Below, we make a quick overview of some desired properties of an ideal programming environment for mathematical modelling. Our current prototype implementation of this environment provides some support within each category, with an emphasis on model specification and model transformation.

Model specification

A machine element object, such as a ring, will have attributes and associated operations. As an example, attributes for a ring are the radius, a formula for parametric definition of the surface of the ring, a formula for elasticity properties etc. Associated operations can for example be to compute the normal vector to the surface of the ring, to compute the volume, or to graphically visualize it on the screen.

The programming environment should support the user in defining such objects. In addition, the creation of composite objects, and inheritance of attributes and equations from general objects to more specialized objects should be supported. It should be possible to model constraints and connections between physical objects within *connection* objects.

The use of such object-oriented programming techniques for modelling in equations eliminates unnecessary repetition of similar formulae, imposes a better structure on the mathematical model, and reduces the overall number of equations.

Model transformation

The user should interactively be able to transform equations and formulae associated with an object, with the help of a computer algebra system. Transformed formulae should be saved within new versions of the objects from which they were derived. It should be possible to specify which variables should be solved for, and which variables are input or output.

Choice of numerical solution routines

The user should be able to choose between a few built-in numerical solution and optimization routines, obtained from some appropriate numerical library. Specification of, e.g. iteration start values and stop conditions should be supported. All this information should be associated with the appropriate equation object.

Generation of numerical code and computation

Formulae and equations produced during the interactive model transformation phase should be used to generate efficient numerical code, e.g. in efficiently compiled languages such as FORTRAN or C++. Code for automatic call of selected numerical routines should automatically be generated.

Visualization of results

Resulting data should be amenable to visualization in two or three dimensions. By using parametric equations for defined objects, this can be obtained, e.g. by calling general 3D graphics routines which accept parametric surface definitions.

8 Currently available tools

In this section we make a quick overview of some currently available tools, their strengths and weaknesses. A more detailed review, including full references and discussion can be found in [7] [8]. So far, none of these

tools offer exactly what we want, although some provide a subset of the desired capabilities.

One group of tools and methods include *finite element analysis* (FEM), *Multibody systems analysis* programs [13], CAD (*Computer Aided Design*) and CAE (*Computer Aided Engineering*). Some of these tools could theoretically solve the above problems. However, they usually generate masses of low-level numerical data, which is not what we want. Also, they tend to be very inefficient, especially on non-linear problems. They provide no equational model development support.

Computer algebra systems such as *Macsyma*, *Maple*, *Reduce* and *Mathematica* are good at representing and manipulating symbolic formulae and equations, but are lacking model development support and efficient numerical computation. However, their symbolic capabilities are needed in our desired environment.

Matrix languages such as *APL*, *Fortran8X* and especially *MATLAB*, provide convenient programming of matrix, vector and linear algebra related problems. However, non-linear problems are not handled and symbolic formula capabilities are missing.

There are several numerical differential equation solving systems, e.g. *SIMNON* [4], which accept as input a set of differential equations expressed in an attractive high-level algebraic form. However, these systems lack algebraic capabilities and use bruteforce numerical methods to solve the differential equations. This often leads to totally unrealistic computation times, and generates vast amounts of low-level data.

Another class of systems are hybrid systems for *algebraic-numerical-graphical analysis*, including *GENTRAN*, *FINGER*, *MathStation* and *MaMis*. These systems combine a computer algebra system with capabilities for generating and executing numerical code, but have no support for object-oriented structuring of equation-based models and inheritance of equations.

9 ObjectMath - an object-oriented modelling language and environment

The modelling language ObjectMath has been designed to be a hybrid language, combining object oriented language constructs with a basic computer algebra language. Rather than inventing yet another computer algebra language, we decided to use an existing language as the computer algebra component. Our choice of Mathematica here is partly motivated by its readable and systematic syntax and language features compared to older languages such as Macsyma, and partly by its support of three-dimensional graphics.

One objective behind the design of the ObjectMath language was to be able to represent all relevant mathematical formulae and equations, including vectors, matrices and tensors. Another objective was to be able to group equations together into objects, and support inheritance of equations between classes to permit better re-use of mathematical models. Both of these objectives have been achieved within ObjectMath.

