Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

INGEGNERIA DELLE MACCHINE E DEI SISTEMI ENERGETICI

Ciclo XXII

Settore/i scientifico-disciplinare/i di afferenza: ING-IND/08

A MODEL FOR DIFFUSIVE AND DISPLACIVE PHASE TRANSITIONS IN STEELS

Presentata da: Mirko Maraldi

Coordinatore Dottorato:

Chiar.mo Prof. Ing. Davide Moro

Relatore:

Chiar.mo Prof. Ing. Pier Gabriele Molari

Co-Relatore:

Dr. Garth N. Wells

Esame finale anno 2010

“et Pangloss disait quelquefois à Candide: Tous les événements sont enchaînés dans le meilleur des mondes possibles; car enfin si vous n’aviez pas été chassé d’un beau château à grands coups de pied dans le derrière pour l’amour de mademoiselle Cunégonde, si vous n’aviez pas été mis à l’inquisition, si vous n’aviez pas couru l’Amérique à pied, si vous n’aviez pas donné un bon coup d’épée au baron, si vous n’aviez pas perdu tous vos moutons du bon pays d’Eldorado, vous ne mangeriez pas ici des cédrats confits et des pistaches. ”

Voltaire, *Candide, ou l’optimisme.*

Abstract

Heat treatment of steels is a process of fundamental importance in tailoring the properties of a material to the desired application; developing a model able to describe such process would allow to predict the microstructure obtained from the treatment and the consequent mechanical properties of the material.

A steel, during a heat treatment, can undergo two different kinds of phase transitions [p.t.]: diffusive (second order p.t.) and displacive (first order p.t.); in this thesis, an attempt to describe both in a thermodynamically consistent framework is made; a phase field, diffuse interface model accounting for the coupling between thermal, chemical and mechanical effects is developed, and a way to overcome the difficulties arising from the treatment of the non-local effects (gradient terms) is proposed.

The governing equations are the balance of linear momentum equation, the Cahn-Hilliard equation and the balance of internal energy equation. The model is completed with a suitable description of the free energy, from which constitutive relations are drawn.

The equations are then cast in a variational form and different numerical techniques are used to deal with the principal features of the model: time-dependency, non-linearity and presence of high order spatial derivatives.

Simulations are performed using *DOLFIN*, a *C++* library for the automated solution of partial differential equations by means of the finite element method; results are shown for different test-cases. The analysis is reduced to a two dimensional setting, which is simpler than a three dimensional one, but still meaningful.

Contents

List of Symbols **xi**

1 Introduction **1**

1.1 From Heat Treatments Modelling To Thermodynamics of Continuum Media 1

1.2 State Of The Art In Phase Transitions Modelling 4

1.3 Outline Of The Thesis 7

1.4 The Material Body 8

2 Order Parameters **11**

2.1 Introduction 11

2.2 Displacive order parameter 12

2.3 Diffusive order parameter 14

2.3.1 Diffusive Order Parameter Suitable For Macroscopic Scale 14

3 Equations Of The Model And Thermodynamic Restrictions **17**

3.1 Introduction 17

3.2 Balance Equations 17

3.2.1 Conservation of mass and chemical powers 18

3.2.2 Balance of momentum and mechanical powers 19

3.2.3 Balance of energy 20

3.3 Thermodynamic Restrictions For Admissible Processes: The Second Law Of Thermodynamics 21

3.4 Helmholtz Free Energy 24

3.4.1 Diffusive part of the free energy 25

3.4.2 Displacive part of the free energy 26

3.4.3 Thermal part of the free energy 26
3.4.4 Phase-coupling part of the free energy 26
3.5 Governing Equations 27
3.5.1 Explicit constitutive relations 28
3.5.2 Boundary and initial conditions for the differential system 29

4 Weak Form Of The Equations Of The Model 31

4.1 Introduction 31
4.2 Cahn-Hilliard Equation 32
4.2.1 Semi-discrete case 32
4.2.2 Fully-discrete case 32
4.3 Balance Of Linear Momentum Equation 33
4.3.1 Semi-discrete case 33
4.3.2 Fully-discrete case 35
4.4 Heat Equation 37
4.4.1 Semi-discrete case 38
4.4.2 Fully-discrete case 38

