Doctoral Thesis

An object-oriented design for efficient microsystem simulation

Author(s):
Emmenegger, Markus; Baltes, Henry

Publication Date:
1999

Permanent Link:
https://doi.org/10.3929/ethz-a-003856338

Rights / License:
In Copyright - Non-Commercial Use Permitted
An Object-Oriented Design for Efficient Microsystem Simulation

A thesis submitted to the

SWISS FEDERAL INSTITUTE OF TECHNOLOGY, ZURICH

for the degree of

Doctor of Technical Sciences

presented by

Markus Emmenegger

Dipl. Phys. ETH Zurich

Born October 6, 1969

Citizen of Luzern and Schüpfheim (LU), Switzerland

submitted on the recommendation of

Prof. Dr. Henry Baltes, supervisor

Prof. Dr. Jan G. Korvink, co-examiner

Prof. Dr. Niels Kuster, co-examiner

1999
CONTENTS

Contents 3
Abstract 1
Zusammenfassung 3

1 Introduction 5
1.1 Microsystems 5
1.2 Microsystem TCAD 6
1.3 Previous and Related Work 7
1.4 Major Results 7

2 Modeling 9
2.1 Lagrangian Equations of Motion 9
2.2 Model Equations 10
2.3 Discretization and Lagrangian Dynamics 23

3 Numerical Methods and Discretization Techniques 31
3.1 Analysis Methods 31
3.2 Discretization Methods 34
3.3 Treating the Time Dependence: Generating Algebraic Equations 42
3.4 Time Marching Algorithms 47
3.5 Enforcing Constraints 59

4 Object-Oriented Analysis and Design 71
4.1 Software Design Methods 71
4.2 Simulator Requirements and Use Cases 74
4.3 Design of the Simulator 76
4.4 Implementation Issues 88
4.5 Managing Software Development 88
Simulations 91
  5.1 Heat capacity measurement Structure 91
  5.2 CMOS MEMS Gas Sensor 94
  5.3 RF Switch 99
Outlook 101
Conclusion 105
Appendix 107
  A.1 The Rotational Square Gradient Identity 107
  A.2 UML Notation and Terminology 109
References 111
Acknowledgements 117
Curriculum Vitae 119
Abbreviations and Symbols 121
Index 127
ABSTRACT

This thesis presents a solution to the simulation requirements of an engineer designing electro-thermo-mechanical microsystems. To efficiently design microsystems, we require to understand the inner workings of these systems, know how energy is transferred inside the devices, and how the output signals come about.

A wide range of physical effects is exploited in microsystems. At the very minimum, electrical, thermal and mechanical effects, as well as the coupling between these domains, has to be taken into account. Additionally, views on the system are used to model the behavior: The stationary view focuses on the equilibrium state after a long time, whereas the transient view looks at the short-time behavior after a sudden change in the state of the system. Harmonic and eigenvalue approaches add insight into how the models behave in the time domain.

All these models are efficiently explored using a simulation tool. We present the FEMEngine, a multi-physics finite element simulator. It allows a designer to gain insight into the physical effects occurring in the devices being developed.

In the development of the FEMEngine, we have striven to achieve a consistent architecture to allow the flexible extension of the simulator. To this end, an object-oriented design method and programming language have been used. The thesis also presents the architecture we have created, and how this architecture allows extension to additional physical effects and solution methods.

At the end of the thesis, the power of the FEMEngine is illustrated by simulations of a selected set of microsystems.
ZUSAMMENFASSUNG


1 INTRODUCTION

This thesis presents the design and implementation of the FEMEngine, a multi-physics finite element simulator for microsystem analysis. We have used object-oriented technology to design and implement this flexible and extensible device simulation tool.

1.1 Microsystems

Microsystems are systems that have a small length scale, that is, they have features that are on the order of micrometers, $10^{-6} \text{m}$, in size. The microsystems considered here are fabricated using the technology developed for integrated circuits (ICs). This fabrication processes can be adapted to not only allow the creation of circuits, but also of microsensors and microactuators. The design of integrated circuits has heavily benefited from the availability of simulation tools for circuits. The benefit is especially large because the cost for producing a small series of microsystems is expensive. The cost per device only goes down when a large number of devices is manufactured. For this reason, a large number of cycles of design, small-series manufacturing, testing, and re-design is not feasible. Through simulation, the number of development cycles is decreased and the risk of failure is minimized.

Circuit simulators have been very successful in improving the IC development process. The scope of circuit simulation, however, is limited to the electrical domain. In the case of microsystems in general, which includes sensors and actuators, multiple domains of physics come into play. Thermal, electrical, electromagnetic, mechanical and fluidic effects must be considered, and also the coupling among domains has to be taken into account. Only in special cases, lumped approaches are feasible for these effects. In general, continuous models are required. These are discretized to obtain solutions to the partial differential equations. For the discretization of a broad range of partial differential equations, covering a large number of coupled physical effects, the finite element method has proven to be very flexible and effective.
1.2 Microsystem TCAD

The design of microsystems is highly computerized, and only computer-aided design (CAD) techniques make the creation of complex microsystems feasible. To support the design cycle, a wide range of software tools is required. We supplement the tool set with the FEMEngine, a simulator capable of predicting the performance of coupled, micro-electro-mechanical devices. The simulator must be embedded in a technology-CAD environment to be useful. To achieve this goal, an interface to a process simulator is required [ISE98], so that the geometry, which is determined by the layout and the fabrication process, does not have to be re-created.

The availability of a coupled microsystem simulator also helps the development of packaging, a highly critical field of microsystem design. The requirements imposed on the package of microsensors and microactuators are much more complex than those in standard IC technology. A simulator capable of predicting mechanical, thermal and electrical performance of a package is therefore highly desired.
1.3 Previous and Related Work

By choosing the finite element method, we were able to build on a very rich knowledge and theory base, obtained from using the method in other engineering applications. A previous project in our organization had already successfully applied the finite element method to microsystem simulation [Funk97]. Another project had provided an object-oriented implementation of the boundary-element method [Bächtold97]. We created the FEMEngine by synthesizing these two approaches. We were also inspired by efforts to implement commercial TCAD systems for IC technology [ISE98]. Commercially available microsystem TCAD systems are being developed by several companies [Coyote] [Microcosm] [Intellisense] [CFDRC] [Tanner].

1.4 Major Results

LAGRANGIAN EQUATIONS OF MOTION

We have developed a theory framework which allows a consistent modelling of the various physical effects that are exploited in microsystems. It is based on a Lagrangian formulation. The framework also simplifies the derivation of discretization schemes. The physical models are presented in Chapter 2.

TIME-DEPENDENT SIMULATION

We have extended our simulation capability to the time-domain, allowing the study of time-dependent phenomena. We have implemented harmonic, transient and eigenvalue analysis capabilities. The analysis methods are discussed in Chapter 3.
1 Introduction

CONSTRAINTS

We have found a consistent theory and implementation to handle constrained systems. The flexible forms of enforcing constraints are presented in Chapter 3.

IMPLEMENTATION

We have created an innovative, object-oriented implementation of the finite element method. Sub-systems with innovative architectures, such as the material database and the discretization sub-system, have been designed and implemented using object-oriented technology. The architecture of the FEMEngine is presented in Chapter 4.

APPLICATIONS

We have applied the new simulation tool to real-life microsystems. The examples show the variety of analysis methods we have implemented, ranging from stationary to harmonic and eigen-mode analysis. The results of the simulations are presented in Chapter 5.
2 Modeling

To better understand the inner workings of the microsystem devices we develop, we require representative physical models. The effects that govern the processes in these devices are classical and quantum-mechanical solid state and fluidic effects. We require the equations of motion for the fields in the device, as their solution lets us track the flux of energy, the displacement and the field distributions in the device. The goal of modeling is to mathematically express these equations of motion in a compact form. Here, we follow a consistent Lagrange approach to obtain these equations, because this has the advantage that a large number of coupled effects can be handled using one single recipe, factoring out the common patterns in the derivations. Furthermore, one finds that using this formalism, a consistent discretization can also be found naturally. The discretization obtained turns out to be the finite element method, which we chose as the discretization method to be used in this project.

2.1 Lagrangian Equations of Motion

We now present a method with which to derive the equations of motion of a continuous field in a solid. It is based on a continuum Lagrangian formulation, generalized to include dissipative effects and external forces. To obtain the equations of motion, which will be the partial differential equations for the fields, expressions for the stored and dissipated energy are required, and a recipe to obtain equations of motion from these energy expressions. The recipe is the continuum Euler-Lagrange equation

\[
\frac{d}{dt} \nabla_q A - \nabla_q (A) + \nabla_q \phi + \nabla \cdot (\nabla_{\nu} \phi(A) + \nabla_{\nu} \phi) = \alpha
\]  

(2.1)
2 Modeling

The derivation of this expression is based on the principle of least action [Nelson79]. The concept of a dissipative content function is due to [Layton98].

Table 2.1 The Symbols used in the continuum Euler-Lagrange equation (2.1)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q(x, t)$</td>
<td>Vector field of the generalized displacements</td>
</tr>
<tr>
<td>$\Lambda = \tau^{\delta} - \nu$</td>
<td>Lagrange density</td>
</tr>
<tr>
<td>$\tau^{\delta}(x, q(x), \dot{q}(x), t)$</td>
<td>Kinetic co-energy density</td>
</tr>
<tr>
<td>$\psi(x, \dot{q}, \nabla_x \dot{q}, t)$</td>
<td>Potential energy density</td>
</tr>
<tr>
<td>$\phi(x, \dot{q}, \nabla_x \dot{q}, t)$</td>
<td>Content density</td>
</tr>
<tr>
<td>$\alpha(x, t)$</td>
<td>Density of external forces</td>
</tr>
</tbody>
</table>

2.2 Model Equations

Various physical effects existent in a solid material are exploited in microsystem devices. We have to specify which fields capture the behavior of the effect, and according to the Lagrange recipe, we next have to find expressions for energy densities listed in Table 2.1. Then Eqn. (2.1) is applied to obtain the governing partial differential equations, i.e. the equation of motion of the fields. In the following sections, the energy expressions for elastic effects in the solid, for electromagnetic fields and for thermo-mechanical effects are presented.

2.2.1 Elastic Behavior

When a solid is deformed, external or internal forces do work on the body. This work is stored inside the solid and can be retrieved when the body is allowed to go back to its original form. When also the kinetic and dissipative energy densities for an elastically deformed crystal are known, we can find the motion of an elastic
body under given initial conditions and external forces. Table 2.2 shows the energy densities involved in this type of physical phenomenon.

**Table 2.2** Expressions for the energy densities and in an elastodynamic solid and associated symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau^*(x, d(x), d(x)))</td>
<td>Kinetic co-energy of a continuous solid</td>
</tr>
<tr>
<td>(\nu(x, d, \nabla d))</td>
<td>Elastic energy density</td>
</tr>
<tr>
<td>(\phi(x, d, \nabla d))</td>
<td>Internal friction content</td>
</tr>
<tr>
<td>(q(x, t) = d(x, t))</td>
<td>Displacement vector field</td>
</tr>
<tr>
<td>(\rho)</td>
<td>Mass density</td>
</tr>
<tr>
<td>(C = [C_{ijkl}] = [C_{klji}])</td>
<td>4th rank elasticity tensor</td>
</tr>
<tr>
<td>(\eta = [\eta_{ijkl}] = [\eta_{klji}])</td>
<td>4th rank viscosity tensor</td>
</tr>
<tr>
<td>(\sigma(x) = f(x))</td>
<td>Body force density vector field</td>
</tr>
</tbody>
</table>

The field to be solved for is a vector field of the local displacements. To find the equations of motion for this field, we insert the symbols of Table 2.2 into Eqn. (2.1), and leaving out zero terms we obtain:

\[
\frac{d}{dt} \left( \frac{1}{2} \rho |d|^2 \right) + \nabla_x \cdot \left( \nabla_x d \left( \frac{1}{2} (\nabla_x d)^T : C : (\nabla_x d) \right) + \nabla_x \nabla_x d \left( \frac{1}{2} (\nabla_x d)^T : \eta : (\nabla_x d) \right) \right) = f \tag{2.2}
\]

Note that the gradient of a quadratic form of second order tensors is given by

\[
\nabla_x u^T : C : u = u^T : C + C : u \tag{2.3}
\]

Because the elastic constants are symmetric with respect to the exchange of the first and second index pairs, the gradient of elastic energy term in Eqn. (2.2) is
2 Modeling

\[ \nabla_{\xi}d \left( \frac{1}{2}(\nabla_{\xi}d)^T : C : (\nabla_{\xi}d) \right) = \frac{1}{2}(\nabla_{\xi}d)^T : C + (C : \nabla_{\xi}d) = C : \frac{1}{2}(\nabla_{\xi}d)^T + (\nabla_{\xi}d) \]  (2.4)

Usually, the symmetric second rank tensor defined by

\[ \varepsilon = \frac{1}{2}(\nabla_{\xi}d)^T + (\nabla_{\xi}d) \]  (2.5)

is denoted the strain, and the second rank tensor defined by

\[ \sigma = C : \varepsilon - \eta : \varepsilon \]  (2.6)

is denoted the stress. Note that there are two components in the stress: The elastic stress coming from Hooke's law, and a non-elastic contribution that is coupled to the strain-rate \( \dot{\varepsilon} \). Using these definitions, and taking all the possible gradients, Eqn. (2.2) is transformed into

\[ \rho \ddot{d} - (\nabla_{\xi} \cdot \sigma) = f \]  (2.7)

which is the well-known equation of motion of linear elastodynamics [Landau86]. Note that to simplify the exposition, linear materials and small strains have been assumed.

**Derivation of the Elastic Energy Densities**

The expression for the stored elastic energy in table 2.2 will now be derived from the theory of ionic crystals. We closely follow the derivation of [Ashcroft76]. It is assumed that the atoms in a crystal oscillate around an equilibrium position, which is located at a Bravais lattice site. Each ion in the crystal can then be identified by its equilibrium lattice site \( R \). Furthermore, only motions that are small compared to the interatomic spacing are considered. At any given time, every ion has a position \( x(R) \). The displacement \( d \) is the difference between the equilibrium position and the momentary position, so that

\[ r(R) = R + d(R) \]  (2.8)
2.2 Model Equations

It is also assumed that the ions in the lattice interact through a pair potential \( \phi \), whose value only depends on the difference of two ion locations. The potential energy in the crystal is then given by

\[
V = \frac{1}{2} \sum_{R, R'} \phi(r(R) - r(R')) = \frac{1}{2} \sum_{R, R'} \phi(R - R' + d(R) - d(R'))
\]  \hspace{1cm} (2.9)

For a general pair potential \( \phi \), the extraction of information from Eqn. (2.9) is extremely difficult, so further simplification is required. We introduce the harmonic approximation, which consists in expanding the potential energy Eqn. (2.9) in a Taylor series around \( R - R' \):

\[
\frac{1}{2} \sum_{R, R'} \phi(R - R' + d(R) - d(R')) = \frac{N}{2} \sum_{R} \phi(R) + \frac{1}{2} \sum_{R, R'} (d(R) - d(R')) \cdot \nabla \phi(R - R') + \frac{1}{4} \sum_{R, R'} (d(R) - d(R'))^T \cdot (\nabla \nabla \phi(R - R')) \cdot (d(R) - d(R')) + O(d^3)
\]  \hspace{1cm} (2.10)

The linear part in Eqn. (2.10) is the force that is exerted on each atom when it occupies its equilibrium position. This term vanishes, because at equilibrium, there is no force. The next nonzero term is the quadratic term, which is the only one that is kept in the so-called harmonic approximation. The potential energy then becomes

\[
V = V_{eq} + V_{harm}
\]  \hspace{1cm} (2.11)

where \( V_{eq} \) is the equilibrium potential energy, and the harmonic contribution is

\[
V_{harm} = \frac{1}{4} \sum_{R, R'} (d(R) - d(R')) \cdot (\nabla \nabla \phi(R - R')) \cdot (d(R) - d(R'))
\]  \hspace{1cm} (2.12)

The zero point of the energy can be chosen arbitrarily, so the constant term can safely be ignored in dynamic problems. We now want to pass to the continuum description commonly used in elastodynamics. We assume that the displacement field \( d \) varies only slowly compared to the interatomic spacing. It can then be considered a continuous function, and can also be expanded in a Taylor series:
When assuming that the pair potential $\phi$ decays rapidly enough, that is only ions close to each other contribute considerably to the interaction energy, Eqn. (2.13) can be inserted into Eqn. (2.12), and the harmonic energy $V_{harm}$ becomes

$$V_{harm} = \frac{1}{4} \sum_{R, R'} \left[ (R' - R) \cdot \nabla d(R) \right] \cdot \left[ \nabla \nabla \phi(R - R') \right] \cdot \left[ (R' - R) \cdot \nabla d(R) \right]$$

(2.14)

From this we see that the harmonic potential energy only depends on $\nabla \phi$. This is consistent with the fact that a constant displacement field, corresponding to a rigid translation of the solid, does not change the internal energy of the solid. Defining the tensor of fourth rank in square brackets in Eqn. (2.14) as

$$\bar{C}(R) = \frac{1}{2} \sum_{R'} ((R' - R) \nabla) \cdot \left[ \nabla \nabla \phi(R - R') \right] (R' - R)$$

(2.15)

the harmonic potential energy becomes

$$V_{harm} = \frac{1}{2} \sum_{R} \nabla d(R)^T \cdot \bar{C}(R) : \nabla d(R)$$

(2.16)

Because the displacement $d$ is assumed to be slowly varying, the sum in Eqn. (2.16) can be transformed into an integral. If one also assumes a homogenous solid, the tensor $\bar{C}$ can not depend on $R$ because of translation symmetry. Therefore, Eqn. (2.16) becomes

$$V_{harm} = \frac{1}{2} \int_{\Omega} \nabla d(x)^T : C : \nabla d(x) d^3 x$$

(2.17)

where the elastic tensor of fourth rank $C$ is given by
and $v_p$ is the volume of the primitive lattice cell. The potential energy density $v$ in a solid is therefore

$$v = \frac{1}{2} \nabla d(x)^T : C : \nabla d(x)$$

which is consistent with the expression given in Table 2.2. The expression

$$C = \frac{1}{2v_p} \sum_{R} ((R') \nabla')^T (\nabla \phi(R))(R')$$

shows the direct connection between the microscopic pair potential $\phi$ and the macroscopically measurable elastic constants $C$. What is still open is to explore the symmetries of $C$. Different methods are used in the literature to show that in a general material, only 21 of the 81 elastic constants are independent. Refer to Table 2.3 for a summary of the arguments [Landau86] [Sokolnikoff56] [Ashcroft76]. What is common in all these arguments is that there are two main reasons for the fundamental symmetries. One is the fact that rigid body rotations do not cause an actual elastic deformation. From this the symmetry of the strain is deduced. The second argument is that the elastic constants are used in a quadratic form to obtain the energy, which yields more fundamental symmetries.

**Table 2.3** Derivation of symmetries in the Elastic constants

<table>
<thead>
<tr>
<th>Landau</th>
<th>Sokolnikoff</th>
<th>Ashcroft/Mermin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric strain</td>
<td>Symmetric stress</td>
<td>From Eqn. (2.20)</td>
</tr>
<tr>
<td>$\varepsilon_{ij} = \varepsilon_{ji}$</td>
<td>$\sigma_{ij} = c_{ijkl} \varepsilon_{kl}$</td>
<td>$c_{ijkl} = c_{kijl}$; $c_{ijkl} = c_{ilkj}$</td>
</tr>
<tr>
<td>$\Rightarrow c_{ijkl} = c_{jikl} = c_{jikl} = c_{klij}$</td>
<td>$\Rightarrow c_{ijkl} = c_{jikl}$</td>
<td>$\Rightarrow c_{ijkl} = c_{klij}$</td>
</tr>
<tr>
<td>Symmetric strain</td>
<td>$\Rightarrow c_{ijkl} = c_{ijlk}$</td>
<td>36 independent constants</td>
</tr>
<tr>
<td>$\Rightarrow 36$ independent constants</td>
<td>$\Rightarrow 36$ independent constants</td>
<td>36 independent constants</td>
</tr>
</tbody>
</table>
Modeling

Energy: \( e = \frac{1}{2} c_{ijkl} \varepsilon_{ij} \varepsilon_{kl} \)

If crystal symmetries are taken into account, even more dependencies among the elastic constants are found [Nye98]. In the extreme case of isotropic materials, two numbers alone suffice to describe the elastic behavior of the material.