The ObjectMath language provides single inheritance. We have chosen to avoid multiple inheritance partly for reasons of simplicity, and partly because we have not yet seen any good applications of it in the examples examined. Multiple inheritance might however be included in a future version of ObjectMath, if we find some reasonable uses for it.

Currently the ObjectMath language does not include a mechanism for enforcing encapsulation. This is because we wanted to keep the language as simple as possible, both to facilitate the implementation of the prototype, and to make the language easy to learn for mechanical engineers who do not have much experience in object oriented programming. The next version of the ObjectMath language will include proper encapsulation.

9.1 Usage of inheritance

ObjectMath has been successfully used to model the rolling bearing mentioned previously. The bearing model contains approximately 200 equations when expressed in conventional mathematical form. The bearing model, when specified in ObjectMath is actually more compact than that, since the use of inheritance has eliminated some unnecessary copying of similar equations. The model size was reduced by a factor of two (from 588 lines to 308 lines) by re-use of equations through inheritance. This is a strong indication of the feasibility of this approach when applied to a realistic example.

9.2 Object oriented language features of ObjectMath

ObjectMath contains just a few constructs in order to allow for object oriented modelling and programming. Parameterized classes can be defined using the *CLASS* construct. As mentioned before, conventional single inheritance between classes is allowed.

The ObjectMath *INSTANCE* declaration is in a traditional sense both a class declaration and a declaration of one instance of this class. This makes the declaration of classes with a singleton instance compact. Classes implicitly declared in an ObjectMath *INSTANCE* declaration can be inherited from, just as classes declared with a *CLASS* declaration.

A symbolic array (or set) of objects can be created from one *INSTANCE* declaration, by adding an index variable in brackets to the instance name. This allows for the creation of large numbers of nearly identical objects, for example the rolling elements in a rolling bearing. To represent differences between such objects, methods that are dependent upon the array index of the instance can be used. The implementation makes it

Figure 11: *The Objectmath high-level modelling environment on the 200-equation rolling bearing example. The three-dimensional view of the bearing has been generated automatically from equations expressed in ObjectMath.*

possible to do computations with a symbolic number of elements in the array.

In comparison with the concept of prototype using standard object-oriented terminology [14], the *INSTANCE* construct serves as a class, i.e. a template, since class-variables and equations are duplicated at instantiation time, whereas they would have been shared in the case of inheritance from a prototype.

Both classes and instances can be parameterized. Parameters to both classes and instances allow equations in the bodies of these declarations to be more generally applicable. In addition to the keywords *CLASS* and *INSTANCE*, a *MODEL* keyword has been introduced to designate whole models.

9.3 Naming in ObjectMath

In addition to equations and variables, ordinary functions and procedures (=methods) can be associated with an object. Since the Mathematica context facility is used in our implementation of objects, the normal dot-notation for referring to intra-class definitions has been replaced by Mathematica's apostrophe notation for referring to definitions within a class. This can be seen in the example: `Coord'TransPoint[r[Ex][u,v],Ex,WNo[i]]`, where the `TransPoint` *method* within the `Coord` object is called.

One convention that we have used in this model is that each equation within an object is named, e.g. `Eq[1]`, `Eq[2]`, in order to allow reference or application of such equations within other objects. Another naming convention could be `Eq[SimpleRing1]`, `Eq[SimpleRing2]` for two equations within the SimpleRing class, which might be advantageous for avoiding name collisions when inheriting equations.

Another naming convention in this model is to let the symbol within the first pair of brackets always denote the coordinate system within which the relevant quantity is defined. For example, the quantity `r` in: `r[WNo[i]][u,v]` denotes the parametric surface function `r` of parameters `u` and `v`, defined within the `WNo[i]` coordinate system. Note that these two conventions are not part of the actual ObjectMath language - they just serve to make models more consistent and easier to understand.

9.4 Implementation of inheritance, classes and instances

Inheritance between classes, and between classes and instances is implemented by copying definitions of equations and other attributes. Only instance declarations actually give rise to generated Mathematica code, which of course contains all equations from classes inherited by the instance. An instance declaration with array brackets will generate one instance of Mathematica code and several data objects. For example, the instance declaration with array brackets: `INSTANCE BodyW[i] INHERITS Roller(W[i])` can give rise to several data objects (indexed by `i`) which can be used during numerical simulation of the model.