5 Computer Code For The Equations Of The Model 41

5.1 Introduction 41
5.2 The *FEniCS* Project 42
5.3 Equations Coding In *DOLFIN* 44
5.3.1 Variational form 46
5.3.2 Parameters of the model and linearised system solver 49
5.3.3 Initial conditions and boundary conditions 52
5.3.4 Acceleration and velocity updates 53
5.3.5 Non-linear solver 54

6 Numerical Results 57

6.1 Introduction 57
6.2 Dimensional Analysis 57
6.2.1 Dimensional basis and fundamental dimensions 58
6.2.2 Dimensionless parameters 58
6.2.3 Values of the Parameters 60

6.3 Domain, Initial And Boundary Conditions 60
6.4 Features Of A Diffusive Phase Transition 63
6.4.1 Diffusive phase transition at various temperatures 66
6.5 Features Of A Displacive Phase Transition 68
6.5.1 Displacive phase transition at various temperatures 73
6.6 Open Issues - Overshoots in Temperature 75

7 Concluding Remarks 77

A DOLFIN/FFC Code 79

B Experimental Characterisation Of An Eutectoid Steel 137
B.1 Introduction 137
B.2 Chemical Analysis 138
B.3 Cooling Test 138
B.4 U curves 140

References 143

e_3

List of Symbols

Roman Symbols

A	Free energy coefficient
B	Free energy coefficient
$\boldsymbol{b}$	Volume mechanical force
c	Diffusive order parameter
c_p	Specific heat (constant pressure)
c_s	Specific heat
D	Free energy coefficient
ds	Surface measure
$\tilde{ds}$	Interior surface measure
dw	Volume measure
E	Free energy coefficient
e_1	Hydrostatic strain
e_2	Displacive order parameter
e_3	Shear strain
F	Free energy coefficient

G	Free energy coefficient
g	Heat flux across the boundary
H	Free energy coefficient
h	Mesh size
$\boldsymbol{j}_c$	Mass flux of c
K_c	Concentration gradient coefficient
K_e	Displacive order parameter gradient coefficient
$\boldsymbol{M}$	Non-local part of the chemical potential
p	Test function
$\mathcal{P}_c^e$	Chemical external power
$\mathcal{P}_m^e$	Mechanical external power
$\mathcal{P}_c^i$	Chemical internal power
$\mathcal{P}_m^i$	Mechanical internal power
p_c^i	Specific chemical internal power
p_m^i	Specific mechanical internal power
$\partial\mathcal{S}$	Boundary of the sub-region of the body
$\boldsymbol{q}$	Heat flux
q	Test function
$\boldsymbol{r}$	Test function
$\mathcal{S}$	Sub-region of the body
s	Test function
T	Temperature

t	Time
T_M	Martensitic transition temperature
T_P	Perlitic transition temperature
$\boldsymbol{t}$	Surface mechanical force
T_{ref}	Reference temperature for undeformed configuration
$\boldsymbol{u}$	Displacement field
u	Specific internal energy
$\boldsymbol{v}$	Velocity
$\boldsymbol{i}$	First Cartesian axis versor
$\boldsymbol{j}$	Second Cartesian axis versor
$\boldsymbol{k}$	Third Cartesian axis versor
x_{12}	Mechanical coupling coefficient
x_{1c}	Chemo-mechanical coupling coefficient
x_{2c}	Phase coupling coefficient

Greek Symbols

α	Thermal dilatation coefficient
α^m	Mid-point parameter for generalised-alpha (inertial term)
α^s	Mid-point parameter for generalised-alpha (stress/external force term)
β	Damping coefficient
β_a	Weighting parameter for acceleration (Newmark approximation)
χ	Motion
$\boldsymbol{\varepsilon}$	Linearised strain tensor