### 2.2.2 Heat Flow and Non-linear Thermo-Mechanics

The theory of elastic behavior of solids is extended by also taking into account the coupling between thermal phenomena and mechanics. Many solids show the phenomenon that when their temperature is increased, they expand. On the other hand, compression work done on a solid is partially stored in the form of unordered internal energy, causing the temperature to increase. This indicates that a two-way coupling between the temperature field in a body and the mechanical motion exists. We want to find expressions for this coupling, and at the same time also consider non-linear elastic terms.

#### Table 2.4 Energy densities for thermo-mechanics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q(x, t) = {0, d} )</td>
<td>Generalized coordinate field</td>
</tr>
</tbody>
</table>
### 2.2 Model Equations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta(t) = \int_{t_0}^{t} T(t')dt'$</td>
<td>Primitive function of the Temperature, so that $T = \theta$</td>
</tr>
<tr>
<td>$\tau^\psi(x, d) = \frac{1}{2} \rho</td>
<td>\dot{d}</td>
</tr>
<tr>
<td>$\nu(x, \dot{\theta}, \nabla_x d) = F(T, \varepsilon)$</td>
<td>Density of the Helmholtz free energy</td>
</tr>
<tr>
<td>$F(T, \varepsilon) = \frac{1}{2} \varepsilon : C : \varepsilon - \varepsilon : C : \alpha T$</td>
<td>Free energy as a function of strain and temperature</td>
</tr>
<tr>
<td>$\varepsilon = \frac{1}{2} \left( (\nabla_x d)^T + \nabla_x d + \nabla_x d (\nabla_x d)^T \right)$</td>
<td>Non-linear strain</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Thermal expansion coefficient tensor</td>
</tr>
<tr>
<td>$T$</td>
<td>Temperature offset to the material's stress-free temperature</td>
</tr>
<tr>
<td>$\Lambda = \tau^\psi - \nu$</td>
<td>Lagrange density</td>
</tr>
</tbody>
</table>

We insert the symbols of Table 2.4 into Eqn. (2.1), and look at all the terms separately. The first term is

$$\frac{d}{dt} \nabla_{(\dot{\theta}, d)} \Lambda = \frac{d}{dt} \left[ \frac{\partial}{\partial T} F, \rho d \right]$$

(2.21)

We use a thermodynamic formalism to obtain equations for the thermal part of the interaction. To apply thermodynamics, it is important to identify the variables and the equations of state. Classically, thermodynamics is applied to gases but we now apply it to a small, mesoscopic part of the solid. Mesoscopic means in this context that the part of the solid is so small that it is considered infinitely small from a macroscopic point of view, but large enough to neglect the individual atoms, so that the averaging assumptions for thermodynamics and continuum mechanics hold. The state variables of a gas are the volume, the pressure and the temperature. The state of the small part of the solid is not completely described by these variables, especially because the deformation of a solid is not completely described by
2 Modeling

stating a change in volume, but by giving the strain. Refer to Table 2.5 for an overview of the state variables in a solid, compared to a gas.

Table 2.5 State variables of a gas and a solid

<table>
<thead>
<tr>
<th>Gas</th>
<th>Solid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature $T$</td>
<td>Temperature $T$</td>
</tr>
<tr>
<td>Volume $V$</td>
<td>Strain tensor $\varepsilon$</td>
</tr>
<tr>
<td>Pressure $p$</td>
<td>Stress tensor $\sigma$</td>
</tr>
</tbody>
</table>

We now use the Maxwell relation [Callen85]

$$\frac{\partial F}{\partial T} = -s$$

(2.22)

where $s$ is the entropy density, to recast Eqn. (2.21) to

$$\frac{d}{dt} \nabla_{(\theta, d)} \Lambda = \left[ \frac{\partial s}{\partial \theta}, \rho \dot{d} \right]$$

(2.23)

Because the Lagrangian does neither explicitly depend on the temperature integral $\theta$ nor the displacement $d$, the next term in Eqn. (2.1) vanishes

$$-\nabla_{(\theta, d)} \Lambda = 0$$

(2.24)

The following term becomes

$$\nabla_x \cdot (\nabla_{\nabla_x (\theta, d)} (\Lambda)) = \nabla_x \cdot \nabla_{\nabla_x (F(T, \varepsilon))} = \nabla_x \cdot (\nabla_\varepsilon F(T, \varepsilon) \cdot \nabla_{\varepsilon \varepsilon})$$

(2.25)

Through the correspondence between strain and volume and between stress and pressure the stress is defined as

$$\nabla_\varepsilon F(T, \varepsilon) = \sigma$$

(2.26)
2.2 Model Equations

This is consistent with the previous definition Eqn. (2.6), in absence of thermal effects and for small strains. The last term in Eqn. (2.25) is recast as

$$\nabla V_x \cdot \varepsilon = \nabla V_x \left[ \frac{1}{2} \left( \left( \nabla_x d \right)^T + \nabla_x d + \nabla_x d \left( \nabla_x d \right)^T \right) \right] = 1 + \nabla_x d$$

(2.27)

Putting together Eqn. (2.23), Eqn. (2.25) and Eqn. (2.27) leads to a system of partial differential equations

$$\begin{cases} 
\dot{s} = 0 \\
\rho \dot{d} - \nabla_x \cdot (\sigma \cdot (1 + \nabla_x d)) = 0
\end{cases}$$

(2.28)

From the first equation we see that processes are isentropic, that is, the entropy density is constant and conserved. This means that irreversible processes have not been taken into account yet. The most important irreversible process in the solid is heat conduction. Heat conduction produces entropy, and it is modeled by an entropy source density. The entropy generation rate due to the irreversible flux of heat from a hotter part of the solid to a colder part is according to Fourier’s law

$$\dot{s}_{\text{flux}} = \frac{\nabla \cdot (\kappa \nabla T)}{T}$$

(2.29)

so that Eqn. (2.28) becomes

$$\dot{s} = \frac{\nabla \cdot (\kappa \nabla T)}{T}$$

(2.30)

$$\rho \dot{d} - \nabla_x \cdot (\sigma \cdot (1 + \nabla_x d)) = 0$$

The equation for the entropy density can be brought to a more familiar form by again using Maxwell relations. First the heat capacity at fixed strain must be found. It is the derivative of the internal energy density with respect to the temperature. The internal energy density $u$ is given by the Legendre transform of the Helmholtz free energy density with respect to the temperature:

$$u(T, \varepsilon) = F(T, \varepsilon) - T \frac{\partial}{\partial T} F(T, \varepsilon) = F - Ts$$

(2.31)
The heat capacity then becomes

$$c = \frac{\partial u}{\partial T} = \frac{\partial F}{\partial T} - s - T \frac{\partial s}{\partial T} = T \frac{\partial s}{\partial T}$$

(2.32)

By inserting Eqn. (2.32) and the definition of the stress (Eqn. (2.26)) into the total time derivative of the entropy density yields

$$s(\varepsilon, T) = \frac{\partial s}{\partial \varepsilon} \dot{\varepsilon} + \frac{\partial s}{\partial T} \dot{T} = \frac{c}{T} \dot{T} - \frac{\partial F}{\partial \varepsilon} \frac{\partial \dot{\varepsilon}}{\partial T} = \dot{\varepsilon} = \frac{c}{T} \dot{T} - \frac{\partial F}{\partial \varepsilon} \frac{\partial \dot{\varepsilon}}{\partial T}$$

(2.33)

Inserting Eqn. (2.33) into Eqn. (2.30) and multiplying by the temperature brings Eqn. (2.28) to the system of partial differential equations of thermomechanics

$$\begin{cases} 
\rho \ddot{d} - \nabla \cdot (\sigma \cdot (1 + \nabla \cdot d)) = 0 \\
\rho \ddot{\sigma} - T \frac{\partial \sigma}{\partial T} \cdot \dot{\varepsilon} + \nabla \cdot (\kappa \nabla T) = 0 
\end{cases}$$

(2.34)

The lower part of Eqn. (2.34) is the familiar heat conduction equation, with an additional source term for the heat generated by the compression of the material.

**2.2.3 Electrodynamics**

To show the power of the method, the equations of electrodynamics, Maxwell’s equations, are derived using this framework of writing down the Lagrange density and applying Eqn. (2.1). Note that for this model, the concepts of kinetic and potential energy do not apply, and the Lagrangian is not of the form kinetic minus potential energy. Nevertheless, a Lagrangian is found that yields the correct equa-
tions of motion [Jackson75] [Nelson79]. The field of generalized coordinates in this case is the potential 4-vector.

Table 2.6  Lagrange density for the electromagnetic field

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q(x, t) = {\psi(x, t), A(x, t)}$</td>
<td>Potential 4-vector</td>
</tr>
<tr>
<td>$\Lambda_F = \frac{\varepsilon_0}{2} (-\nabla \psi - \dot{A})^2 - c^2 (\nabla \times A)^2$</td>
<td>Free electromagnetic field Lagrange density</td>
</tr>
<tr>
<td>$\Lambda_I = -\rho \psi + JA$</td>
<td>Interaction Lagrange density</td>
</tr>
<tr>
<td>$\Lambda = \Lambda_F + \Lambda_I$</td>
<td>Total lagrange density</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Charge density</td>
</tr>
<tr>
<td>$\varepsilon_0$</td>
<td>Dielectric constant</td>
</tr>
</tbody>
</table>

The electric field $E$ and the magnetic flux density $B$ are obtained from the potential 4-vector $\{\psi, A\}$ through

$$E = -\nabla \psi - \dot{A}$$  \hspace{1cm} (2.35)

$$B = \nabla \times A$$  \hspace{1cm} (2.36)

By this construction the two homogenous Maxwell equations

$$\nabla \cdot B = 0$$  \hspace{1cm} (2.37)

$$\nabla \times E + \dot{B} = 0$$  \hspace{1cm} (2.38)

are automatically satisfied. In the present case, since dissipative effects are neglected, many terms in Eqn. (2.1) vanish. Expanding all the terms separately

$$\frac{d}{dt} \nabla \psi \cdot (\psi, A) \cdot \Lambda = \frac{d}{dt} \left[ \varepsilon_0 \left( \dot{A} + \nabla \psi \right) \right]$$  \hspace{1cm} (2.39)
Combining Eqn. (2.39), Eqn. (2.40) and Eqn. (2.42) yields

\[
\begin{bmatrix}
-\rho + \nabla_x \cdot \varepsilon_0 (\nabla \Psi + \dot{A}) \\
\varepsilon_0 \frac{d}{dt} (\dot{A} + \nabla \Psi) - \dot{J} + \varepsilon_0 c^2 \nabla \times \nabla \times A
\end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\]  

(2.43)

Inserting the ancillary conditions, Eqn. (2.35) and Eqn. (2.36), and noting that

\[
\varepsilon_0 c^2 = \frac{1}{\mu_0}
\]

(2.44)

brings Eqn. (2.43) to the standard form of the two inhomogenous Maxwell equations:

\[
\begin{bmatrix}
-\rho + \varepsilon_0 \nabla \cdot E = 0 \\
\varepsilon_0 \dot{E} - \dot{J} + \frac{1}{\mu_0} \nabla \times B = 0
\end{bmatrix}
\]  

(2.45)

This subject can be pursued further to even yield a complete theory of the coupling of mechanics, electrodynamics and thermodynamics [Nelson79] [Fowler97], describing piezoelectric, optical and electro-acoustic effects.

(1) For a full derivation of Eqn. (2.41), see Appendix A.1 "The Rotational Square Gradient Identity".
2.3 Discretization and Lagrangian Dynamics

To find analytical solutions for the equations of motion derived in the previous sections is only possible for heavily simplified device geometries. To obtain solutions for realistic device geometries, it is inevitable to resort to numerical computations. To this end, the equations of the models have to be discretized, that is, approximated by schemes with a finite number of degrees of freedom. Equations of motion for discrete systems are ordinary differential equations (ODEs).

We use the finite element method to discretize the equations of motion. It turns out that the finite element discretization can be found directly from the Lagrangian formulation as follows. The Lagrangian of the whole system is obtained by integrating the energy densities over the volume. So the Lagrangian $L$ is given by

$$ L = \int_{\Omega} \Lambda(x, t) d^n x $$

(2.46)

The dissipative content function $D$ is
D = \left[ \int_\Omega \phi(x, t) d^n x \right] (2.47)

The forcing function \( Q \) is obtained from the gradient of the virtual work of the external forces with respect to the virtual displacements:

\[ Q = \nabla_{\delta q} \delta W \] (2.48)

The generalized displacement field \( q \) is discretized in space, that is, it is approximated using a finite dimensional function space. This transforms the continuous system, having an infinite number of degrees of freedom, into a discrete system with a finite number of degrees of freedom.

2.3.1 Discretized Elastodynamics

Looking first at the elastodynamic case, the displacement field \( d \) is expanded as

\[ d(x) = \sum_i d_i N_i(x) \] (2.49)

where the \( N_i \) are the spatial basis functions and the \( d_i \) are the numerical weights. Inserting Eqn. (2.49) into the expressions from table 2.2 for the kinetic co-energy \( T^* \) yields

\[ T^* = \frac{1}{2} \int_V \rho |d|^2 d^n x = \frac{1}{2} \rho \left| \sum_i d_i N_i(x) \right|^2 d^n x \]

\[ = \frac{1}{2} \rho \sum_i d_i N_i(x) \cdot \sum_j d_j N_j(x) d^j x = \frac{1}{2} \sum_{i,j} d_i d_j \int_V N_i \cdot N_j d^n x = \frac{1}{2} \sum_{i,j} d_i d_j M_{ij} \] (2.50)

The potential energy \( V \) is similarly discretized

\[ V = \int_V \nu d^n x = \frac{1}{2} \sum_{i,j} \left( \int_V d_i \nabla_x N_j \right)^T : \left( \int_V d_j \nabla_x N_i \right) d^n x \]

\[ = \frac{1}{2} \sum_{i,j} d_i d_j \int_V (\nabla_x N_i)^T : (\nabla_x N_j) d^n x = \frac{1}{2} \sum_{i,j} d_i d_j K_{ij} \] (2.51)

So is the dissipative content function \( D \)
2.3 Discretization and Lagrangian Dynamics

\[
D = \int_{\Omega} \phi(x, t) d^n x = \int_{\Omega} \left( \sum_j \tilde{d}_j \nabla_x N_j(x) \right)^T : \eta : \left( \sum_j \tilde{d}_j \nabla_x N_j(x) \right) d^n x
\]
\[
= \frac{1}{2} \sum_{i,j} \tilde{d}_i \tilde{d}_j \int (\nabla_x N_i) \nabla_x N_j \, d^n x = \frac{1}{2} \sum_{i,j} \tilde{d}_i \tilde{d}_j \Delta_{ij} \quad (2.52)
\]

The external force vector \( Q \) is obtained by considering the virtual work done by the external body forces:

\[
\delta W = \int_{\Omega} \delta d(x) a(x) d^n x = \int_{\Omega} \sum_i \delta d_i N_i(x) a(x) d^n x = \int_{\Omega} \sum_i \delta d_i N_i(x) a(x) d^n x \quad (2.53)
\]

Using Eqn. (2.48), the i-th component of the generalized force vector \( Q \) is

\[
Q_i = (\nabla_{\delta d} \delta W)_i = \int_{\Omega} \nabla_i N_i(x) a(x) d^n x \quad (2.54)
\]

Equations 2.50, 2.51 and 2.52 implicitly define the mass matrix \( M \), the stiffness matrix \( K \), and the dissipation matrix \( \Delta \). The Lagrangian \( L \)

\[
L = T - V = \frac{1}{2} ((\dot{d}, M \dot{d}) - (d, K d)) \quad (2.55)
\]

together with the dissipative content \( D \)

\[
D = \frac{1}{2} (\ddot{d}, \Delta \dot{d}) \quad (2.56)
\]
yields the equation of motion through the Lagrangian ordinary differential equation

\[
\frac{d}{dt} (\nabla_{\dot{q}} L) - \nabla_{\dot{q}} L + \nabla_{\dot{q}} D = Q \quad (2.57)
\]

When inserting the Lagrangian and the content into Eqn. (2.57), we obtain the expected ODE

\[
M \ddot{d} + \Delta \dot{d} - K d = Q \quad (2.58)
\]
2 Modeling

2.3.2 Discretized Electrodynamics

For electrodynamics, a discretization can also be found immediately by using Eqn. (2.46) and Eqn. (2.57), and defining the following discretization for the potential 4-vector

\[ \{ \Psi, A \} = \left\{ \sum_j p_j N_j(x), \sum_j a_j R_j(x) \right\} \]  

(2.59)

The total electromagnetic Lagrangian is:

\[ L = \int_\Omega \frac{\varepsilon_0}{2} \left[ \left( \nabla \Psi \right)^2 - 2 \nabla \Psi \cdot \dot{A} + \dot{A}^2 - c^2 (\nabla \times A)^2 \right] - \rho \Psi + J A \, d^3 x \]  

(2.60)

We now insert the discretization Eqn. (2.59) into each term of Eqn. (2.60). The first term yields

\[ \int_\Omega \frac{\varepsilon_0}{2} \left( \nabla \Psi \right)^2 \, d^3 x = \int_\Omega \frac{\varepsilon_0}{2} \left( \sum_j p_j \nabla N_j(x) \right) \left( \sum_j p_j \nabla N_j(x) \right) \, d^3 x = \sum_{i,j} p_i p_j \Phi_{ij} \]  

(2.61)

where the electrostatic influence matrix \( \Phi \) is given by

\[ \Phi_{ij} = \int_\Omega \frac{\varepsilon_0}{2} \nabla N_i(x) \cdot \nabla N_j(x) \, d^3 x \]  

(2.62)

The next term gives

\[ \int_\Omega \varepsilon_0 \nabla \Psi \cdot \dot{A} \, d^3 x = \int_\Omega \varepsilon_0 \left( \sum_i p_i \nabla N_i(x) \right) \cdot \left( \sum_j \partial_j R_j(x) \right) \, d^3 x = \sum_{i,j} p_i \partial_j G_{ij} \]  

(2.63)

which implicitly defines the electro-magnetic coupling matrix \( G \). The third term gives
2.3 Discretization and Lagrangian Dynamics

\[ \int_{\Omega} \frac{\varepsilon_0}{2} A^2 \, d^3 x = \int_{\Omega} \frac{\varepsilon_0}{2} \left( \sum_i \dot{a}_i R_i(x) \right) \left( \sum_j \dot{a}_j R_j(x) \right) \, d^3 x = \sum_{i,j} \dot{a}_i \dot{a}_j S_{ij} \]  
(2.64)

which defines the vector potential "mass" matrix \( S \). The fourth term under the integral gives

\[ \int_{\Omega} \frac{\varepsilon_0 c^2}{2} (\nabla \times A)^2 \, d^3 x = \int_{\Omega} \frac{\varepsilon_0 c^2}{2} \left( \nabla \times \sum_i a_i R_i(x) \right) \left( \nabla \times \sum_j a_j R_j(x) \right) \, d^3 x = \sum_{i,j} a_i a_j U_{ij} \]  
(2.65)

which defines the magnetic field energy storage matrix \( U \). The last two terms for the charge density and the current density become

\[ \int_{\Omega} \rho(x) \psi(x) \, d^3 x = \int_{\Omega} \rho(x) \sum_i p_i N_i(x) \, d^3 x = \sum_i p_i \int_{\Omega} \rho(x) N_i(x) \, d^3 x = \sum_i p_i \eta_i \]  
(2.66)

\[ \int_{\Omega} J(x) A(x) \, d^3 x = \int_{\Omega} J(x) \sum_i a_i R_i(x) \, d^3 x = \sum_i a_i \int_{\Omega} J(x) R_i(x) \, d^3 x = \sum_i a_i \gamma_i \]  
(2.67)