Figure 13: *Structure of the implemented environment.*

e.g `A[one]` is a new variable different from `A`. Underscore is used at the end of pattern matching variable names in matching rules.

```
INSTANCE BodyEx INHERITS SimpleRing(Ex)
 (* Coord system Ex is centered at object BodyEx*)
 p[ExG][Ext] := { 0, 0, 0 };
 (* Surface and normal in WNo[i] system *)
 r[WNo[i_]][u_, v_] :=
        Coord‘TransPoint[r[Ex][u, v], Ex, WNo[i]];
 n[WNo[i_]][u_, v_] :=
       Coord‘TransVector[n[Ex][u, v], Ex, WNo[i]];
 (* Equations *)
 Eq[1] := Sum[F[ExG][BodyW[i]], {i, 1, Wn}] +
             F[ExG][Ext] == { 0, 0, 0 };
 Eq[2] := Sum[M[ExG][BodyW[i]] +
        Cross[ConExW[i]‘p[ExG],
                  F[ExG][BodyW[i]]],{i,1,Wn}] +
            M[ExG][Ext] +
       Cross[p[ExG][Ext], F[ExG][Ext]] == {0,0,0};
 Eq[3][i_] := F[WNo[i]][BodyW[i]] ==
   Coord‘TransVector[F[ExG][BodyW[i]],ExG,WNo[i]];
 Eq[4][i_] := M[WNo[i]][BodyW[i]] ==
   Coord‘TransVector[M[ExG][BodyW[i]],ExG,WNo[i]];
END BodyEx;
```

9.5 Implementation status

ObjectMath has been changed several times during the design process, and will be extended in the future. A first prototype of the environment and language has been implemented using a combination of Gnu Lisp,

Scheme, Mathematica and C++, running on a Sun Sparcstation.

A parser and translator for the hybrid language ObjectMath has been implemented in Scheme. Object oriented language features such as classes, inheritance and instances have been implemented by transformations of the internal representation generated by the parser to appropriate Mathematica code.

The current prototype environment supports a separate window/emacs buffer for each class or instance definition. One window for each object definition simplifies future extensions such as versioning of equation objects. A graphical browsing tool, implemented using the ET++ class libraries [15], has recently been added. The environment has a separate interaction window where the user can interactively perform simplifications, substitutions and other transformations on model equations using Mathematica.

A protocol for transparently calling C++ functions and procedures has been developed, including passing arguments of vector and matrix type. Numerical C++ code can also be generated from symbolic expressions. This code is linked to numerical FORTRAN library routines. So far, the environment has been able to handle the formulae and equations of the 200 equation bearing example, which has been solved symbolically. For numerical solution of the bearing example, we are using the MINPACK routines HYBRD and HYBRJ [9] which solves systems of non-linear equations.

9.6 ObjectMath improvement areas

Mathematica's symbolic solution capabilities were somewhat weaker than expected, especially when trigonometric functions are involved. This can sometimes be remedied by adding extra transformation rules. However, the user is never stuck, since a solution strategy with smaller and easier steps is always possible by inserting additional equations and transformation steps. This supports our view that user interaction is necessary for the solution of realistic problems.

Another implementation problem is Mathematica's tendency to immediately apply all pattern matching rules, often making expanded formulae too big for solution. This can be remedied by extending the ObjectMath surface language to provide explicit control of the application of rule sets.

9.7 Related work on object oriented hybrid systems

A related system featuring a modelling language called Omola (Object-oriented MOdelling LAnguage) has been developed for control and process engineering applications [2][11]. Before doing simulations the model is turned into differential/algebraic/equation (DAE) form and then solved using standard DAE solvers.

To our knowledge, Omola and ObjectMath are currently the only systems for object-oriented equational model definition which support inheritance. Dymola [5] is an earlier system without inheritance, which has been used for larger practical applications than Omola, but does not allow interactive algebraic transformation of equations. Several examples using Dymola are described in [3].

In comparison to ObjectMath, the Omola language is not integrated with a computer algebra system, and does not permit interactive manipulation of equations. Equations represented in Omola are relatively simple. The emphasis is rather on automatic transformation and solution of simpler sets of equations in DAE form, and interactive model browsing tools.