$\boldsymbol{\varepsilon}$	Linearised strain tensor
η	Specific entropy
Γ	Set of state variables
γ	Mobility
γ_v	Weighting parameter for velocity (Newmark approximation)
κ	Conductivity
μ	Chemical potential
Ω	Region occupied by the body
Φ	Dissipation
$\partial\Omega$	Boundary of the region occupied by the body
ψ	Specific Helmholtz free energy
$\tilde{\partial\Omega}$	Interior boundary of the region occupied by the body
ρ_∞	Spectral radius in the high-frequency limit
$\boldsymbol{\sigma}^{an}_{loc}$	Anelastic local part of the stress tensor
$\boldsymbol{\sigma}^{el}_{loc}$	Elastic local part of the stress tensor
$\boldsymbol{\Sigma}$	Non-local part of the Cauchy's stress tensor
$\boldsymbol{\sigma}$	Cauchy's stress tensor
θ	Mid-point parameter for theta-method
ξ	Stabilisation parameter (discontinuous Galerkin)

Other Symbols

$\langle . \rangle$	Average operator
$[\![.]\!]$	Jump operator

Chapter 1

Introduction

1.1 From Heat Treatments Modelling To Thermodynamics of Continuum Media

Heat treatment of steels is a process of fundamental importance in tailoring the properties of a material to the desired application. Steel quenching, among other heat treatments, is an old-established technology still extensively used today; this process is mainly utilised for hardening the mating surfaces of bodies in contact, enhancing their mechanical properties, reducing wear, and is essential to produce useful tools for the common life and for the machining of other materials. Heat treatment of steels involves microstructural phase transitions which can be divided into *displacive* (short time) and *diffusive* (long time) transformations.

The modelling of this process is of paramount importance: it would allow to predict the microstructure - and the mechanical properties, thus - in every part of a machine component, with advantages for both the heat treatment set-up and the designing procedure. For instance, a reliable prediction of mechanical properties of the machine component would make more refined stress analysis possible.

Up to now, heat treatment modelization developed for the aforementioned purposes has been based on empirical laws which tend to describe the experimental evidences at a macroscopic scale (useful for technological applications).

Take, for example, a simple treatment achieved by heating a specimen above a critical temperature (to obtain an austenitic structure) and suddenly cooling it (fig. 1.1):

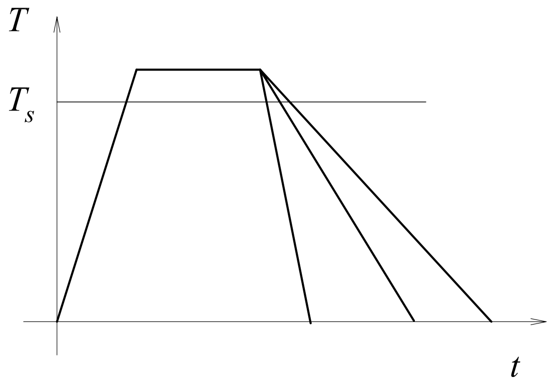


Figure 1.1: Temperature setting for typical heat treatments.

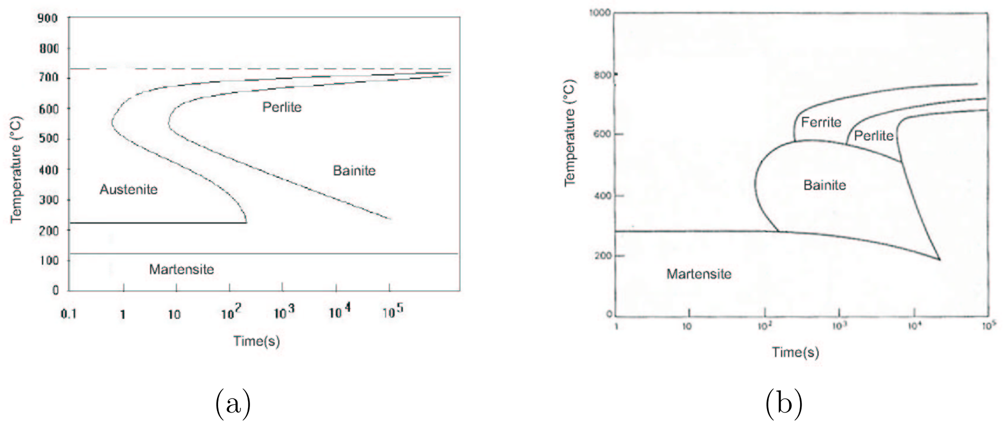


Figure 1.2: a) TTT diagram and b) CCT diagram for an eutectoid steel.