Eqn. (2.66) and Eqn. (2.67) define the charge source vector \( \eta \) and the current source vector \( \gamma \). All these equations are converted to scalar products written in index-free notation, and are combined to yield the total Lagrangian \( L \):

\[ L(\{ p, \alpha \}, \{ \dot{p}, \dot{\alpha} \}) = (p, \Phi p) - (p, G \dot{\alpha}) + (\alpha, U \dot{\alpha}) - (\dot{\alpha}, S \dot{\alpha}) - (p, \eta) + (\alpha, \gamma) \]  
(2.68)

The total Lagrangian is now inserted into the Euler-Lagrange equation

\[ \frac{d}{dt} \left( \nabla_{\{ p, \alpha \}} L \right) - \nabla_{\{ p, \alpha \}} L = 0 \]  
(2.69)

to obtain the equations of motion. We first insert Eqn. (2.68) into the first term of Eqn. (2.69)
\[
\begin{align*}
\nabla_{\{\rho, \alpha\}} L &= \nabla_{\{\rho, \alpha\}} \left(- (p, G\alpha) - (\alpha, S\alpha)\right) = \begin{bmatrix} 0 \\ S\alpha - G^T p \end{bmatrix} \tag{2.70}
\n\end{align*}
\]

and then apply the time derivative

\[
\frac{d}{dt}(\nabla_{\{\rho, \alpha\}} L) = \begin{bmatrix} 0 \\ S\alpha - 2G^T p \end{bmatrix} \tag{2.71}
\]

The second term in Eqn. (2.69) becomes

\[
\nabla_{\{\rho, \alpha\}} L = \begin{bmatrix} \Phi p - G\alpha - \eta \\ \gamma - U\alpha \end{bmatrix} \tag{2.72}
\]

Combining Eqn. (2.72) and Eqn. (2.73) to the equations of motion

\[
\begin{bmatrix} -\Phi p + G\alpha \\ S\alpha - G^T \rho + U\alpha \end{bmatrix} = \begin{bmatrix} -\eta \\ \gamma \end{bmatrix} \tag{2.73}
\]

For insight into the behavior of solutions to Eqn. (2.73), consider the case of a cavity inside of which neither current densities nor charge densities are present. For this case

\[
\begin{bmatrix} -\eta \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.74}
\]

If we make the reasonable assumption that the assembled matrix \(\Phi\) is invertible, the top equation in Eqn. (2.73) can be solved for \(p\):

\[
p = \Phi^{-1} G\alpha \tag{2.75}
\]

Inserting Eqn. (2.75) into the bottom equation of Eqn. (2.73) we obtain

\[
(S - G^T \Phi^{-1} G)\alpha + U\alpha = 0 \tag{2.76}
\]
This has the form of an undamped system of harmonic oscillators. If a generalized
eigensolver is applied to Eqn. (2.76), the resonant modes of the cavities can be found.\(^{(2)}\)

\(^{(2)}\) see Section 3.3.3, "Harmonic and Eigenvalue Solutions"
3 Numerical Methods and Discretization Techniques

In chapter 2 we modeled the physical effects in selected microsystems and obtained partial differential equations. Solving these equations analytically is possible only for the most simple of geometries. Through numeric computation, the equations can be solved for realistic microsystem geometries. We will now present the methods for numerical approximation and solution of these equations. The organizational framework is strongly motivated by implementation requirements, the subject of chapter 4.

3.1 Analysis Methods

Figure 3.1 shows a road map of the steps and tools that are required to obtain insight into the complex physical processes exploited in microsystems. The PDEs obtained in chapter 2 are defined for an infinite dimensional space, and are as such not suitable for calculation using a finite sized computer. To overcome this limitation, the equations are discretized, that is, reduced to a finite number of dimensions. Here we consider semi-discretization, which implies that space and time are discretized using two different methods. The spatial discretization, the subject of Section 3.2, converts the PDE into a finite system of ordinary differential equations. Through the use of time-steppers or time-integrators, solutions for the ODE system are directly obtained. This method is computationally expensive, because small time steps are required for reasonable accuracy, and therefore the response of the system has to be evaluated a large number of times. If one is interested in a stationary solution only, describing the state of the system after all disturbances have died out and equilibrium has been reached, the ODE can be con-
3 Numerical Methods and Discretization Techniques

Figure 3.1  Numerical Solution Road Map
3.1 Analysis Methods

verted into a system of algebraic equations \( \Theta \). These equations are in general non-linear. By using a non-linear solving method, such as Newton iteration, a solution for these equations can be found. However, in many cases, the solution of a linearized form \( \Theta \) of the problem is sufficiently accurate. In that case, the stationary state can be found by applying a linear solver to the problem.

Between the time-stepping solution, which captures the full dynamics of the system, and the stationary solutions, there also exist some intermediate forms. One of them is harmonic analysis. This analysis requires the system to be linear. An excitation with a fixed-frequency, i.e. a sinusoidal time-dependent driving force is assumed. Because the system is linear, its response will also be sinusoidal with the same frequency, but each degree of freedom can have a different amplitude and phase relative to the forcing terms. The analysis is performed using complex numbers to describe the amplitudes and phases of the degrees of freedom. For each frequency, the method yields a linear system of complex equations \( \Theta \). The response to an arbitrary time-dependent forcing function can be calculated by decomposing the forcing function into its spectrum through a Fourier analysis.

For the special case of linear mechanical or electrodynamical systems, there exists yet another analysis type that can give insight into the dynamic behavior of the system. Under the assumption that there are no external driving forces and no friction to cause damping, an eigenvalue analysis can be performed. Again assuming sinusoidal motion in time, the ODE is converted into an eigensystem \( \Theta \). The result of solving this eigensystem are the eigenvectors, which represent the shapes of the resonant vibrational or electrodynamical modes, and the eigenvalues, which are related to the resonance frequencies. This analysis requires a numerical eigensolver to obtain the results. It is closely related to harmonic analysis in the following way: When the frequency of the driving force is approaching one of the resonant modes, the system will start oscillating mainly in that mode and therefore assume the shape of this mode. The width of the resonance peak will be determined by damping effects, and for large values of damping, the resonance frequency will even shift away from the eigenvalue. So the eigenvalue analysis will yield the frequencies as an indication of where to look for resonances and will give an estimate for what these resonances will look like. A harmonic analysis in the thus discovered frequency range will provide the response of the system also between the resonant modes, including the effects of damping.
To summarize, to get a full-fledged simulation system we require:

- a discretization method to obtain a set of ordinary differential equations
- time-marching algorithms to step through the time-evolution of the system
- non-linear solvers to find the stationary states of a non-linear system
- linear solvers to solve the linearized systems
- eigensolvers to analyze undamped vibrating structures
- complex linear solvers to obtain the harmonic frequency response

Each of these points will be discussed in the rest of this chapter.

### 3.2 Discretization Methods

Spatial discretization is the method of converting equations defined on an infinite dimensional function space into ordinary differential equations for a finite number of degrees of freedom. This is achieved by two steps. First, a basis for a finite dimensional subspace is picked to approximate the full function space. Then, expressions for the coefficients of the basis functions are derived with the goal to minimize the approximation error. To relax the requirements posed on the continuity of the basis functions, we use the weighted residual method to convert the original partial differential equations into integro-differential equations [Strang73]. We will also show that the discretization scheme using weighted residuals and Galerkin finite elements is consistent with the discretization obtained from the lumping of the Lagrangian described in Section 2.3.

#### 3.2.1 Weighted Residual

The technique of weighted residuals is a tool to derive a whole family of numerical methods for the solution of PDEs. It consists in transforming the equations into integral equations and then choosing appropriate function spaces to find the solutions. In general, a PDE can be written as

\[ (Df)(x, t) - h(x, t) = 0 \quad \forall x \in \Omega, \forall t \]  \hspace{1cm} (3.1)

with the boundary conditions

\[ (Bf)(x, t) - b(x, t) = 0 \quad \forall x \in \partial\Omega, \forall t \]  \hspace{1cm} (3.2)
3.2 Discretization Methods

\( f \) is the function to be solved for, \( D \) is the differential operator, \( h \) is the source function, \( B \) is the boundary operator and \( b \) is the boundary source function. In chapter 2, we obtained a PDE for thermomechanics, Eqn. (2.34). We now consider only the heat transfer part of this and use it as an example on how to obtain a discretization using the method of weighted residuals. For this case, the symbols Eqn. (3.1) and Eqn. (3.2) take the values of Table 3.1.

Table 3.1 Heat conduction symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value for Heat Conduction</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(x, t) )</td>
<td>( T(x, t) )</td>
<td>temperature distribution</td>
</tr>
<tr>
<td>( D )</td>
<td>( \nabla \cdot k(x) \nabla - c(x) \frac{\partial}{\partial t} )</td>
<td>heat conduction equation</td>
</tr>
<tr>
<td>( h )</td>
<td>( h(x, t) )</td>
<td>heat source density function</td>
</tr>
<tr>
<td>( B )</td>
<td></td>
<td>Dirichlet and Neumann operators</td>
</tr>
<tr>
<td>( b )</td>
<td></td>
<td>Dirichlet and Neumann values</td>
</tr>
</tbody>
</table>

The form of the differential equation Eqn. (3.1) and Eqn. (3.2) is the so-called strong form, because the equation has to be satisfied in all points of the domain \( \Omega \). Introducing the scalar product on a function space

\[
\langle f, g \rangle = \int_{\Omega} f(x)g(x) \, d^nx
\]  

(3.3)

and through the use of weighting functions, this is converted to the weak form. We introduce a set of functions \( W \) which are defined on the domain and the boundary, and reformulate the problem to

\[
\langle Df - g, w \rangle = 0 \quad \forall w \in W, \forall t
\]  

(3.4)

\[
\langle Bf - b, w \rangle = 0 \quad \forall w \in W, \forall t
\]  

(3.5)

This is the so-called weak form of the equation. If the weighting functions form a so-called total set [Kreyszig89], the weak and strong forms are equivalent. We now insert the operators and symbols of the heat transfer equation from Table 3.1
into the weak form Eqn. (3.4) and insert the definition of the scalar product Eqn. (3.3):

$$\int_{\Omega} \left( \nabla \cdot \kappa(x) \nabla T(x, t) - h + c(x) \frac{\partial}{\partial t} T(x, t) \right) w(x) \, d^p x = 0 \quad (3.6)$$

Furthermore, we assume that the boundary of the domain is split in two parts,

$$\partial \Omega = \Gamma_N \cup \Gamma_D \cap \Gamma_N \cap \Gamma_D = \emptyset \quad (3.7)$$

where $\Gamma_D$ denotes the Dirichlet boundary, on which the temperature values are prescribed by

$$T(x) = F_D(x) \quad \forall x \in \Gamma_D \quad (3.8)$$

and $\Gamma_N$ the Neumann boundary, on which the heat flux density is prescribed:

$$n(x) \cdot \kappa(x) \nabla T(x) = F_N(x) \quad \forall x \in \Gamma_N \quad (3.9)$$

We may now integrate the first terms in Eqn. (3.6) by parts which yields

$$\int_{\Omega} (\nabla \cdot \kappa \nabla T) w \, d^p x = \int_{\partial \Omega} w n \cdot \kappa \nabla T \, dS(x) - \int_{\Omega} \nabla w \cdot \kappa \nabla T \, d^p x \quad (3.10)$$

As a result, Eqn. (3.4) is now cast into the following form, which is the basis for the finite element discretization:

$$\langle e \frac{\partial T}{\partial t}, w \rangle - \langle h, w \rangle - \langle \kappa \nabla T, \nabla w \rangle + \int_{\partial \Omega} w n \cdot \kappa \nabla T \, dS(x) = 0 \quad (3.11)$$

Note that in Eqn. (3.5), only first derivatives with respect to the coordinates occur. This allows us to relax the conditions on differentiability of the weighting and trial functions.
3.2 Discretization Methods

3.2.2 Finite Elements

The classical Bubnov-Galerkin finite element method starts from Eqn. (3.11). If we restrict the function space we seek the solution from to those which satisfy the Dirichlet conditions Eqn. (3.8). For that case, we can assume that the weighting functions vanish on the Dirichlet boundary:

\[ w(x) = 0 \forall x \in \Gamma_D \quad (3.12) \]

Using this fact and inserting the boundary condition Eqn. (3.9), the boundary integral in Eqn. (3.11) is recast as

\[
\int_{\partial \Omega} wn \cdot \kappa \nabla T \, dS(x) = \int_{\Gamma_N} w(x)n(x) \cdot \kappa(x) \nabla T(x) \, dS(x) = \int_{\Gamma_N} w(x)F_N(x) \, dS(x) \quad (3.13)
\]

We now further limit the space of functions in which the solution is sought to a finite dimensional space. Analogously to the procedure in Section 2.3, we write the temperature field as a superposition of basis functions:

\[ T(x) = \sum_j T_j N_j(x) \quad (3.14) \]

The distinctive feature of the Galerkin finite element method is that for the weighting functions, the same basis functions are used:

\[ w(x) = \sum_j w_j N_j(x) \quad (3.15) \]

Inserting Eqn. (3.13), Eqn. (3.14) and Eqn. (3.15), Eqn. (3.11) is cast into

\[
\langle c \sum_i \bar{T}_i N_i(x), \sum_j w_j N_j \rangle - \langle h, \sum_j w_j N_j \rangle - \langle \kappa \sum_i T_i, \nabla N_i \rangle \sum_j w_j \nabla N_j \rangle
+ \int_{\Gamma_N} \sum_j w_j N_j(x) F_N(x) \, dS(x) = 0 \quad (3.16)
\]

Using linearity of the integration and the scalar products, the finite sums can be pulled out of the integrals, which yields
3 Numerical Methods and Discretization Techniques

\[
\sum_j w_j \tilde{\mathbf{T}}_i \langle \varepsilon N_j, N_j \rangle - \sum_j w_j \langle h, N_j \rangle - \sum_j \sum_i T_i w_j \langle \kappa \nabla N_i, \nabla N_j \rangle + \sum_j w_j \int_{\Gamma_N} N_i(x) F_N(x) dS(x) = 0
\]  

(3.17)

Eqn. (3.17) has to hold for arbitrary weighting functions, that is for any choice of the \( w_j \), so we obtain an equation for every \( i \). We may therefore write Eqn. (3.17) as the vector equation

\[
\mathbf{C} \hat{\mathbf{T}} - \mathbf{s} - \mathbf{L} \hat{\mathbf{T}} + \mathbf{F} = 0
\]  

(3.18)

with the heat conduction matrix \( \mathbf{L} \) given by

\[
L_{ij} = \langle \kappa \nabla N_i, \nabla N_j \rangle = \int_{\Omega} \kappa(x) \nabla N_i(x) \cdot \nabla N_j(x) \, d^\Omega x
\]  

(3.19)

and the heat capacity matrix \( \mathbf{C} \) defined by

\[
C_{ij} = \langle \varepsilon N_i, N_j \rangle = \int_{\Omega} \varepsilon(x) N_i(x) N_j(x) \, d^\Omega x
\]  

(3.20)

The heat source vector \( \mathbf{h} \) is

\[
h_i = \int_{\Omega} h(x) N_i(x) \, d^\Omega x
\]  

(3.21)

and the Neumann boundary contributions \( \mathbf{F} \) are defined as

\[
F_i = \int_{\Gamma_N} N_i(x) F_N(x) dS(x) df
\]  

(3.22)

With this equation, a discretization scheme is obtained yielding the ordinary differential equation Eqn. (3.18), and we arrived at point 2 in Figure 3.1 To obtain a numerical method which can actually be implemented, two components are still missing: The exact definition of suitable basis functions \( N_i \), and techniques to evaluate the integrals which occur in Eqn. (3.19) to Eqn. (3.22). These issues will be discussed in Section 4.3.3.
3.2 Discretization Methods

In Section 2.3, we discussed the derivation of discretization schemes from a lumped Lagrangian formulation. If we compare the heat conduction matrix Eqn. (3.19) with the electrostatic influence matrix \( \Phi \) in Eqn. (2.62), we see that their form is completely equivalent. Electrostatics and heat conduction are both governed by a Poisson-type equation. The Galerkin approach and the derivation using a Lagrangian formulation lead to the same discretization for these equations. It can be shown that if the Galerkin approach is applied to elastodynamics, the same discretization as found from the Lagrangian approach is found.

### 3.2.3 Boundary Elements

The goal of the boundary element method (BEM) [Brebbia84] is to save computational resources by converting the equations into a form that only contains boundary terms. It is a highly specialized technique applied to electrostatic, thermostatic and potential flow simulations. The method is best applied when the ratio of volume the area of the material interfaces is large, which is the case when fields are allowed to extend into infinite free space. Because only terms on the boundary are involved, only the boundary but not the volume needs to be discretized. This can not only be a large efficiency gain, but also increase the flexibility in the presence of moving electrostatic boundaries, as they often occur in electromechanical MEMS devices.

The first step to obtain the method is integrating the third term in Eqn. (3.11) by parts:

\[
\langle \kappa \nabla T, \nabla w \rangle = \langle \nabla T, \kappa \nabla w \rangle = \int_{\partial \Omega} T n \cdot \kappa \nabla w \, dS(x) + \int_{\Omega} \nabla \cdot \kappa \nabla w \, d^v x \quad (3.23)
\]

Inserting Eqn. (3.23) into Eqn. (3.11) yields the so-called inverse statement:

\[
\langle c \frac{\partial T}{\partial t}, w \rangle - \langle h, w \rangle - \langle T, \nabla \cdot \kappa(x) \nabla w \rangle - \int_{\partial \Omega} T n \cdot \kappa \nabla w \, dS(x) + \int_{\partial \Omega} w n \cdot \kappa \nabla T \, dS(x) \quad (3.24)
\]

For ease of explanation, we assume that no heat sources are present inside the domain, and that we only seek the stationary solution:
3 Numerical Methods and Discretization Techniques

\[ \frac{\partial T}{\partial t} = 0, \quad \kappa = 0 \]  
(3.25)

To simplify the notation, we assume that the thermal conductivity is unity:

\[ \kappa = 1 \]  
(3.26)

We now choose as weighting functions the fundamental solution of the Laplace operator, which is defined as the solution to

\[ \nabla \cdot \nabla w(x) = -\delta(x - x_0) \]  
(3.27)

where \( x_0 \) is an arbitrary point in the simulation domain, and \( \delta \) denotes Dirac’s delta function. In 3D, the fundamental solution is given by:

\[ w(x, x_0) = \frac{1}{|x - x_0|} \]  
(3.28)

If we insert Eqn. (3.27) into the third term of the inverse statement Eqn. (3.24), we obtain by the definition of Dirac’s delta function

\[ \langle T, \nabla \cdot \nabla w \rangle = \langle T, -\delta(x - x_0) \rangle = -T(x_0) \]  
(3.29)

Eqn. (3.24) is then becomes

\[ T(x_0) - \int_{\partial \Omega} T(x) n \cdot \nabla w(x, x_0) dS(x) + \int_{\partial \Omega} w(x, x_0) q(x) dS(x) = 0 \]  
(3.30)

where we introduced the surface heat flux density

\[ q_\perp = n \cdot \nabla T \]  
(3.31)

Eqn. (3.30) states that for every point inside the domain, the temperature is determined by the two boundary integrals, if the temperature and the heat flux on the boundary is known. This only holds if \( x_0 \) is inside the domain. If is on the boundary, a correction factor \( b \) has to be introduced to account for the jump of the integral across the boundary:

\[ 40 \]
3.2 Discretization Methods

\[ b(x_0)T(x_0) = \int_{\partial\Omega} T(x)n \cdot \nabla w(x, x_0) dS(x) - \int_{\partial\Omega} w(x, x_0)q_{\perp}(x) dS(x) \]  

(3.32)

We now introduce the discretization. Note that, since only boundary integrals are involved, the basis functions \( N_i \) need now only be defined on the boundary of the domain, and not in the interior as for the finite element method. We use the ansatz for the temperature and the surface flux density:

\[ T(x) = \sum_i T_i N_i(x) \]  

(3.33)

\[ q_{\perp}(x) = \sum_i q_i N_i(x) \]  

(3.34)

Inserting Eqn. (3.33) and Eqn. (3.34) into Eqn. (3.32) yields:

\[ b(x_j) \sum_i T_i N_i(x_j) = \int_{\partial\Omega} \left( \sum_i T_i N_i(x) \right) n \cdot \nabla w(x, x_j) dS(x) - \int_{\partial\Omega} w(x, x_j) \left( \sum_i q_i N_i(x) \right) dS(x) \]  

(3.35)

We may now pick a suitable set of the so-called collocation nodes \( x_j \), and make a one-to-one association between basis functions and these collocation nodes. Furthermore, we have the freedom to pick \( N_i \) and \( x_j \) such that

\[ N_i(x_j) = \delta_{ij} \]  

(3.36)

holds. Then, Eqn. (3.32) is recast as a vector equation of the form

\[ Gq = HT \]  

(3.37)

It can be shown that for piecewise smooth boundaries, the correction factor \( b = 1/2 \), so that the matrix \( H \) is given by

\[ H_{ij} = \int_{\partial\Omega} N_i(x)n \cdot \nabla w(x, x_j) dS(x) - \frac{1}{2} \delta_{ij} \]  

(3.38)
and the matrix $G$ is

$$G_{ij} = \int_{\Omega} w(x, x_j) N_i(x) dS(x) \quad (3.39)$$

With Eqn. (3.37) we have now arrived at point Q in Figure 3.1. We do not go into the details of the evaluation of the integrals in Eqn. (3.38) and Eqn. (3.39) and the choice of the basis functions. For a more detailed treatment, see [Bächtold97].