Also, our application domain (machine elements), for which ObjectMath is developed, is different with an order of magnitude greater complexity of systems of equations. Because of this complexity, the user has to work in dialogue with the environment during the solution process. Thus our approach is to integrate the modelling notation with a computer algebra system to provide interactive formula manipulation. By allowing interactive formula manipulation, the ObjectMath environment will need to support versioning of equation objects in the future.

9.8 Other related work

An interesting computer algebra system that provides some object-oriented features called categories and domains (but not full inheritance) is SCRATCHPAD [10]. Kamal Abdali et.al [1] describes a new computer algebra system design based on an object-oriented style of implementation. This system views certain algebraic structures such as "rings", "groups" and "fields" as objects, which is quite different from the grouping of equations into objects and inheritance of equations provided by ObjectMath. Another system, called Cactus [12], is an object-oriented implementation of numerical linear algebra operations, with no symbolic or modelling capabilities.

10 Conclusions

There is a need for efficient high level tools and techniques in machine element analysis. We have seen that an object-oriented model description is natural for the machine element analyst and the models thus obtained will be both easier to understand, better structured, and allow a higher degree of re-use than conventional models. This conclusion is supported by the successful modelling of a realistic application, the rolling bearing example. This has been done using the ObjectMath language which supports object-oriented mathematical modelling in equations. We have also seen that the complexity of realistic systems of equations in machine element anal-

ysis makes it essential that the environment allows the user to interactively supply extra information to guide the solution process. However, once these extra solution steps have been supplied, the transformation from high-level model down to executable numerical code is largely automatic.

11 Acknowledgments

The authors wish to thank the Managing Director of the SKF Engineering & Research Center, Dr. H. Wittmeyer, for his permission to publish this paper. This project is supported by NUTEK - the Swedish Board for Technical Development, and by the SKF Engineering & Research Centre.

References

[1] S. K. Abdali, G. W. Cherry, and N. Soffier. An object oriented approach to algebra system design. In *Proc. of the ACM Symposium on Symbolic and Algebraic Computation.*, July 1986.

[2] M. Andersson. Omola — an object-oriented language for model representation. Licentiate thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, May 1990.

[3] F. E. Cellier. *Continuous System Modeling.* Springer Verlag, New York, 1991. ISBN 0-387-97502-0.

[4] H. Elmqvist. *An Interactive Simulation Program for Nonlinear Systems — User's Manual TFRT–3091.* Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1975.

[5] H. Elmqvist. *A Structured Model Language for Large Continuos Systems.* PhD thesis, Lund Institute of Technology, Lund, Sweden, May 1978.

[6] D. Fritzson and P. Fritzson. Equational modeling of machine elements — Applied to rolling bearings. Technical report, Department of Computer and Information Science, Linköping University, S-581 83, Linköping, Sweden, 1991.

[7] P. Fritzson and D. Fritzson. The need for high-level programming support in scientific computing - Applied to mechanical analysis. Technical report, Department of Computer and Information Science, Linköping University, S-581 83, Linköping, Sweden, 1991.

[8] P. Fritzson and D. Fritzson. The need for high-level programming support in scientific computing - Applied to mechanical analysis. *Computers & Structures*, 1992. To appear.

[9] B. S. Garbow, K. E. Hillstrom, and J. J. More. *User guide for MINPACK-1.* Argonne National Laboratory, Argonne, Illinois, USA, report anl-80-74 edition, March 1980.

[10] R. D. Jenks. A primer: 11 keys to new SCRATCHPAD. In J. Fitch, editor, *Proc. of EUROSAM 84 Int. Symposium on Symbolic and Algebraic Computation.* Springer Verlag, July 1984.

[11] S.E. Mattson and M. Andersson. A kernel for system representation. In *11-th IFAC World Congress, Tallin, Estonia, USSR, Aug13-17, 1990*, pages 91–96, 1990.

[12] J. A. McDonald. Object-oriented programming for linear algebra. In *Proc. of OOPSLA 1989 New Orleans*, 1989.

[13] W. Schiehlen, editor. *Multibody Systems Handbook.* Springer-Verlag, Berlin Heidelberg, Germany, 1990. (ISBN 0-387-51946-7).

[14] P. Wegner. Learning the language. *Byte*, pages 245–253, March 1989.

[15] A. Weinand, E. Gamma, and R. Marty. ET++ – an object-oriented application framework in C++. In *Proc. of OOPSLA '88*, September 1988.