### 3.3 Treating the Time Dependence: Generating Algebraic Equations

In the previous section, discretization techniques have been discussed. The semi-discretization methods lead to systems of ordinary differential equations (ODEs). We now discuss the numerical methods to obtain solutions to these ODEs.

#### 3.3.1 Non-Linear and Linear Stationary Situation

When interested in the state of the system after a very long time, that is after all initial disturbances have died out, one seeks a stationary solution. Looking at the equation of motion

$$\dot{x}(t) = f(x, t) \quad (3.40)$$

we see that to have a stationary solution, the time derivatives have to vanish, i.e.

$$f(x, t) = 0 \quad (3.41)$$

If the forcing function does not explicitly depend on the time, this reduces to the residual function

$$f(x) = 0 \quad (3.42)$$

In general, this equation defines a non-linear root-finding problem, i.e., a set of non-linear equations. The most common algorithm to solve such a set of equations is the Newton-Raphson method. It functions as follows: It is assumed that there
exists an initial guess of the solution named $x^{(k)}$. Further, it is assumed that the residual function can be linearly approximated in the vicinity of the zero one is looking for, that is, the function can be written in a Taylor expansion:

$$f(x^{(k)} + \Delta) = f(x^{(k)}) + J(x^{(k)}) \cdot \Delta + O(\Delta^2)$$  \hspace{1cm} (3.43)

where the Jacobian matrix $J$ is given by

$$J(x^{(k)})_{ij} = \left( \frac{\partial f_i}{\partial x_j} \right)$$  \hspace{1cm} (3.44)

Because it is desired that the function vanishes, the following linear equation for $\Delta$ is obtained:

$$f(x^{(k)} + \Delta) = 0 = f(x^{(k)}) + J(x^{(k)}) \cdot \Delta$$  \hspace{1cm} (3.45)

which, solved for $\Delta$, yields

$$\Delta = -J(x^{(k)})^{-1} f(x^{(k)})$$  \hspace{1cm} (3.46)

So a better approximation for the zero of the residual function can be found through

$$x^{(k+1)} = x^{(k)} + \Delta = x^{(k)} - J(x^{(k)})^{-1} f(x^{(k)})$$  \hspace{1cm} (3.47)

Linear solvers are used to find the solution to Eqn. (3.47). The initial guess is replaced by the new value, and the process is iterated. As soon as the residual $f(x^{(k)})$ is small enough, the equation system is considered solved. Many devices are accurately described by linear theory. In this case, the residual function is linear, of the form

$$f(x) = Jx - f_0$$  \hspace{1cm} (3.48)

The first iteration of the Newton algorithm will immediately lead to convergence, and further iterations need not be performed.
3.3.2 Stability Analysis and Buckling

In microsystems, especially in membrane and beam structures, we often observe bifurcation phenomena [Ziebart99]. In these systems, the so-called prestress plays an important role, because it influences the transversal stiffness of a slender mechanical system. If the prestress is uniform within the system, and the system is linearized, the residual function is written as

\[ f(x) = (K + \lambda P)x - f_0 \]

(3.49)

where \( K \) is the stiffness matrix which does not depend on the scalar prestress \( \lambda \), \( P \) is the stress-stiffening matrix and \( f_0 \) a vector of external forces. For negative, that is compressive prestresses, the overall stiffness matrix \( K + \lambda P \) can become singular, or in other words, the equation

\[ (K + \lambda P)x = 0 \]

(3.50)

has non-trivial solutions \( x \). These solutions are the eigenvectors of the matrix \( P^{-1}K \), and the lowest eigenvalue is the critical buckling load \( \lambda \). The eigenvectors are an indication of what the buckled structure will look like. This is due to the fact that the eigenvector does not have any elastic energy associated with it, and therefore, a displacement of this form does not cause a net restoring force. How-
ever, the buckling amplitude cannot be calculated from the linearized system, it is determined by the geometrically nonlinear terms that start to dominate in the buckled state.

### 3.3.3 Harmonic and Eigenvalue Solutions

Departing from a stationary formulation, there exists an intermediate solution method that yields some aspects of the dynamic behavior of the system, without the need to perform time-stepping. To apply harmonic analysis, the system has to be linear, that is, the equation of motion has to be of the form

\[ \dot{x} = f(x, t) = Ax + g(t) \]  

(3.51)

where \( g \) is a forcing function that depends on time alone, and \( A \) is a square numerical matrix. In harmonic analysis, it is assumed that the forcing function varies sinusoidally, that is, it consists in a single frequency component. It is most convenient to express this fact using complex notation. Then, the forcing function has the form

\[ g(t) = g_0 \cdot e^{i\omega t} \]  

(3.52)

where \( i = \sqrt{-1} \), and \( \omega \) is the real-valued angular frequency. Under this assumption, Eqn. (3.51) can be solved explicitly. Inserting the forcing function yields

\[ \dot{x} = Ax + g_0 \cdot e^{i\omega t} \]  

(3.53)

Inserting the ansatz

\[ x(t) = x_0 \cdot e^{i\omega t} \]  

(3.54)

into Eqn. (3.53) yields

\[ x_0 \cdot i\omega e^{i\omega t} = Ax_0 \cdot e^{i\omega t} + g_0 \cdot e^{i\omega t} \]  

(3.55)

Because Eqn. (3.55) has to hold at all times, and the factor \( e^{i\omega t} \) never vanishes, the factor can be removed and the linear complex equation
(3.56)

\[(A - i\omega I)x_\omega = -g_\omega\]

is obtained. When solved for \(x_\omega\) we obtain the field amplitude and phase of every
degree of freedom of the system at the given frequency, so that frequency sweeps
can be performed. The whole procedure can be viewed as a Fourier-transfor-
mation of the original system. In the circuit simulation terminology this kind of anal-
ysis is called “AC analysis”, because it describes the response of a circuit when
driven with an AC signal.

**EIGENMODE ANALYSIS**

From the study of mechanical systems, a related kind of analysis is obtained: The
analysis of free vibrations. Mechanical systems have the property that if they are
not damped, they can have solutions that oscillate for an infinite time, even though
the forcing function \(g\) vanishes. This is due to the fact that in an undamped
mechanical system, energy is conserved, and it is transferred back and forth
between potential and kinetic energy. In linear systems with \(n\) degrees of freedom,
there exist \(n\) independent modes of free vibration, and each can have a different
vibration frequency. To find the modes, the forcing function is set to zero, so that
the Fourier-transformed equation Eqn. (3.56) becomes

\[(A - i\omega I)x_\omega = 0 \quad (3.57)\]

Eqn. (3.57) has a non-trivial solution only if \(x_\omega\) is an eigenvector of the matrix \(A\)
for the eigenvalue \(i\omega\). If the matrix is totally anti-hermitian, that is, when

\[(Ax, y) = -(x, Ay) \forall x, y \quad (3.58)\]

holds, where \((, )\) denotes a hermitian scalar product, the eigenvalues are all
purely imaginary, and can therefore be written as \(i\omega\) with a real angular frequency
\(\omega\). In this case, the solution is purely oscillatory, and no damping is present.
3.4 Time Marching Algorithms

When studying transient phenomena in microsystems, the full time evolution of the system must be calculated. This is done using time-steppers. To solve the time-stepping problem

$$\dot{x}(t) = f(x, t)$$

(initial conditions are required, that is, the initial state $x(t_0)$ of the system must be given. Such a problem is called an initial-value problem. We require the motion of the system up to a time $t_f$. There is an extremely large zoo of methods described in the literature for solving ordinary differential equations, and finding better algorithms is still a concerted field of research. In this project, we focused on the implementation of a powerful simulation tool, but not on the development of new time stepping algorithms. Therefore, we have made a choice among the methods, and have designed the software in such a way that new developments in the field or special modeling requirements are easily be supported. We follow the same path in the exposition of the theory: In the sections that follow we describe those algorithms that we have actually implemented and used. These are:

- Forward Euler (Section 3.4.1)
- Fourth order embedded Runge-Kutta with error control (Section 3.4.1)
- Backward Euler (Section 3.4.3)
- Semi-implicit stiff Bulirsch-Stoer extrapolation stepper (Section 3.4.4)

3.4.1 Explicit time marching methods

To find a numerical solution to Eqn. (3.59), we replace its differential quotients by finite differences, and take only a finite number of time steps. The most straightforward algorithm to obtain such a solution is the Euler time stepper. It works as follows: As one has arrived at the i-th time step, at the time $t_i$, the system occupies the state $x(t_i)$. One finds the state of the system at a subsequent time $t_i + \Delta$ by assuming linear motion in the interval:

$$x(t_i + \Delta) = x(t_i) + \Delta \cdot f(x(t_i), t_i) + O(\Delta^2)$$

(3.60)
We see from Eqn. (3.60) that for the Euler method, we have to calculate the right-hand side function \( f \) once for every time step. The error that is introduced is only one order smaller than the stepsize, as we see from the Taylor expansion. The Euler method is therefore said to be a first-order method. By using more function evaluations at suitable points inside the current step interval and using well-chosen linear combinations of the resulting terms, lower-order error terms cancel, and higher order methods are obtained. These methods are called Runge-Kutta methods, and are formally written as [Iserles97]

\[
\xi_{k+1} = x(t_k) + \Delta \sum_{j=1}^{n-1} A_{kj} f(\xi_j, t_j + c_j \Delta)
\]

\[
x(t_{k+1}) = x(t_k + \Delta) = x(t_k) + \Delta \sum_{q=1}^{n} b_q f(\xi_q, t_q + c_q \Delta)
\]

where \( A \) is the RK matrix, the vector \( c \) are the RK nodes and \( b \) denotes the RK weights. If the RK matrix is lower triangular, that is, all its entries above the diagonal are zero, the method is an explicit method, and the \( \xi_k \) depend on previously evaluated quantities alone. Otherwise, the method is implicit and a set of equations, which are in general non-linear because they involve the possibly non-linear function \( f \), must be solved at each step. Finding RK matrices describing efficient methods is beyond the scope of this thesis. Refer to [Iserles97] for an introduction.

One of the most commonly used RK methods is the explicit fourth order method shown in Table 3.2. It only requires four function evaluations per step. If the response function \( f \) is smooth enough, we are able to use stepsizes that are more than four times larger than we would be allowed with the Euler method, and in this case the method is the more efficient one.

Table 3.2 Fourth order Runge-Kutta

\[
A_{kl} = \begin{bmatrix}
\frac{1}{2} & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} \quad c = \begin{bmatrix}
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1
\end{bmatrix} \quad b = \begin{bmatrix}
\frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{bmatrix}
\]
3.4.2 Error Control and Adaptive Time Stepping

When finding the motion of our microsystem, we have to supply the stepper algorithm with a time-step size $\Delta$. We require to obtain a certain accuracy in the solution, and want to choose the stepsize accordingly. However, in most cases it is not possible to find the necessary stepsize a priori, and we cannot tell by looking at the solution what accuracy we have obtained. Furthermore, the system might go through different phases in its time evolution, some of which require very small steps, because fast changes are taking place, and slower phases, where large steps can be taken without losing accuracy. Since the computational cost is inversely proportional to the stepsize, we want the largest stepsize consistent with accuracy requirements at all times. This is also most desirable from the user's point of view, because non-experts in time marching methods should also be able to obtain transient solutions without having to do calculations of stepizes. All these requirements are satisfied by using an adaptive method, where the error is controlled by constantly adjusting the stepsize. The Runge-Kutta (RK) schemes lend themselves to develop such a method. Since a large family of RK methods of different orders exist, it is possible to run two methods of a lower and a higher order in parallel and use the higher order result to estimate the error introduced by the lower order method. The calculation of the error should be of only minimal computational cost. One of the methods that satisfies these requirements is the algorithm invented by Fehlberg and refined by Cash and Karp [Press95]. It combines a fifth
and a fourth order RK stepper that use different linear combinations of the same six function evaluations.

### Table 3.3 Cash-Karp RK Method

<table>
<thead>
<tr>
<th>$A_{kl}$</th>
<th>$c$</th>
<th>$b$</th>
<th>$b^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{5}$</td>
<td>$\frac{3}{40}$</td>
<td>$\frac{3}{10}$</td>
<td>$\frac{11}{54}$</td>
</tr>
<tr>
<td>$0$</td>
<td>$\frac{9}{40}$</td>
<td>$\frac{9}{10}$</td>
<td>$\frac{5}{2}$</td>
</tr>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$\frac{6}{5}$</td>
<td>$\frac{70}{27}$</td>
</tr>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$\frac{35}{27}$</td>
</tr>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td></td>
<td>$0$</td>
<td>$\frac{7}{8}$</td>
<td>$\frac{1}{8}$</td>
</tr>
</tbody>
</table>

Table 3.3 shows the tableau for this method. The vector $b^*$ contains the weights used to obtain the fifth order method, which is used to obtain the error estimate $e$ through

$$e = \Delta \sum_{q=1}^{6} (b_q - b^*_q) f(\xi_q, t_i + c_q\Delta)$$

(3.62)

Common practice not only uses the fifth order terms to estimate the error, but also takes advantage of the higher order to advance the solution. This may not be rigorously justifiable, but in most cases yields acceptable results.

### 3.4.3 Stiff Systems: Implicit and Semi-Implicit methods

Systems of ordinary differential equations can have a property that makes it harder to solve them using finite differencing schemes. ODEs can be stiff, which means that a variety different timescales are involved in the underlying system. Stiff ODEs are not an exotic phenomenon at all; they are frequently encountered in microsystems when modeling circuits, diffusion processes, heat transfer phenomena, and fluid flow. The effect of stiffness is the instability of the explicit solution algorithms. This can cause catastrophic precision loss, which has to be prevented by using extremely small timesteps to assure stability, where with a stable algo-
3.4 Time Marching Algorithms

Larger timesteps would still be consistent with accuracy requirements. To illustrate the situation, we look at a set of linear ODEs with constant coefficients of the form

\[ \dot{x} = -Cx \]  

(3.63)

where \( C \) is a positive definite, symmetric matrix. The matrix \( C \) then has only real, positive eigenvalues, and can be written as:

\[ C = E \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n)E^{-1} \]  

(3.64)

where \( \text{diag}() \) denotes a diagonal matrix with the given values on the diagonal. The analytical solution to Eqn. (3.63) is then

\[ x(t) = E e^{-tD} E^{-1} x(0) \]  

(3.65)

where

\[ e^{\text{diag}(-i\lambda_1, -i\lambda_2, ..., -i\lambda_n)} = \text{diag}(e^{-i\lambda_1}, e^{-i\lambda_2}, ..., e^{-i\lambda_n}) \]  

(3.66)

We see from Eqn. (3.65) that the solution will always tend to zero as time goes to infinity, but the different eigenmodes decay at different rates. If we solve Eqn. (3.63) using explicit forward-Euler differencing, we obtain by inserting into Eqn. (3.60):

\[ x_{i+1} = x_i - \Delta \cdot Cx_i = (1 - \Delta \cdot C)x_i \]  

(3.67)

and therefore, the i-th step is given by

\[ x_i = (1 - \Delta \cdot C)^i x_0 \]  

(3.68)

To reproduce the result that independent of the initial conditions, the solution goes to zero, the following must hold:
Eqn. (3.69) only holds if

\[ \Delta \leq \frac{2}{h_{\text{max}}} \] (3.70)

So for larger stepsizes than required by Eqn. (3.70), there exist initial conditions for which the solution goes to infinity, which is an artificial effect introduced by the time-stepping scheme, and is the reason for instability. Any roundoff error that contributes to a spurious mode that is amplified will lead to an geometrically exploding solution. [Iserles97] [Press95]

The remedy for these problems is to resort to implicit methods. Unfortunately the costs for an implicit method is much larger than for an explicit method, because generally a non-linear equation has to be solved for every time step. To obtain a slightly simpler notation for the sections that follow, we will assume that the function \( f \) does not explicitly depend on time. This is not a serious restriction, because the time can be added as an independent variable:

\[ u = [x, t] \] (3.71)

and the ODE for these new degrees of freedom are

\[ \dot{u} = [f, 1] \] (3.72)

which is equivalent to the original ODE, Eqn. (3.59), but the right-hand side now only depends on independent degrees of freedom.

**THE BACKWARD-EULER METHOD**

To illustrate the fact that a non-linear equation system must be solved for implicit methods, we describe the most straightforward of all implicit methods, the backward-Euler scheme, which is defined by

\[ x_{i+1} = x_i + \Delta \cdot f(x_{i+1}) + O(\Delta^2) \] (3.73)
3.4 Time Marching Algorithms

Note that the new value for the state of the system, \( x_{i+1} \), occurs on both sides of Eqn. (3.73), so that the non-linear equation system

\[
x_i - x_{i+1} + \Delta \cdot f(x_{i+1}) = 0
\]  

(3.74)

has to be solved for every step. This is usually done using Newton’s method, described in Section 3.3.1.

If we go back to the linear ODE example of Eqn. (3.63) and now apply implicit differencing, we obtain by inserting Eqn. (3.63) into Eqn. (3.73)

\[
x_{i+1} = x_i - \Delta \cdot C x_{i+1}
\]  

(3.75)

In this linear case, we can solve for \( x_{i+1} \) and obtain

\[
x_{i+1} = (1 + \Delta \cdot C)^{-1} x_i
\]  

(3.76)

The i-th step is then given by

\[
x_i = (1 + \Delta \cdot C)^{-i} x_0
\]  

(3.77)

In this case, even as the stepsize \( \Delta \) goes to infinity, the solution correctly tends to zero. Of course the solution at earlier times will not be accurate for large stepsizes, but there are no spurious artificially generated solutions, and the method is stable.

Figure 3.3 shows a graphical comparison between the explicit forward-Euler and the implicit backward-Euler time stepping schemes.

**THE SEMI-IMPLICIT EULER METHOD**

Solving Eqn. (3.74) using a full Newton iteration cycle is not always necessary. Because of the special form of Eqn. (3.74), the contribution of the nonlinear terms in \( f \) become smaller for smaller timesteps \( \Delta \). So one can develop \( f \) in Eqn. (3.74) around the starting value \( x_i \), and obtain, when dropping terms above linear order,

\[
x_{i+1} = x_i + \Delta \cdot \left[ f(x_i) + \nabla_x f(x_i) \cdot (x_{i+1} - x_i) \right]
\]  

(3.78)
3 Numerical Methods and Discretization Techniques

Eqn. (3.78) is a linear equation for $y_{i+1}$, which has the solution

$$y_{i+1} = y_i + \frac{1}{2} \Delta \left[ V_y f(x_i) \right]^{-1} \Delta \cdot f(x_i)$$

(3.79)

Eqn. (3.79) defines the semi-implicit Euler time stepping algorithm. Unlike fully implicit steppers, algorithms using Eqn. (3.79) are not unconditionally stable, but are stable for timesteps that are small enough. But the work per timestep is greatly reduced.

### 3.4.4 Semi-Implicit extrapolation method

Higher order methods allowing larger steps can also be obtained for stiff systems, for example using a semi-implicit extrapolation scheme invented by Bader and Deuflhard [Bader83] [Press95]. It is based on the implicit midpoint rule, which is given by

$$x_{i+1} - x_i = 2 \Delta \cdot f \left( \frac{x_{i-1} + x_{i+1}}{2} \right)$$

(3.80)
3.4 Time Marching Algorithms

We develop the function $f$, this time around the midpoint $x_i$, and obtain:

$$f\left(\frac{x_{i-1} + x_{i+1}}{2}, t_i\right) = f(x_i, t_i) + \nabla_x f(x_i, t_i) \cdot \left(\frac{x_{i-1} + x_{i+1}}{2} - x_i\right) + \ldots$$
(3.81)

which results in a linear equation for the endpoint value $x_{i+1}$:

$$[1 - (\Delta - \nabla_x f(x_i))]x_{i+1} = [1 + \Delta - \nabla_x f(x_i)]x_{i-1} + 2\Delta \cdot [f(x_i) - \nabla_x f(x_i) \cdot x_i]$$
(3.82)

The power of this method lies in the fact that the rule can be used in an extrapolating time stepper. To this end, a not so small macro-step $H$ is divided into $m$ micro-steps of the size

$$\Lambda = \frac{H}{m}$$
(3.83)

The macro time stepper crosses the large step $H$ as follows. For the first micro-step, the semi-implicit Euler rule Eqn. (3.79) is used, because it does not require the value at a previous position $x_{i-1}$. Then $m-1$ steps of type Eqn. (3.82) are taken. The last micro-step is smoothed using

$$\bar{x}_{n} = x_{n-1} + hf(x_n)$$
(3.84)

and $x_{n}$ is taken to be the result of this macro step. One advantage of the scheme is that by choosing $m$, the actual order of the method can be selected at runtime. No special bookkeeping is required when changing the stepsizes or the order from one step to the other, so that adaptivity and error control are still easily implemented. The full power only comes into play when applying Richardson’s deferred approach to the limit. The state of the system at the end of the macro step $H$ is considered to be a function of the micro stepsizes $\Lambda$:

$$x(t + H) = g(\Lambda)$$
(3.85)

By increasing the number of sub-steps $m$, we obtain a monotonously decreasing sequence of $\Lambda_j$: 
The idea is now to fit a polynomial to the function $g$ using the values obtained from running the time stepper with different numbers of micro-steps. Because the analytical solution to the ODE would be obtained for a stepsize of zero, we extrapolate the function $g$ to zero stepsize using the polynomial fit. If our macro stepsize $H$ is not too large, the extrapolation will converge when increasing $m$, which lets us stop and proceed to the next macro step as soon as the desired extrapolation accuracy for the current macro-step is reached. Figure 3.4 shows how the stepper crosses a macro-step by using an increasing number of micro-steps and extrapolation to zero stepsize.

It does not make sense to increase the number of micro-steps indefinitely, and at some point one has to conclude that the macro stepsize $H$ is too large, and a new attempt has to be made with a smaller macro stepsize. The choice of how to increase $m$ is again not trivial. The literature suggests using the sequence

$$m = \{2, 6, 10, 14, 22, 34, 50, \ldots\} \quad (3.87)$$

and decreasing the macro stepsize if more than 50 micro-steps are necessary.

Also for this time marching algorithm, we want to monitor accuracy and adjust the macro and micro stepsizes to achieve a given accuracy. Because of the complex nature of the time stepper, we again refer to the literature on this topic [Deuflhard83]. It involves constantly calculating the work required at different sub-step numbers, and aims at convergence of the extrapolation at a number of substeps which requires minimal work per time unit traversed.

### 3.4.5 Systems with first and second order time derivatives

The study of ordinary differential equations is simplified by the fact that all higher-order ODEs can be written as systems that only feature first order time derivatives. As seen in Section 2.3.1, mechanical systems are generally described by equations that feature second order time derivatives. In this section we apply the manipulations necessary to obtain first order systems.
3.4 Time Marching Algorithms

![Diagram of time marching algorithms]

Figure 3.4 Extrapolation time-stepper

**STANDARD THERMO-ELASTODYNAMICS**

Standard discretized linear Thermo-Elastodynamics has the following form. The state of the discretized system is described by the vector $\mathbf{y}$ consisting of temperature degrees of freedom $T$ and displacement degrees of freedom $d$:

$$\mathbf{y} = [T, d]$$  \hspace{1cm} (3.88)

The equation of motion features second order time derivatives, and written in block-matrix form is given by

$$
\begin{bmatrix}
0 & 0 \\
0 & M
\end{bmatrix}
\ddot{\mathbf{y}} +
\begin{bmatrix}
C & A \\
0 & D
\end{bmatrix}
\dot{\mathbf{y}} +
\begin{bmatrix}
L & 0 \\
B & K
\end{bmatrix}
\mathbf{y} -
\begin{bmatrix}
s(t) \\
f(t)
\end{bmatrix} = 0
$$  \hspace{1cm} (3.89)

**Symbol Definition**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>Vector of Mechanical Displacements</td>
</tr>
<tr>
<td>$T$</td>
<td>Temperature Vector</td>
</tr>
<tr>
<td>$y$</td>
<td>State Vector $=[T, d]$</td>
</tr>
</tbody>
</table>
We will now rewrite this ODE in a form containing only first-order time derivatives.

**FIRST ORDER THERMO-ELASTO-DYNAMICS**

To obtain only first-order time derivatives, we treat the velocities as separate degrees of freedom. We then obtain a new state vector

\[
x = [T, u, v]
\]

(3.90)

where the velocity degrees of freedom \( v \) are defined by

\[
v = \dot{u}
\]

(3.91)

Inserting Eqn. (3.90) and Eqn. (3.91) into the standard Thermo-Elasto-Dynamic equation Eqn. (3.89), we obtain

\[
\begin{bmatrix}
C & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & M
\end{bmatrix}
\dot{x} +
\begin{bmatrix}
L & 0 & A \\
0 & 0 & -1 \\
B & K & D
\end{bmatrix}
x +
\begin{bmatrix}
-h \\
0 \\
-f
\end{bmatrix} = 0
\]

(3.92)
To get this into the standard form for ODEs

\[ \dot{x} = f(x, t) \]  

the first block-diagonal matrix in Eqn. (3.92) must be inverted, and we obtain

\[ \dot{x} = f(x, t) = \begin{bmatrix} C^{-1}L & 0 & C^{-1}A \\ 0 & 0 & -1 \\ M^{-1}B & M^{-1}K & M^{-1}D \end{bmatrix} x + \begin{bmatrix} C^{-1}h(t) \\ 0 \\ M^{-1}f(t) \end{bmatrix} \]  

Note that for the heat capacitance matrix \( C \) and the mass matrix \( M \), lumping techniques exist to find diagonal approximations which do not considerably change the behavior of the system. Having to invert only diagonal matrices greatly reduces the work required to pass from Eqn. (3.92) to Eqn. (3.94). [Hughes87]

### 3.5 Enforcing Constraints

The discretized system whose motion we seek stems from physical effects that were modeled using continuous partial differential equations, and discretized using the finite element method. The geometrical models have boundaries, and the interaction with the surrounding world at these boundaries has to be modeled as well. If energy is exchanged across the boundary or forces act on the boundary, those are simply translated into modifications of the forcing function of the discrete system. More difficult to handle are those interactions that can be cast in the form of constraint equations. These usually come from specifying the values of fields in certain regions, such as when the device is in contact with a bath of fixed temperature, or through the mechanical connection of a chip to its packaging which constrains the possible motions of the device. Constraints may change the number of discrete degrees of freedom.

Here we present a consistent formalism to treat these constraints. The method decomposes the operation into three steps:
3 Numerical Methods and Discretization Techniques

- Construction of the constraint equations from the physical model
- Calculation of the transformation onto the reduced, unconstrained degrees of freedom
- Assembling a reduced system matrix through the transformation matrix

For flexibility, we have chosen to explicitly build and store the transformation matrix. Other implementations [Bächtold97] [Korvink93] use implicit methods to handle constraints. The general framework presented in this chapter allows a consistent analysis of all these methods.

3.5.1 Origins and Formulation of Constraints

Table 3.4 and Figure 3.5 show some examples of constraint equations that are important when performing microsystem simulation. The most important ones are those which originate in a model for the interaction of the device being simulated with the its surroundings. Additionally, we consider the T-node constraints, which occur if the discretization mesh features so-called green points [Korvink93] so that a discretization node of one element lies on an edge or face of another element.

Table 3.4 Origins and forms of constraints

<table>
<thead>
<tr>
<th>Name</th>
<th>Thermal</th>
<th>Mechanical</th>
<th>Discretized Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dirichlet</td>
<td>Fixed</td>
<td>Clamped</td>
<td>( x_{i_1} = x_{i_2} = \ldots = x_{i_m} = d )</td>
</tr>
<tr>
<td>Temperature</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floating</td>
<td>Constant</td>
<td>Rigid motion in one direction, and prescribed total force in that direction</td>
<td>( x_{i_1} = x_{i_2} = \ldots = x_{i_m} )</td>
</tr>
<tr>
<td>Temperature,</td>
<td>but prescribed total heat flux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T-Node</td>
<td>(2D)</td>
<td></td>
<td>( x_i = 1/2(x_k + x_j) )</td>
</tr>
</tbody>
</table>

The equations of motion may always be written in the form of a first order equation

\[
\ddot{x} = f(x, t) \quad (3.95)
\]
3.5 Enforcing Constraints

\[ g(x, t) = 0 \quad \forall t \]  

where the constraint function \( g \) is an arbitrary vector valued function

\[ g(x, t) = \{g_1(x_1, \ldots, x_n, t), \ldots, g_m(x_1, \ldots, x_n, t)\} \]  

We now restrict ourselves to a special subset of constraint functions, namely, those that are linear and of the form

\[ Bx(t) - d(t) = 0 \]  

where \( B \) denotes the numerical constraint matrix, and \( d(t) \) is an arbitrary vector function of time. We require that \( B \) has maximum rank, and does not depend on the time. Note that the number of rows of the matrix \( B \) must be less than the number of columns, because otherwise the behavior of the system would be completely determined by the constraints alone.

**Figure 3.5** Boundary models and mesh constraints

where \( x \) denotes the vector of the degrees of freedom of the unconstrained system, and \( t \) the time. In general, constraints are cast as an equation of the form
3.5.2 Explicit Enforcing of Linear Constraints

For the case of linear constraints, it is always possible to find a linear transformation of variables for which the constraints are identically satisfied. These new variables, we call them $v$, are the generalized coordinates in the Lagrangian formulation. The vector $v$ has less entries than the original degrees of freedom in $x$. The following ansatz is used for the constrained degrees of freedom $x$:

$$x(t) = T v(t) + c(t)$$  \hspace{1cm} (3.99)

The transformation matrix $T$ and the vector $c$ are now calculated from the constraints described by Eqn. (3.98). To satisfy the constraints identically,

$$B(Tv(t) + c(t)) - d(t) = BTv(t) + Bc(t) - d(t) = 0 \hspace{1cm} \forall t$$  \hspace{1cm} (3.100)

must hold. Because Eqn. (3.100) must be satisfied for any value of $v$, the terms which are independent of $v$

$$Bc(t) - d(t) = 0$$  \hspace{1cm} (3.101)

and the terms involving $v$

$$BTv(t) = 0$$  \hspace{1cm} (3.102)

must vanish separately. We first consider Eqn. (3.101). Because the constraint matrix $B$ has maximum rank but is rectangular, Eqn. (3.101) has many solutions. For ease of computation, we require the following: Take

$$c(t) = B^T q(t)$$  \hspace{1cm} (3.103)

then the equation for $q$ is

$$BB^T q(t) = d(t)$$  \hspace{1cm} (3.104)

Because $B$ has maximum rank, Eqn. (3.104) has one and only one solution. To satisfy Eqn. (3.102), the image of the transformation matrix $T$ must be the kernel of the constraint matrix $B$, that is, the operator equation
must hold, with the transformation matrix $T$ of maximum rank. For the general case, this can be obtained through a partial Gauss-Jordan reduction of the constraint matrix $B$. In the special case that the constraints originate from Dirichlet and floating boundary conditions only, the $T$-Matrix can also be found directly, as will be shown in Section 3.5.6.

### 3.5.3 Solving constrained linear Systems

We now show how to solve a stationary, linear system (case $\bullet$ in Figure 3.1) subject to constraints. The system equation to be solved is

$$Ax = b \quad (3.106)$$

If the system matrix $A$ is obtained from a finite element discretization, it will be singular in the absence of constraints. This is due to the fact that the heat transfer equation and the elastodynamic equation have zero energy solutions, that is, there exist temperature or displacement fields that do not cause a restoring force. They correspond to rigid body motions in a mechanical system, or to fields of constant temperature in a heat conduction simulation. The constraints have to be such that these zero-energy solutions are suppressed. This means that in a mechanical simulation, the constraints have to forbid free translation and rotation of the device, and in an heat flow simulation, the value of the temperature has to be fixed in at least one point. The constraints are now enforced by introducing the transformation matrix $T$ and the constant vector $c$. Inserting Eqn. (3.99) into Eqn. (3.106) yields

$$ATv = b - Ac \quad (3.107)$$

Because the matrix $AT$ is a vertical matrix, Eqn. (3.107) has no solution in general. One can only require a minimal residual, that is, find a solution vector $v$ for which

$$|ATv - f + Ac|_2^2 = \min \quad (3.108)$$
holds. To be a minimum, the gradient of the functional on the left hand side of Eqn. (3.108) with respect to solution vector \( \nu \) must vanish, which yields the following symmetric system:

\[
T^T A^T A \nu - T^T A^T b + T^T A^T c = 0 \quad (3.109)
\]

This is the general case for non-symmetric system matrices \( A \). If the system matrix is symmetric and positive semi-definite, as is the case for systems coming from finite element discretized heat transfer, electrostatics, and mechanical systems, a much simpler expression is obtained. In the symmetric case, consider the quadratic functional

\[
\frac{1}{2} (A x, x) - (x, b) \quad (3.110)
\]

A vanishing gradient of Eqn. (3.110) is exactly equivalent to Eqn. (3.106) for symmetric matrices, so that Eqn. (3.106) is solved if the functional Eqn. (3.110) has a stationary point. Note that for non-symmetric matrices \( A \), the anti-symmetric part of \( A \) would be discarded\(^{(1)}\) in the quadratic term of Eqn. (3.110), therefore finding a stationary point is not equivalent to solving the non-symmetric equation system Eqn. (3.106). Inserting Eqn. (3.99) into Eqn. (3.110) yields

\[
\frac{1}{2} (A (T \nu + c), T \nu + c) - (T \nu + c, b) \quad (3.111)
\]

Requiring the gradient of this function to vanish, the following symmetric linear equation system is obtained:

\[
T^T A T \nu + T^T A c - T^T b = 0 \quad (3.112)
\]

Solving Eqn. (3.112) for \( \nu \), and inserting the solution into Eqn. (3.99) yields an \( x \) which is consistent with the constraints and is "closest" to a solution of the unconstrained problem.

\[\textbf{(1) If } A^T = -A, \text{ then } (x, A x) = (A^T x, x) = (-A x, x) = (x, -A x) = -(x, A x) \Rightarrow (x, A x) = 0\]
3.5 Enforcing Constraints

3.5.4 Assembling the constrained system

In the finite element method, the assembled system matrix $A$ is formally obtained as a sum of element and node contributions. In the FEMEngine, it is assembled term by term, such that

$$A = \sum_i A^{(i)}$$  \hspace{1cm} (3.113)

where

$$A_{lm}^{(i)} = a^{(i)} \delta_{p(i)} \delta_{q(i)m}$$  \hspace{1cm} (3.114)

that is, it is assembled from matrices that have only one non-zero entry at row $p(i)$ and column $q(i)$. The form of the single $i$-th contribution $A^{(i)}$ is shown in

![Figure 3.6 A single contribution $A^{(i)}$ to the total system matrix $A$](image)

Figure 3.6 We now show how to assemble the constrained system Eqn. (3.112). The first term, the effective system

$$A' = T^T AT$$  \hspace{1cm} (3.115)

and the remaining terms, forming the effective right-hand-side...
are to be assembled. First we consider the effective stiffness \( A' \). Using linearity of the operators yields

\[
A' = \sum_i T^T A^{(i)} T = \sum_i A^{(i)}
\]  

(3.117)

The i-th term of this sum, using index-notation, reads

\[
A_{im}^{(i)} = a^{(i)} \sum_{j,k} T_{jT} \delta_{j(i)} \delta_{kq(i)} T_{km}
\]  

(3.118)

which gives, when removing the Kronecker-\( \delta \) symbols:

\[
A_{im}^{(i)} = a^{(i)} T_{p(i))j} T_{q(i)m}
\]  

(3.119)

This is the outer product of the two rows of the transformation matrix \( T \) corresponding to the column and row in the unconstrained stiffness matrix, respectively. The right-hand side is handled by considering

\[
b' = \sum_j T^T b^{(j)} - \sum_i T^T A^{(i)} c
\]  

(3.120)

where \( b^{(j)} \) is the j-th contribution to the right hand side. This partial right-hand side vector only has one non-zero entry. The position of this entry picks a column of \( T^T \), which is a row of \( T \), multiplies it with the constant \( b^{(j)} \) and adds it to the effective right-hand side. The second term in Eqn. (3.120) is treated by looking at the l-th entry of \( b^{(i)} \):

\[
(T^T A^{(i)} c)_l = a^{(i)} \sum_{j,k} T_{jT} \delta_{j(l)} \delta_{kq(l)} c_k = a^{(i)} c_{q(l)} T_{p(l)i}
\]  

(3.121)

The equations states that the current non-zero column of the partial stiffness matrix selects an entry in the constant vector \( c \), and the current unconstrained row selects a row of the T-matrix. This row is scaled with \( a^{(i)} \) and the constant entry \( c_{q(l)} \), and is then subtracted from the effective right hand side.
3.5 Enforcing Constraints

3.5.5 Enforcing Constraints using Lagrange Multipliers

There is an alternative method to enforce the constraints, which does not require finding a transformation matrix and constraint-free coordinates. If we view the problem as finding a stationary point for

\[ f(x) = |Ax - b|^2 = \text{extremum} \] (3.122)

under the constraints

\[ g(x) = Bx - c = 0 \] (3.123)

we can apply the method of Lagrange multipliers [Strang89]. We introduce the vector of multipliers \( \lambda \). The size of the vector \( \lambda \) is equal to the number of rows in \( B \). The Lagrange multiplier rule now requires that to find a stationary point of the function \( f \) under the constraints \( g \), the following gradient has to vanish:

\[ \nabla_{(x, \lambda)}(f(x) + \lambda \cdot g(x)) = 0 \] (3.124)

Inserting Eqn. (3.122) and Eqn. (3.123) into Eqn. (3.124) and evaluating the gradient yields

\[
\begin{bmatrix}
A^T A x - 2 A^T b + B^T \lambda \\
B x - c
\end{bmatrix} = \begin{bmatrix}
0 \\
0
\end{bmatrix}
\] (3.125)

which in block-matrix form is

\[
\begin{bmatrix}
A^T A & B^T \\
B & 0
\end{bmatrix}
\begin{bmatrix}
x \\
\lambda
\end{bmatrix} =
\begin{bmatrix}
2 A^T b \\
c
\end{bmatrix}
\] (3.126)

If the system matrix \( A \) is symmetric, we can again find a stationary point of the functional

\[ f(x) = \frac{1}{2} (Ax, x) - (x, b) \] (3.127)
to obtain simpler system equations. Using the multiplier rule Eqn. (3.124) we obtain a block matrix system of the form

\[
\begin{bmatrix}
A & B^T \\
B & 0
\end{bmatrix}
\begin{bmatrix}
x \\
\lambda
\end{bmatrix} =
\begin{bmatrix}
b \\
c
\end{bmatrix}
\]  

(3.128)

The method of using Lagrange multipliers has the advantage that the step of finding the transformation matrix is eliminated. On the down side, the method introduces additional degrees of freedom, so that the size of the system matrix is increased. However, there is information in the \( \lambda \) part of the solution vector. If we assume that we have a solution to Eqn. (3.128), the term

\[B^T \lambda\]  

(3.129)

can be viewed as a modification of the external forces and sources \( b \) which would have to be applied to make the system satisfy the constraints. So from the \( \lambda \) vector, we can through Eqn. (3.129) obtain the reactive forces that act on the supports, or the heat fluxes that flow into or out of a thermal reservoir.

### 3.5.6 Building the \( T \)-Matrix Directly

For the simple cases of Dirichlet and floating boundary conditions, the transformation matrix \( T \) can be found directly, instead of obtaining it through a gauss-Jordan reduction of the \( B \)-Matrix. Even \( T \)-node constraints, which are written in the form

\[2x_i - x_{u_1} - x_{u_2} = 0\]  

(3.130)

can be handled in this way, if the degrees of freedom \( x_{u_1} \) and \( x_{u_2} \) are unconstrained.

Algorithm 3.1 describes the procedure to directly build the \( T \)-Matrix. In step 2 of the algorithm, the unconstrained degrees of freedom are handled. In step 3, the entries for the floating nodes are built. In step 4, the fully constrained Dirichlet
3.5 Enforcing Constraints

conditions are enforced, and in step 5, the T-nodes are handled. Figure 3.7 shows an example of a T-matrix produced using Algorithm 3.1.

**Algorithm 3.1 Generating the T Matrix**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Initialize current column to 0</td>
</tr>
<tr>
<td>2.</td>
<td>For each degree of freedom DOF</td>
</tr>
<tr>
<td>2.1</td>
<td>If DOF is free</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Put a 1 on DOF's row in current column of the T matrix</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Increment current column by 1</td>
</tr>
<tr>
<td>3.</td>
<td>For each floating region R</td>
</tr>
<tr>
<td>3.1</td>
<td>For each DOF in the region R</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Put 1 on DOF's row in current column of the T matrix</td>
</tr>
<tr>
<td>3.2</td>
<td>Put total force/flux into b-vector of one of the DOFs</td>
</tr>
<tr>
<td>3.3</td>
<td>Increment current column by 1</td>
</tr>
<tr>
<td>4.</td>
<td>For each Dirichlet Region R</td>
</tr>
<tr>
<td>4.1</td>
<td>For each DOF in the region R</td>
</tr>
<tr>
<td>4.2</td>
<td>Put Dirichlet value into vector c</td>
</tr>
<tr>
<td>5.</td>
<td>For each T-node constraint</td>
</tr>
<tr>
<td>5.1</td>
<td>Put 1 on the first free DOF's row in current column of the T matrix</td>
</tr>
<tr>
<td>5.2</td>
<td>Put 1 on the first free DOF's row in (current column + 1) of the T matrix</td>
</tr>
<tr>
<td>5.3</td>
<td>Put 0.5 on the constrained node's row in the current column of the T matrix</td>
</tr>
<tr>
<td>5.4</td>
<td>Put 0.5 on the constrained node's row in (current column + 1) of the T matrix</td>
</tr>
<tr>
<td>5.5</td>
<td>Increment current column by 2</td>
</tr>
</tbody>
</table>
3. Numerical Methods and Discretization Techniques

- **Free**
- **× Dirichlet**
- **nią Floating Region A**
- **○ Master**
- **菱 Slave**

**Figure 3.7** Example of a mesh with constraints and the corresponding T-Matrix produced by Algorithm 3.1
Combining all the various methods and tools to efficiently simulate microsystem components leads to a simulation system with a high complexity. Object-oriented methods are best suited to design and implement complex software systems. We have applied these methods to combine a large set of the numerical techniques known today, and have created a stable, extensible and flexible object-oriented architecture of a microsystem simulator. We now describe the architecture of the simulator, the FEMEngine, and show what tools were used to implement the design in a simulation tool.

4.1 Software Design Methods

Efforts to develop engineering methods for the design of complex software systems started back in the 1960s. One of the most successful methods for the traditional, high-level programming languages of the 1960s and 1970s such as Fortran, Pascal, Cobol and C, is top-down structured design. It is described in [Meyers78], and was based on the work of Wirth [Wirth75]. The method focuses on the main concept in algorithmic languages, the sub-program. The design method consists in decomposing a big, complex task into simpler, smaller tasks, which are in turn broken up into even smaller tasks. The designer recursively descends to tasks that can be implemented in relatively short sub-programs. This method has been very successful, but, as described in [Stein88], does not scale well for large systems, and is therefore not suitable for software systems with the complexity which is common on today’s fast computer hardware.

In recent years, object-oriented methods have become generally accepted in the software industry as the most effective tool in software development. They have
become prevalent with the advent of the newer, object-oriented programming languages such as Java, C++, Object Pascal and Smalltalk.

In an object-oriented method, the system under construction is viewed as a collection of collaborating objects. The objects are taken from the application domain, so in a microsystem simulation tool, examples for objects in the program are the computational mesh, a material, a partial differential equation or a physical field. Each object in the program has certain tasks to perform, which are found directly from its nature. As examples, the mesh has to be able to find out what material is present at a specified point in space, and a material must be able to state the numerical values of its material properties. Each object has local storage for the data required to perform its tasks.

Furthermore, the objects are grouped into classes. A class defines the possible operations that can be performed on an individual object of the class. The individual objects are called instances of the class. As an example, we consider the behavior of all the different material objects. The materials silicon, silicon oxide and polysilicon, are all instances of the class material, whose definition is shown in

<table>
<thead>
<tr>
<th>Material</th>
</tr>
</thead>
<tbody>
<tr>
<td>name:String</td>
</tr>
<tr>
<td>propertyValues:ListOfValues</td>
</tr>
<tr>
<td>getName():String</td>
</tr>
<tr>
<td>PropertyValue(whichProperty):Value</td>
</tr>
</tbody>
</table>

**Figure 4.1** Example material class

Figure 4.1. This example class defines that possible operations on a material are

- obtain numerical property values
- obtain the name of the material

The individual data, like the name and the values of the material properties, are stored inside each instance. So each material object can have its own name and
4.1 Software Design Methods

property values, but the behavior, that is the possible operations, are the same for all the objects of the class material.

Object-Oriented analysis consists in identifying classes and objects that model the desired behavior of the finished system. In the design phase, new classes and objects are invented and added to the application objects. The product of the design is an architecture of the system. The architecture lists the classes in the system, their operations, and the data stored in the instances. This architecture can then be implemented in an object-oriented language such as C++.

The actual development of the software is done in an iterative process. First, a system with a simplified architecture and only the most basic features is implemented and tested. New classes or even requirements are then formulated that have been found from this first product, and the architecture is extended. This process is then iterated until all requirements are met.

A software development method does also involve a notation for documenting the requirements, the classes and objects, and the architecture in the system. A popular method was the one due to Booch. The classical text about this method, [Booch94], also gives an excellent introduction to the history and philosophy of object-oriented methods.

Of course, Booch’s method is not the only one. On the contrary, a plethora of object-oriented methods, each featuring their own concepts, terminology, notational conventions and development processes are documented in the literature (See [Booch94] for a list of methods and their references). The three authors Booch, Rumbaugh and Jacobson, each of which had developed their own method, fortunately joined in 1995 and created a unified method, called the “Unified Modelling Language” or “UML”. It is quickly gaining acceptance, and there is a rich literature about the method, the notation and the associated development process. An excellent short introduction is [Fowler97]. The full documentation of the UML is found in [Booch99] and [Rumbaugh99]. Where appropriate, we adopt the UML notation to document our architecture, as we have already done for the first time in Figure 4.1. An overview of UML notational elements is found in the Appendix A.2.
4.2 Simulator Requirements and Use Cases

At the outset of the development project, the requirements the finished system must meet are defined. These requirements are then refined with a use case analysis, which consists of a list of scenarios where an input for the system is sketched and the desired behavior or output produced by the system is described. The use cases are then analyzed and a design is created that implements the expected behavior.

4.2.1 Requirements

For a useful tool in microsystem development, the following points are required:

- **Physics:**
  - Multiple physical effects and their coupling
  - Flexibility for special equations such as plate and shell theory
- **Algorithms and numerics:**
  - All analysis methods shown in Figure 3.1, in particular harmonic and transient analysis
  - Finite element technology
- **Constitutive equations:**
  - Scalar and tensor-type material properties with input value validation
  - Material properties that are environment-dependent (dependent on local temperature or doping), featuring table or Taylor-series based material dependencies
- **Computational meshes:**
  - Support general meshes to allow the use of external meshers
  - Allow adaptive refinement to be added
- **Programming issues:**
  - Ease of extension in all directions: new physical models, new shape function sets, additional material properties, more complex material models, a variety of linear, non-linear, eigen- and time-stepping solvers

4.2.2 Use Case Analysis

After having stated the requirements, these need to be made more concrete through use case analysis [Rumbaugh99]. A use case is a unit of externally visible functionality of a system. In a use case, external actors act on the software system to perform a certain task. It is focused on the external view, that is, one is not inter-
4.2 Simulator Requirements and Use Cases

ested in how the system completes the task, but in the fact that it completes the
task in a fashion that is consistent with the requirements. Only at a later stage,
when designing the system, one goes about specifying how to actually perform the
task. Use cases are entities which are good candidates for tests of the system: Each
use case defines an external stimulus or input for the software system, and speci-
fies what kind of behavior, result or output is expected. This can immediately be
translated into a test.

In the context of our microsystem simulator, the actor in the use cases is the design
engineer who wants to model a sensor or actuator. Because of this simplicity, we
present the uses cases in list form and do not draw use case diagrams.

**USE CASES FOR THE MICROSYSTEM SIMULATOR**

The following list states the use cases that the simulator was required to support.
They stem from a more detailed analysis of the requirements stated in Section
4.2.1.

- obtain temperature distribution in a 3D model of a thermally driven microsensor, giving
fixed-temperature boundary conditions and distributed heat sources
  - extension: with anisotropic heat conductivities
  - extension: with temperature-dependent heat conductivities
- obtain static deformation and temperature distribution of a thermomechanically driven
chemical microsensor beam
  - extension: obtain harmonic response of the chemical microsensor beam
  - extension: obtain transient response of the chemical microsensor beam
- the deformation of a electrostatically driven RF microswitch. The volumetric mesh is ob-
tained from a commercial mesh generator
- obtain modes of free vibration of an AFM tip
- obtain deformation of a thin rectangular silicon membrane of a pressure sensor under load
- obtain modes of free vibration thin rectangular silicon plate with metal lines on top
- obtain deformation in a piezoelectrically driven viscosity sensor using volumetric ele-
ments
- obtain deformation in a piezoelectrically driven viscosity sensor using plate elements
- obtain deformation of a thermomechanically driven AFM tip using multi-layered plate el-
ements
- obtain derivative fields from primary field distributions (stress, heat flux, electric fields)
• make the simulation results more accurate
• uniformly promote shape functions in the elements from linear to quadratic polynomials
• adaptively refine the order of the shape function and the size of the elements
• display a field distribution (stress, temperature...) on a structure
• display 2d plot of a field (stress, temperature...) along a line in the interior of a structure
• obtain the average temperature in a region of a test structure

4.3 Design of the Simulator

4.3.1 Overall Structure

Figure 4.2 shows the global architecture of the FEMEngine. It shows the main classes that collaborate to perform a simulation. The Mesh class provides the geometrical description of the device under examination. It has a collection of elements. The partial differential equation, represented by the class PDE, is composed of discretized operators. As we have seen in Section 3.2.2, finite element discretization of heat conduction leads to a heat conduction matrix $L$ of the form
4.3 Design of the Simulator

\[ L_{ij} = \langle \nabla^2 N_i, N_j \rangle = \int_{\Omega} \kappa(x) \nabla N_i(x) \cdot \nabla N_j(x) \, d^nx \]  \quad (4.1)

The integral in Eqn. (4.1) is calculated by the discretized operator using collaborating classes described in detail in Section 4.3.2. We have also seen that especially the operator in Eqn. (4.1), a discretized Laplace operator, occurs in various physical models. Having a separate class for each operator makes it possible to reuse the code for similar operators in different PDEs. The operators generate terms, which are stated as a pair of shape functions, a physical field and the order of the time derivative. These terms must then be converted by an assembler into indices of unconstrained degrees of freedom. The mathematical methods to perform this transformation have been described in Section 3.5. The converted terms are then fed into a so-called system, which represents the analysis method. It is responsible of storing the terms according to time order, and call a solver to obtain solutions to the resulting equations. The solutions are then stored in a solution set, which is then written to disk for post-processing.

4.3.2 Differential Equations and Operators

The core of the FEMEngine is the structure of the physical models, or PDEs. The class hierarchy of the PDEs is shown in Figure 4.3. We have implemented a large variety of physical models, but focused on what is required for micro electro-mechanical systems (MEMS) and their packages. The PDEs are composed of discretized operators. An excerpt from the hierarchy of these operators is shown in

\begin{center}
\includegraphics[width=\textwidth]{figure4.3.png}
\end{center}

\textbf{Figure 4.3} Class hierarchy for the partial differential equations
Figure 4.4. We have omitted the more complex operators used in coupling different physical domains. The operators are used to build PDEs as shown in Figure 4.5. Standard heat flux in a solid is composed of a discretized Laplace operator, and a mass-type operator to model the heat capacity. The Laplace operator is also used in electrostatics, so the same class and associated implementation is used there. Elastodynamics is in turn implemented by using the Navier operator class and a mass-type operator, which in this case models the mechanical inertia.

The power of the object-oriented design comes into play when creating coupled equations. Thermomechanical analysis is implemented by first using standard elastodynamics and standard heat flux equation objects. Then, additional operators are included to account for the coupling. The same technique is used for piezoelectricity, which combines electrostatics with elastodynamics.

4.3.3 Elements, Shape Function Sets and Integrators

The operators do their work using the element objects. The class diagram centered around the element is shown in Figure 4.6. An element is a complex object composed of many parts, which collaborate to represent the geometry and calculate the terms for the finite element method. The element has two sets of responsibilities, coming from two separate requirements. First, an element is the building block of the geometry of the device being simulated. This purely geometrical part is delegated to the polytope object, which is a generic geometrical object. Second, the element is used as a finite element, and to support the FE method, it must provide...
4.3 Design of the Simulator

Figure 4.5 PDEs are built by combining operators, shown in the rightmost column of the graph. Operators occurring in different physical models are re-used, as is shown here for the mass and Laplace operators. The "pure" equations are again combined into coupled equations, shown in the leftmost column. For this case, also coupling operators, such as the thermo-mechanical (TM) coupling operator, must be implemented and added.

shape functions, has to be able to integrate functions for the operators, and provide information about material properties.

The integrations themselves are performed using the integrator class, which in turn uses the quadrature in the ElementType class. This class is a "smart enumeration" [Taschini98], and is a combination of a shape and a shape function set. The shape is represented by a smart enumeration object of the class PolytopeType, and the type of shape functions is represented by an object of the class ShapeFunctionSet, another smart enumeration.

A similar splitting of responsibilities exists for the nodes. There are the purely geometrical nodes, which only act as anchors for the elements and store the positions of the node points in 3D space. They are included by the actual finite element nodes, which also include information about the degrees of freedom (DOFs). There may be more computational nodes than geometrical nodes. As an example, consider a triangle with quadratic shape functions: The geometry of the triangle is
Figure 4.6  Mesh, Elements, Nodes and Shape Functions

completely specified by three geometrical nodes, but for quadratic shape functions, additional computational nodes on the centers of the edges are required.

Computational nodes must also provide a way to obtain the values of the physical fields at the nodes, whereas geometrical nodes are not concerned about physical fields.

In Figure 4.7 the collaboration between the classes involved in the calculation of the finite element terms is shown. The responsibilities of the classes are as follows:

PDE  Creates the list of operators, along with the physical fields they operate on and the kind of kernel they have to integrate, to model the physical effects it describes. This class is completely decoupled from shape functions and numerical integration. During assembly, this class loops over all the elements and passes each to all its operators to have them assemble the terms for the element.
4.3 Design of the Simulator

Figure 4.7 Element, quadrature, integrator and operator classes collaborate to calculate the terms to be assembled

Field
A smart enumeration of the physical fields, such as temperature, displacement, or electric potential.

IntegrationKernel
The model-dependent part in the integral in (4.1). There is a big variety of kernels, but most of them are bi-linear forms depending on material properties. They are used to separate the physical models from the details of the integration and evaluation of the shape functions. This class is completely decoupled from the operators and the shape functions, but represents a purely mathematical object that calculates the bilinear forms. As an example, a poisson kernel consists of a second order tensor $\kappa$ and takes as an argument two vectors, $v_1$ and $v_2$, and returns $v_1 \cdot \kappa \cdot v_2$.

Quadrature
Responsible for providing a list of integration points and corresponding weights for numerical quadrature [Press95]. Every type of element has its own quadrature, depending on the polynomial order of the shape functions and the shape of the element. The implementation of the class is able to produce the list of integration points and weights required for a given accuracy. This class is completely decoupled from shape functions, fields and operators.
4 Object-Oriented Analysis and Design

**ElementType**
A smart enumeration describing the type of element. It is a combination of a shape, a shape function set and a corresponding quadrature. As an example, an object of this class is a triangular element with a set of quadratic shape functions, and a quadrature able to exactly integrate second order polynomials on a triangle.

**Integrator**
This class is responsible for calculating the integrals occurring in the operators. It evaluates the shape functions or their gradients at the integration points, which it obtains from the quadrature. It then applies the integration kernel, and forms the weighted sum to obtain the integral. The integrator provides one operation for each of the possible forms of terms, that is combinations of shape functions and gradients, occurring in the physical models. There is one integrator object for every element in the geometry.

**DiscretizedOperator**
Selects which form of term to calculate, that is, which operation of the integrator to call. It constructs the appropriate integration kernels from the material properties and passes these kernels to the integrator. The PDE calls the operation calculating the terms for every element in the geometry.

**ShapeFunctionSet**
This class is able to evaluate the shape functions, their gradients and second-order derivatives. A shape function set is associated with a certain shape, and with a set of nodes. The Integrator class obtains the shape function values from this class.

**Element**
The element is the object specifying one element in the geometry. It provides access to the actual geometrical position of this particular element, to the material objects and the element type.

### 4.3.4 Materials, Structures and Properties

The objects that calculate the terms for the equations of motion require access to material properties. The subsystem performing this task is the material database. The challenge in implementing a flexible material database lies in the fact that it is exposed to the user of the FEMEngine: The user has to be able to specify material properties in a flexible way. The requirements of microsystem simulation lead to the following hierarchy of requirements:

- multi-layered plates require support for multi-material structures
- material properties are scalars or tensors, so one property has many real numbers
- one material has many properties
- a property may be environment-dependent
- the environment-dependency may be given as a table or as coefficients of a polynomial, such as temperature coefficients
4.3 Design of the Simulator

The design for the material database follows this hierarchy. The architecture of the material database we created is shown in Figure 4.8. The responsibilities of each class are:

- **Structure**: A structure is a composite material. Its main application are multi-layered plates, which consist of a sandwich of different materials of different thicknesses. Another application would be materials with multiple phases. For each member material in the structure, there may be one or more attributes describing the role of the member inside the structure, such as the thickness of the layer. This makes it possible to enter the...
properties of a material only once, and to use it in different contexts, for example with different thicknesses. For physical models where the features of the structure class are not needed, as in conventional 3D simulations, the trivial structure consisting of only one material and no attributes, may be used interchangeably with the material itself.

Attribute
Describes the role of the material in the structure, such as its thickness in a layered structure.

Material
The central class in the material database. An instance of this class denotes one particular material. It stores the material's name, and provides access to a collection of the properties that have been entered for this material.

PropertyID
Objects of this class denote the abstract idea of one particular material property, such as the heat conductivity, or Young's modulus. When requesting the value of a material property from a material object, a PropertyID object is passed to state the type of property that is requested. The PropertyID also states the unit of measure of the property, which is checked against user input to ensure consistency.

Property
This class denotes the value of one particular property of a material. The associated PropertyID object states what kind of property this object is. The most important operation is to request the value of the property from the object. Material properties are tensors in general, so the interface of the material classes is based on tensors.

Validator
The validator class is responsible for making sure the user specified a physically meaningful value of the property. Examples are the limited range of Poisson's ratio (between zero and one-half), or the requirement for a symmetric, positive definite heat conduction tensor. For properties outside of the specified ranges, an error message is generated.

Environment
Properties may depend on the local environment in the model, such as the local doping, the stress, or the temperature. Objects of the Environment class encapsulate these local parameters. The environment must be stated whenever the value of a particular property is requested.

EnvironmentParameter
Objects of this class represent one particular environment parameter, such as the temperature, or the local dopant concentration.

PropertyFunction
This class is the abstract base class for evaluating environment-dependent properties. The derived classes store all the information necessary to evaluate the value of the property at the given environment. The class also provides an operation to evaluate derivatives of the properties with respect to a given environment parameter, which has to be stated as an object of the class EnvironmentParameter.

PropertyFunctionConstant
The basic property function, stating that the property does not depend
4.3 Design of the Simulator

on the environment at all. Used as the default when no environment-depen-
dence is specified by the user.

PropertyFunctionTable
A material property where the user specified a table for the environment
dependence. Between tabulated values, the property is linearly interpo-
lated.

PropertyFunctionTaylor
The dependence of the property on the environment is specified as a
taylor series around a specific default environment. Polynomials up to
the second degree have been implemented. This is used for materials
where the properties and their temperature coefficients have been mea-
sured at a reference temperature.

Element
Each finite element has one structure associated with it. A structure may
be shared by a multitude of elements.

MaterialDatabase
The material database manages the collection of materials with its prop-
erties.

This infrastructure needs to be filled with material properties by the user. Because
big efforts have been made in our institution to measure the thin-film properties
of materials used in microsystems, we have implemented the web-based front-end
to the measured properties, the ICMAT system, so that the required input files are
directly generated from the measured data entered by the characterization engi-
neers [VonArx98].

4.3.5 Systems and Solvers

The System class is responsible for collecting the terms into numerical matrices,
and call appropriate solvers to obtain solutions to the equations described by these
matrices. For each analysis method in Figure 3.1, there is a corresponding class in
the systems hierarchy shown in Figure 4.9.

Because a large variety of methods for solving stationary problems exist, the static
systems are specialized further. The requirements on the storage format is very
different for direct and iterative solvers. Iterative solvers only require the matrix-
vector product of a linear system to be evaluated, so the matrix format can be opti-
mized accordingly. We show two iterative solvers that we used, the conjugate gra-
dient and the generalized minimal residual [Saad96] [Barrett94]. We also provide
an interface to direct solvers. With this hierarchy, we achieved a great flexibility
in adding new, state-of-the-art solvers when they become available, without adding coupling to the details of the finite element method.

We required the same flexibility for the time-stepping solvers. The methods and algorithms for time-stepping have been described in Section 3.4. The class diagram of the design for the time-steppers is shown in Figure 4.10.

The central class in the design is the interface a time-stepper sees of a time-stepping problem. As seen in Section 3.4, the time-stepping problem can always be formulated as

\[ \dot{x}(t) = f(x, t) \]  

(4.2)

So the main interface for the time-steppers just require the evaluation of the function \( f \). This interface is represented by the class ProblemDynamic, and steppers that solve such problems are derived from the class TimeStepper. Two concrete classes are derived from this class: One implements the forward Euler scheme, and the other, the class TimeStepperRK, implements the quality-controlled Runge-Kutta solver described in Section 3.4.2.
Some more sophisticated solvers, especially implicit or semi-implicit time-step- pers, additionally require the evaluation of the jacobian of the forcing-function, and the partial derivative with respect to time, that is, the two terms

\[ \nabla f \] (4.3)

and

\[ \frac{\partial f}{\partial t} \] (4.4)

Steppers that require this interface are of the class TimeStepperGradient, and the problem interface, providing the evaluations of the required terms, is defined by the abstract class ProblemDynamicGradient. Because of the substitution principle, which states that a sub-class can always be used in place of its superclass, the solvers that solve problems of the class ProblemDynamic can also solve problems of the class ProblemDynamicGradient, but they do not use the additional information provided by the gradient.

The class SystemStepper, which is multiply derived from the System class and the ProblemDynamicGradient class, has the responsibility of providing the time stepper with the standard interface to the finite element world. It can request assembly
from the finite element system under the control of the time stepper, and return the
derivatives and jacobian matrices. The advantage of this design is that implement¬
ing new time-stepping methods can be done without any knowledge about the
finite-element part of the problem.

4.4 Implementation Issues

4.4.1 Programming Language and Operating System

Previous experience in our organization existed with developing software in C++
under the Solaris operating system [Bächtold97]. The excellent portability and
availability of the unix operating system led us to keep this development platform.
The FEMEngine is now a system consisting of about 470 classes written in C++.

4.4.2 Foundation Class Library

In the course of a previous project at our institution, the LibBase, a base library of
C++ classes for collections, numerical vectors and matrices, linear solver and
input/output, has been developed for the boundary element simulation package
BEMModule [Bächtold97]. We strived to re-use as much as possible from the
LibBase, so we used the classes in the library as a basis for our development. In
the course of the new project, we constantly extended, maintained and optimized
the classes in the LibBase. The use of this library has greatly sped up the start-up
phase of the project. We have also added some classes to the LibBase that are of
interest to the entire C++ community [Taschini98].

4.5 Managing Software Development

When implementing complex software systems, the issues of managing the devel¬
opment have to be addressed. Complex software is developed in a team, and tools
have to be used to make teamwork on a large set of source files as efficient as pos¬
sible. Another issue is the stabilization of development goals and successfully
implemented use cases. To this end, a testing environment has to be set up to allow
continuous testing of the existing features to find bugs caused by changes in the
4.5 Managing Software Development

program, and to set up tests for the newly implemented use-cases. We present the
tools and methods we used for these tasks in this project.

4.5.1 Team Development and Version Control

Software as complex as described in this thesis must be implemented in teams.
The resulting system is large, and therefore is beyond the power of a single de¬
veloper. When developing software in a team, it is crucial to decouple the indi¬
vidual developers. During the development iterations of the software system, ex¬
periments are made that cause the code to be in an inconsistent or non-functioning
state, which causes it to be either not compilable or to run giving wrong or incon¬
sistent results. If all the developers work on the same copy of the code, they all
suffer from these experiments, and they have to constantly coordinate the changes
they make. What makes things worse is that is extremely hard to keep track of the
changes made, the bugs fixed and introduced, and to stabilize the working features
that have been achieved by the current implementation. Team development with¬
out decoupling and versioning is therefore practically impossible.

The decoupling of the developers and the management of the changes is obtained
by using specialized software tools. Such tools have to provide three major func¬
tions. First, they must let the developers make experiments on their local copies
of the source files. By doing this, the developers become out of sync, and the tools
must therefore provide, as a second function, a way to synchronize the developers
again, after they have finished, tested and stabilized their changes. The third func-
tion of the tools is to make it possible to understand the changes made, keep track
of them, and make it possible to undo the changes at any time, should the changes
have introduced errors.

A further complication of the situation arose in this project because of collabora-
tion with a team at a remote site. For this case, it is desired for all developers on
the project to change and re-synchronize the source code. To allow this, one site
is appointed to be the main repository site. The server machine on this site has to
run a server to allow remote check-in and check-out over the internet.

In developing the FEMEngine, the version control system in use was CVS, the
Concurrent Version System developed by volunteers for the Free Software Foun-
dation, and is maintained by Cyclic Software [CyclicWWW]. It provides all the
necessary features: It allows decoupled development in a team, provides methods to synchronize source code, and additionally allows to be run in a client-server mode over a network.

### 4.5.2 Testing and Quality Control

In the development of software testing is crucial. From the project management point of view, tests stemming from use cases are the most important. They monitor what use cases have been implemented, and therefore reflect the current status of the project. Repeated, automatic running of a suite of tests makes sure that iterations in the development cycle do not destroy the correct implementation of use cases that were achieved in previous iterations.

In the specific case of a simulation program, there are three kinds of tests:

- **Analytical tests**, where the result of the simulation is compared to the analytical solution of the problem
- **Cross-simulator tests**, where the result of the simulation is compared with the result obtained from a foreign, commercially or freely available simulator
- **Regression tests**, where the result of the simulation with the current development version is compared with the results of an earlier version

Analytical tests are the most rigorous, but are only available for restricted geometries where analytical solutions are known. Cross-simulator tests are only available if a commercial or other simulator exists which can perform the same simulation. After being tested for the first time, analytical and cross-simulator tests can be turned into anchored regression tests. These are regression tests where the initial solution is known to be correct. The big advantage of regression tests is that they can be run automatically, and if they fail, and the results differ from previously obtained output, they can give an indication on where to find the error.

In the course of the project, we have created many tests for the FEMEngine, some analytical, some cross-simulator, and some pure regression without anchor. We will describe some of the tests in chapter 5. The resulting test suite is an indispensable tool for the development.
5 SIMULATIONS

To demonstrate the use of this new tool, we show a number of selected simulations. The examples involve:

- Harmonic thermal analysis
- Eigenmode analysis of a mechanical system
- Coupled thermomechanical harmonic analysis
- File exchange with foreign formats

5.1 Heat capacity measurement Structure

For the design and simulation of efficient Microsystems, we require knowledge about the properties of the materials used to build the devices. The properties of thin films are not necessarily equal to those of larger, bulk parts of the material, and therefore, microstructures are required to measure the properties of microsystem materials. These structures require the minimization of parasitic effects to achieve a high sensitivity with respect to the properties that are to be measured. We have applied the FEMEngine to simulate a structure that measures the heat capacity of thin films [VonArx98] [Emm98.3].

5.1.1 Device structure and operation principle

A SEM micrograph of the heat capacity measurement structure, along with a simplified mask layout is shown in Figure 5.1. The device consists of a micromachined cantilever, an integrated heater at the tip of the cantilever, and 8 thermocouple contacts to sample the temperature distribution on the device. The device is operated as follows: An AC heating current is fed into the heater at the tip. The device is operated in vacuum, so all the heat generated by the heater has to flow along the cantilever into the substrate. Because of the heat capacity, the temperature distribution is frequency-dependent. The temperature distribution is sampled at the 8 thermocouple contacts. By using a one-dimensional model for the temper-
5 Simulations

Figure 5.1 SEM micrograph (left) and simplified mask layout (right) of the heat capacity measurement structure. The cantilever has been micromachined using EDP etching technique.

ature distribution along the cantilever, and sweeping the frequency of the heating current, values for the heat conductivity and the heat capacity of the sandwich of thin films that form the cantilever are obtained. By modifying the sandwich in different structures, that is, by selectively leaving away certain layers, the contributions of the individual materials to the transient behavior are separated. Not all materials can be completely left. The metal and the polysilicon are always required to form the thermocouple contacts, as is an oxide layer for insulation.

5.1.2 Goals of the simulation

The heat conductivity and heat capacitance are extracted using a one-dimensional behavioral model. The structure has been designed to make this possible. One goal of the simulation is to take into account higher-dimensional effects, such as non-symmetries and boundary effects, to check whether the one-dimensional model is valid. From the software-development point of view, the simulation is a good test for correctness, because a direct comparison with analytical models and actual measurements is possible.
5.1.3 Simulation procedure

We have first created a finite element model of the structure, which is shown in Figure 5.2. We assumed that the one-dimensional model was correct, and inserted

the material properties, extracted from the device measurements using the 1D model, into the finite element simulator. If the simulation reproduces the measurement results, the assumption was correct. If not, the one-dimensional model is not applicable, and more complex ways have to be found to extract the parameters from the measurements. We used harmonic analysis described in section 3.3.3, and applied it to the discretized heat conduction equation.

5.1.4 Results

Figure 5.3 shows the simulated and measured temperature amplitudes at the four thermocouple contacts. Excellent agreement between measurement and simulation shows that the extraction of the material parameters using the one-dimensional model applicable to this device. Inspection of the data showed agreement
Figure 5.3 Simulated and measured temperature amplitudes at the four thermocouple contacts

of the temperature amplitude with no more than 3% of error, and an agreement of the phase within 0.2°.

A design variant of the original device was also tested. In this variant, the cantilever is no longer covered by a fully closed, uniform sheet of metal, but by thin stripes. The stripes are required to form the thermocouple contacts, but they reduce the amount of metal in the sandwich. This variant was intended to separate the influence of the metallization from the rest of the sandwich. Inspection of the simulation results, shown in Figure 5.4, shows that the phase is no longer uniform across the cantilever. A one-dimensional model is therefore not applicable for this device, and a new design has to be used to separate the heat capacity of the aluminium metallization from the rest of the sandwich.

5.2 CMOS MEMS Gas Sensor

In the course of the project to develop an electronic CMOS MEMS nose [Baltes97], a resonating beam gas sensor is being developed [Lange99]. We have simulated the harmonic response of this device using the FEMEngine [Emm98.2].
5.2 CMOS MEMS Gas Sensor

Figure 5.4 Thermal phase on a variant of the measurement structure. It uses metal stripes to obtain a reduced contribution of the metal to the heat capacity.

5.2.1 Device structure and operation principle

The gas sensor consists of a cantilever beam which is driven in resonance using a harmonically varying heating power. A gas adsorbing/absorbing thin film coating deposited on the beam and causes a weight-proportional shift in the resonance frequency of the structure. The cantilever, which is depicted in Figure 5.5, is manufactured using the 2.0 μm double metal CMOS process of AMS Austria Mikrosysteme. The beam sandwich consists of the process dielectrics and a silicon n-well, and is released from the bulk silicon by an electrochemically etch-stopped anisotropic back etch combined with an oxide etch and a reactive ion etching (RIE) step from the front-side of the wafer. A p-diffused resistor within the released n-well forms the heating resistor of the beam. The heat dissipated in the resistor causes a temperature rise in the base of the beam. Different thermal expansion coefficients in the beam sandwich lead to a deflection of the beam. For normal device operation, the deflection of the beam is measured using an integrated piezoresistive stress sensor, and feedback circuitry keeps the sensor in resonance.
5 Simulations

Figure 5.5 SEM micrograph of the CMOS gas sensor. The whole cantilever is coated with a selectively absorbing thin film. The stress sensor is used to drive the cantilever in resonance using feedback circuitry.

5.2.2 Goals of the simulation

The designers of this sensor required to understand the interaction between the geometry of the device and the amplitude at resonance at a given heating power amplitude. The goal of the simulation was to reproduce the measured frequency spectrum of the device, that is, the amplitude of the tip deflection as a function of the frequency of the AC heating power fed into the heating resistors.

5.2.3 Physical Model and Discretization

The thermomechanical model described in section 2.2.2 was used in the simulation. We have applied eigenmode analysis (section 3.3.3) to the linear mechanical equation to obtain the shapes of the resonant modes and resonance frequencies of the device. We then applied harmonic analysis, as described in section 3.3.3, to perform frequency sweeps and obtain the amplitude and phase of the beam tip deflection at a large set of frequencies.

5.2.4 Results

Due to structural and excitation symmetry, the simulation domain was halved, cutting the beam geometry along its longitudinal axis, yielding a mesh of $N = 380$ nodes. Note that this step automatically excludes anti symmetric vibration modes, a reasonable assumption for the device as observed from measurement results. We then compared the computed frequency of the first mode, and found that it
matched the measured value to within the tolerance of the measured material [Baltes96] and geometric parameters. Next, damping was considered. The beam does not suffer from squeezed air film effects, because the gap below the beam is of the order of the wafer thickness 500μ, whereas the maximum beam amplitude is less than 85 nm. The dominant damping effect is therefore viscous air damping. We used a semi-empirical model to account for the air and structural damping. From the measurements, the combined damping tensor η, see Table 2.2, for the beam was determined. The frequency response of the deflection at the tip of the beam is plotted in Figure 5.6, for the range of angular frequencies around the first
to the fifth mechanical resonance mode, using a heating power of 200 mW.

The method correctly predicts the first three beam resonance frequencies as shown in Figure 5.7. The remaining resonance peaks could not be experimentally verified, because the laser interferometer used to determine the beam tip velocity and displacement has a resolution limit around 0.05 nm, as indicated by the horizontal

![Figure 5.6 Beam tip amplitude of the chemical microsensor. The insets show the mode shape of the beam at the resonance frequencies.](image)

97
Figure 5.7 Comparison of measurement and simulation

The dependence of the amplitude of the beam tip deflection on the heating power has been shown to be linear \cite{Brand94}, so that the figure-of-merit in designing the resonating beam is the ratio of amplitude to heating power. Given the geometry, the material parameters, and the Q factor, we are able to correctly compute this figure-of-merit.
5.3 RF Switch

To allow the embedding of the FEMEngine into a TCAD system, it is required that foreign file formats can be read. One demonstrator example is that of a radio-frequency switch developed at Stanford university [Wilson98]. The research group provided us with a geometry they had obtained by using their own process simulator, followed by a mesh generation step using a commercially available mesher. We were able to read this mesh, and apply a load to the structure simulating the electrostatic force on the switch. The simulation result and graph produced by our visualization tool is shown in Figure 5.8.

Figure 5.8 RF switch deflected by a uniform force to model the actual electrostatic force. The shading indicates the magnitude of the displacement in the structure.
6 Outlook

The FEMEngine is a research tool, used to explore the possibilities and challenges in microsystem simulation. The extension of the tool should be possible with very little work. This was the main driving force behind choosing the object-oriented approach for the architecture. We recommend the following topics for future extensions:

- new physical effects, such as fluid flow, diffusion in process simulation, drift-diffusion in semiconductors, theory of thin shell structures, theory of beams, thermal radiation
- new material models, such as plasticity
- more complex shape function sets for higher accuracy
- additional solvers for eigenproblems, linear equation systems, and time-stepping problems
- additional analysis methods, such as the Laplace-transform
- addition of different discretization schemes, such as boundary elements, finite differences and control volume methods
- inclusion of circuit components to model circuitry and lumped components along with the microsensor or microactuator

We now discuss how our design simplifies work in these directions, compared to more traditional implementations.

The traditional implementation of finite element programs has been strongly influenced by structured analysis and design, briefly described in section 4.1. The flexibility of these implementations is in the subroutines that form an element. These subroutines calculate the terms from the discretized equation and provide the shape functions and their required gradients. The result is a collection of elements, where each element has a shape, a set of shape functions, and a certain physical model associated with it. If the collection of elements is expanded by a new physical model, this model will only be available for a certain shape and a certain set of shape functions.
In Figure 6.1, we show three of the dimensions in which a finite element program can be extended. We have taken great care in the design of the FEMEngine to keep these dimensions orthogonal. The work for extending the FEMEngine to new areas is therefore greatly reduced. If we add a new PDE or physical model, we can
calculate solutions using any kind of geometry, using all the existing shape functions. If a new set of higher-order shape functions is added, all PDEs can immediately be calculated using these basis functions. If, on the other hand, a new analysis method, for example an analysis using Laplace transformations, is added, it can immediately be used by all physical models, with all the shape functions.

The task of adding physical models probably is the most frequent request by microsystem designers. We further simplified the task of adding PDEs by decomposing them into a set of operators, as shown in section 4.3.2. This allows a great amount of re-use of existing code, if the new PDE shares operators or even whole sub-PDEs. But even completely new PDEs, that may even pose special requirements on the shape function set, can be implemented in a de-coupled, orthogonal way.

Adding various discretization schemes such as the boundary element method would still re-use a large part of the framework that we have created. Also in the BEM, there are shape functions, elements, and material properties, and the resulting equation system has to be solved using linear and non-linear solvers. Because there already is an object-oriented implementation of the BEM [Bächtold97], that rests on the same base library, it would be possible to completely merge the two methods into one simulator, and use the synergies by applying each FEM and BEM to areas where they are strongest.

Another extension is the inclusion of simple circuit elements to allow co-simulation of circuitry and device. Circuits are described by ordinary differential equations, and therefore can be treated in exactly the same fashion as a FE-discretized model. All the implemented analysis methods, from stationary to transient and harmonic, would automatically be available for the simulation of the circuits.

These extensions would be possible without prohibitive cost, because the framework for solving equations, and for the manipulation of numerical objects, already exists.
7 CONCLUSION

We have presented the successful design, implementation and application of the FEMEngine, a multi-physics, multi-analysis simulation tool for microsystems.

We have shown how the object-oriented approach used in the design of the FEMEngine leads to a flexible, extensible architecture and have shown the benefits obtained from the synthesis of traditional finite element techniques with object oriented technology. We have presented how our architecture provides a flexible framework for implementing a wide variety of physical models, discretization techniques and analysis methods.

The simulation examples have illustrated that we have implemented our design successfully. We have simulated real-life microsystems and have helped designers to better understand and optimize their devices.
A.1 The Rotational Square Gradient Identity

In this section, the identity

\[ \nabla_x \cdot \nabla \nabla_A (\nabla \times A)^2 = -2(\nabla \times \nabla \times A) \]  \hspace{1cm} (A.1)

is derived. This is done by passing to index notation. First, an index-based form for the squared rotation is given, by using the totally antisymmetric 3x3x3 tensor \( \varepsilon_{ijk} \)

\[ (\nabla \times A)^2 = \sum_{j,k,l,m} \varepsilon_{ijk} \frac{\partial}{\partial x_j} A_k \varepsilon_{ilm} \frac{\partial}{\partial x_l} A_m \]  \hspace{1cm} (A.2)

To simplify the notation, the gradient of the vector potential \( A \), which is a 3x3 tensor, is defined to be called \( C \) as follows

\[ C_{jk} = (\nabla A)_{jk} = \frac{\partial}{\partial x_j} A_k \]  \hspace{1cm} (A.3)

Eqn. (A.2) then becomes

\[ (\nabla \times A)^2 = \sum_{j,k,l,m} \varepsilon_{ijk} C_{jk} \varepsilon_{ilm} C_{lm} \]  \hspace{1cm} (A.4)

The gradient with respect to the gradient \( \nabla_A \) is then written as

\[ (\nabla_{\nabla A})_{pq} = \frac{\partial}{\partial C_{pq}} \]  \hspace{1cm} (A.5)

so the gradient on the left hand side of Eqn. (A.1) becomes
The divergence of a tensor of rank 2, which in index notation is

\[(\nabla_x \cdot D)_q = \sum_p \frac{\partial D_{pq}}{\partial x_p}\]  

(A.7)

is applied to Eqn. (A.6), yielding

\[\left(\nabla_x \cdot \sum_{i,j,k} 2\epsilon_{ipq}\epsilon_{ijk}C_{jk}\right)_q = \sum_{p,i,j,k} 2\frac{\partial}{\partial x_p}(\epsilon_{ipq}\epsilon_{ijk}C_{jk}) = \]

\[\sum_{p,i,j,k} 2\frac{\partial}{\partial x_p}\epsilon_{ipq}\epsilon_{ijk}\frac{\partial A_k}{\partial x_j} = 2 \sum_{p,i,j,k} \epsilon_{ipq}\frac{\partial}{\partial x_p}\epsilon_{ijk}\frac{\partial A_k}{\partial x_j}\]  

(A.8)

which is

\[2 \sum_{p,i,j,k} \epsilon_{ipq}\frac{\partial}{\partial x_p}\epsilon_{ijk}\frac{\partial A_k}{\partial x_j} = -2 \sum_{p,i,j,k} \epsilon_{ipq}\epsilon_{ijk}\frac{\partial A_k}{\partial x_j} = -2(\nabla \times \nabla \times A)_q\]  

(A.9)
A.2 UML Notation and Terminology

We give a short overview of the graphical elements in the unified modeling language. The full notation and terminology is documented in [Rumbaugh99].

<table>
<thead>
<tr>
<th>Material</th>
<th>name: String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>propertyValues: ListOfValues</td>
</tr>
<tr>
<td>GetName(): String</td>
<td>PropertyValue(whichProperty): Value</td>
</tr>
</tbody>
</table>

![Diagram of UML class diagram elements]

**Figure A.1** UML class diagram elements
The format of the references is

[Author Year] Last Name1, Initials1/Last Name2, Initials2/.../Last Name n, Initials n “Title”, Publisher, Location of Publisher, Year, (Pages in this thesis the work is cited on)


[Booch94] Booch, G. "Object-Oriented Analysis and Design", Benjamin/Cummings, Redwood City, 1994 (73,73)


[IntelliSense Corporation, website, http://www.intellisense.com]

[ISE Integrated Systems Engineering AG, “ISE TCAD User Manuals”, ISE AG, Technoparkstrasse 1, Zurich, Switzerland, 1998 (6,7)]


[Korvink, J. G. “An implementation of the adaptive finite element method for semiconductor sensor simulation”, Ph.D. Thesis No. 10143, ETH Zurich, Verlag der Fachvereine, Zurich, 1993 (60,60)]


[Ziebart99] Ziebart, V./Paul, O./Baltes, H. “Strongly Buckled Micromachined Square Membranes”, IEEE Journal of Microelectromechanical Systems, accepted for publication (44)
ACKNOWLEDGEMENTS

First of all, I would like to thank Prof. Dr. Henry Baltes who made it possible for me to work on this project, and provided an excellent environment for creativity. He guided and supported the project, and through excellent diplomacy sailed the project ship around the shoals of research politics.

My greatest thanks go to my co-examiner Prof. Dr. Jan G. Korvink. He supported and motivated me during the whole project, and offered me guidance, advice and friendship. He helped me through the crises and inspired me with his enthusiasm about the achievements. He also put a tremendous amount of work into helping me finish the text of the thesis.

I'd like to thank Prof. Dr. Niels Kuster for co-examining this thesis, and Prof. Dr. Arockia Nathan for fruitful discussions and suggestions.

Special thanks go to Stefano Taschini for his strong collaboration on this project. Without him, this project would never have gotten as far as it has. I will always remember the many interesting discussions about the object-oriented design of our software, but also about the comparison of human languages, swiss and italian cuisine, politics, and about science-fiction literature and movies.

I thank Dr. Martin Bächtold for getting me started on this project, and for the discussions which more and more clarified the ideas for object-oriented numerical software. He also laid the foundations of our code by implementing the LibBase. I also thank Dr. Jörg Funk for the collaboration on the material database, and for his providing an interface to the ISE Integrated Systems Engineering company. Thanks go to Prof. Dr. Oliver Paul and Dr. Martin von Arx for providing the idea of implementing harmonic analysis in our finite element package, and helping me with first-hand knowledge to apply our simulator to their material characterization structures, and to Luca Plattner for performing some actual simulations. I also thank Dr. Stefan Koller, Dirk Lange and Dr. Mark Hornung for inspiring me with hints about actual, real life devices. The work that Lorenza Ferrario and Martin Bossard put into the FEMEngine is also gratefully acknowledged.
Of course I also thank our system administrators Igor Levak, Max Schlapfer and Chris Kolb for keeping the computer systems operational, and Erna Hug for organizing everything.

I’d like to thank the people at the ISE Integrated Systems Engineering company, especially Dr. Hans-Petter Lien, Dr. Lars Bomholt, Dr. Karine Lamboglia, Dr. Diederik Fokkema, and Dr. Roland Rühl, as well as Dr. Gilda Garretton at the IIS for their help with their TCAD software products.

I thank Dr. Andreas Greiner, Jens Müller, Ricardo Osorio, Stefan Maier and Sadik Hafizovic for the interesting discussions and the friendly atmosphere during my short visit at Freiburg University while writing up this thesis.

I also thank all the co-workers at PEL for the friendly atmosphere they helped create, namely Dr. Daniel Bolliger, Dr. Oliver Brand, Dr. Johannes Bühler, Dr. Ruggero Castagnetti, Christoph Hagleitner, Dr. Andreas Hierlemann, Dr. Andreas Hämmerli, Dr. Dominik Jaeggi, Nicole Kerness, Andreas Koll, Dr. Stefan Linder, Christoph Maier, Dr. Piero Malcovati, Dr. Felix Mayer, Michael Mayer, Dr. Matthias Metz, Thomas Müller, Ulrich Münch, Andri Schaufelbühl, Donat Scheiwiller, Dr. Klaus Schneeberger, Dr. Michael Schneider, Dr. Franz-Peter Steiner, Dr. Ralph Steiner, Dr. Rolf Vogt, Yelena von Allmen, Marc Wälti, Volker Ziebart and Martin Zimmermann.

I would also like to thank the many programmers that have contributed to the free software tools that we have been using extensively in this project, especially all the developers that are behind the gnu tools emacs, gcc, cvs and res. Without them, developing software would be much harder.

And finally, I want to thank my wife Evelin for her love and support.
Curriculum Vitae

Markus Emmenegger

Born October 6, 1969

Citizen of Luzern and Schüpfheim (LU), Switzerland

1978-1982  Primary School Cham (ZG), Switzerland
1982-1984  Secondary School Cham (ZG), Switzerland
1984-1989  Kantonsschule Zug, Switzerland
1989      Matura, Type B
1990-1995  Student of Physics at the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland
1995      Diploma in Physics of the ETH Zurich, Switzerland
1996-1999  Work on Ph.D. thesis and related topics at the Physical Electronics Laboratory, directed by Prof. Dr. H. Baltes, at the ETH Zurich
ABBREVIATIONS AND SYMBOLS

2D two-dimensional
3D three-dimensional
A system matrix
A' unconstrained system matrix
A magnetic vector-potential
a \textit{i-th} term to enter the system matrix
a vector of discretized vector potential values
AFM atomic force microscope
B matrix defining the constraint equations
B magnetic flux density
b \textit{j-th} term to contribute to the right-hand side
BEM boundary element method
C C programming language
C heat capacity matrix
\[ C = [C_{ijkl}] = [C_{klij}] \] 4th rank elasticity tensor
c speed of light
c heat capacity per unit volume
c constant vector in transformation from unconstrained to constrained DOFs
C++ C++ programming language
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>conjugate gradient, an iterative solution method for linear equations</td>
</tr>
<tr>
<td>CMOS</td>
<td>complementary metal-oxide semiconductor, a logic family and the related process technology</td>
</tr>
<tr>
<td>$D$</td>
<td>dissipative content function</td>
</tr>
<tr>
<td>$d$</td>
<td>right-hand side in constraint equation</td>
</tr>
<tr>
<td>$d(x, t)$</td>
<td>displacement vector field in elastodynamics</td>
</tr>
<tr>
<td>DOF</td>
<td>degree of freedom</td>
</tr>
<tr>
<td>$E$</td>
<td>electric field</td>
</tr>
<tr>
<td>$F(T, \varepsilon)$</td>
<td>density of the Helmholtz free energy</td>
</tr>
<tr>
<td>$f(x)$</td>
<td>body force density vector field</td>
</tr>
<tr>
<td>FEM</td>
<td>finite element method</td>
</tr>
<tr>
<td>$G$</td>
<td>electro-magnetic coupling matrix</td>
</tr>
<tr>
<td>$G$</td>
<td>boundary element flux matrix</td>
</tr>
<tr>
<td>GMRES</td>
<td>generalized minimal residual, an iterative solution method for linear equations</td>
</tr>
<tr>
<td>$H$</td>
<td>boundary element temperature matrix</td>
</tr>
<tr>
<td>$h$</td>
<td>heat source density</td>
</tr>
<tr>
<td>IC</td>
<td>integrated circuit</td>
</tr>
<tr>
<td>$K$</td>
<td>stiffness matrix</td>
</tr>
<tr>
<td>$L$</td>
<td>Lagrangian</td>
</tr>
<tr>
<td>$L$</td>
<td>heat conduction matrix</td>
</tr>
<tr>
<td>$M$</td>
<td>mass matrix</td>
</tr>
<tr>
<td>MEMS</td>
<td>micro-electro-mechanical systems</td>
</tr>
<tr>
<td>$N_i$</td>
<td>i-th basis function or shape function</td>
</tr>
<tr>
<td>ODE</td>
<td>ordinary differential equation</td>
</tr>
<tr>
<td>$p$</td>
<td>vector of discretized electric potential values</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>PDE</td>
<td>partial differential equation</td>
</tr>
<tr>
<td>$Q$</td>
<td>forcing function</td>
</tr>
<tr>
<td>$q(x, t)$</td>
<td>vector field of the generalized displacements</td>
</tr>
<tr>
<td>RHS</td>
<td>right-hand side</td>
</tr>
<tr>
<td>RIE</td>
<td>reactive ion etching</td>
</tr>
<tr>
<td>RK</td>
<td>Runge-Kutta time integration</td>
</tr>
<tr>
<td>$S$</td>
<td>vector potential &quot;mass&quot; matrix</td>
</tr>
<tr>
<td>$dS(x)$</td>
<td>measure in surface integrals</td>
</tr>
<tr>
<td>$T$</td>
<td>transformation matrix from generalized to constrained degrees of freedom</td>
</tr>
<tr>
<td>$T^*$</td>
<td>kinetic co-energy</td>
</tr>
<tr>
<td>$t$</td>
<td>time</td>
</tr>
<tr>
<td>$U$</td>
<td>magnetic field energy storage matrix</td>
</tr>
<tr>
<td>UML</td>
<td>unified modeling language</td>
</tr>
<tr>
<td>$q_{\perp}$</td>
<td>surface heat flux density</td>
</tr>
<tr>
<td>$V$</td>
<td>potential energy</td>
</tr>
<tr>
<td>$v$</td>
<td>vector of unconstrained degrees of freedom</td>
</tr>
<tr>
<td>$w$</td>
<td>weighting function</td>
</tr>
<tr>
<td>$w(x, x_0)$</td>
<td>fundamental solution for the Laplace operator</td>
</tr>
<tr>
<td>$x$</td>
<td>position vector in space</td>
</tr>
<tr>
<td>$\alpha(x, t)$</td>
<td>density of external forces</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>thermal expansion coefficient tensor</td>
</tr>
<tr>
<td>$\Gamma_N$</td>
<td>Neumann boundary</td>
</tr>
<tr>
<td>$\Gamma_D$</td>
<td>Dirichlet boundary</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>current source vector</td>
</tr>
</tbody>
</table>
Δ  dissipation matrix
δ_{ij}  Kronecker delta symbol, defined by δ_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}
\Lambda  time step width
\varepsilon  strain tensor
\varepsilon_{ijk}  totally antisymmetric 3x3x3 tensor
\varepsilon_0  dielectric constant
\eta = [\eta_{ijkl}] = [\eta_{klji}]  4th rank viscosity tensor
\eta  charge source vector
\theta(t) = \int_{t_0}^{t} T(t') dt'  primitive function of the temperature, so that T = \dot{\theta}
κ  heat conductivity tensor
Λ  Lagrange density
Λ_F  free electromagnetic field Lagrange density
Λ_I  interaction Lagrange density
λ  vector of Lagrange multipliers
λ  buckling load
μ_0  magnetic permittivity in vacuum
\nu  potential energy density
ρ  mass density
\rho  charge density
σ  stress
τ^\varphi  kinetic co-energy density
Φ  electrostatic influence matrix
\phi  content density
\phi  interatomic potential
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\psi$</td>
<td>electric potential</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>domain</td>
</tr>
<tr>
<td>$\partial \Omega$</td>
<td>boundary of the domain $\Omega$</td>
</tr>
<tr>
<td>$\omega$</td>
<td>angular frequency</td>
</tr>
<tr>
<td>$\nabla$</td>
<td>nabla, the gradient operator</td>
</tr>
<tr>
<td>$\nabla_x$</td>
<td>gradient operator with respect to $x$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>empty set</td>
</tr>
<tr>
<td>$I$</td>
<td>identity matrix</td>
</tr>
</tbody>
</table>
# Index

<table>
<thead>
<tr>
<th>A</th>
<th>AC analysis</th>
<th>46</th>
<th>attribute</th>
<th>84</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>backward Euler</td>
<td>52</td>
<td>Bader</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>BEM</td>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Booch</td>
<td>73</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>boundary elements</td>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>buckling</td>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>C++</td>
<td>73, 88</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cash-Karp</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>class</td>
<td>72</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>conjugate gradient</td>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraints</td>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>crystal</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CVS</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Deuflhard</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>differential equations</td>
<td>23, 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>9, 34</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dirichlet</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>discretization</td>
<td>23, 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DiscretizedOperator</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>displacement</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dissipation matrix</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>eigenmode analysis</td>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>eigensystem</td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>elastic energy</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>elastodynamics</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>electric field</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>electrodynamics</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>electro-magnetic coupling matrix</td>
<td>26, 122</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>electrostatic influence matrix</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Element class</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>energy</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>elastic</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>internal</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>entropy</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Environment</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EnvironmentParameter</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>error control</td>
<td>49</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Euler</td>
<td>47, 52</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>extrapolation</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>finite elements</td>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>floating boundary condition</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>forward Euler</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>GMRES</td>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>harmonic analysis</td>
<td>33, 45</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hooke’s law</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>implicit time-steppers</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ion</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>Lagrange multipliers</td>
<td>67</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lagrangian</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>lattice</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LibBase</td>
<td>88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>magnetic field energy storage matrix</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>magnetic flux density</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mass matrix</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>material database</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maxwell relation</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maxwell’s equations</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>Term</td>
<td>Page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------</td>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>stiffness matrix</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Newton-Raphson</td>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>object-oriented method</td>
<td>71</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ODE</td>
<td>23, 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>operator</td>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>PropertyFunction</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PropertyID</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>quadrature</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>regression test</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>requirements</td>
<td>74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RK</td>
<td>Runge-Kutta</td>
<td>48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>semi-implicit Euler</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ShapeFunctionSet</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Solver</td>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stability</td>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stationary solution</td>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stiff equations</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>testing</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>time marching</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TimeStepper class</td>
<td>86</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T-Matrix</td>
<td>62, 68</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T-Node</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>transient</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>UML</td>
<td>73, 109</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unix</td>
<td>88</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use Case</td>
<td>74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>Validator</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>version control</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>weighted residual</td>
<td>34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>