Austenite becomes unstable and new phases appear, depending on the cooling rate.

In the shop practice, as well as in the computer aided simulation [Bergheau et al. (2000)], the heat treatment design is based on Bain curves mapping; these curves are collected into two diagrams [Matteoli (1990)], namely *time-temperature-transformation (TTT) diagrams* and *continuous cooling transformation (CCT) diagrams.*

TTT diagrams describe an ideal process which can be far from the real one. In fact, they reproduce a constant temperature, long time effect after an initial quasi-instantaneous cooling of the specimen. The curves depicted on the diagram mark the onset and the end of metal structural transformations using a logarithmic scale for both time and temperature. Figure 1.2a shows a TTT diagram for an eutectoid steel.

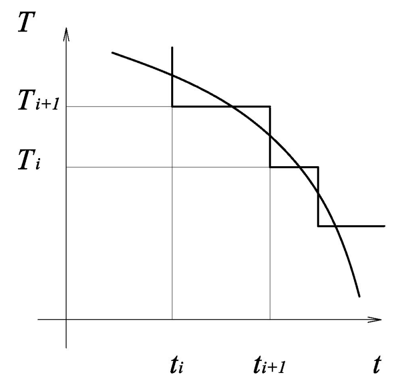


Figure 1.3: Sub-division of the cooling phase into many isothermal steps.

To account for the velocity of cooling and its influence on the phase transformations, the CCT diagrams were derived (Fig.1.2b, eutectoid steel). Again, temperature is plotted against time using a logarithmic scale.

It has to be remarked that these diagrams cannot describe the transformations and the exchange of energy occurring during a heat treatment with the demanded accuracy; for example, in many of the commonly manufactured processes, nor the temperature or the cooling rate are constant. To cope with the fact that the only diagrams available are suitable for isothermal or constant cooling velocity conditions, a common strategy consists in splitting the whole process into a number of isothermal sub-processes (fig. 1.3) for which the TTT diagrams are considered reliable and then combining the intermediate results obtained through an additivity law; one of the most widely accepted relations is the Scheil's additivity rule [Bergheau et al. (2000)], [Brokate and Sprekels (1996)].

A heat treatment model is complemented with some laws for the mass fraction evolution of the different phases involved. For example, the evolution of the phase obtained from a displacive transformation can be described by the Koistinen-Marburger rule [Brokate and Sprekels (1996)], while to predict the phase fraction obtained from a diffusive transformation the Johnson-Mehl formula can be used [Brokate and Sprekels (1996)].

The modelling approach depicted above has some drawbacks. First of all, it does not clarify the causes of the phase changes, and is far from getting the actual physics of the phenomena involved, especially in non-isothermal conditions; moreover, the mechanical effects of a heat treatment, despite being of paramount importance from a designing

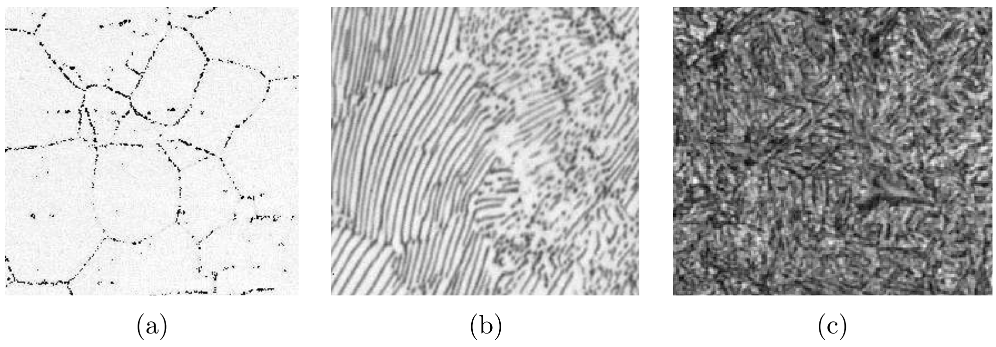


Figure 1.4: Microscope image of: (a) Austenite, (b) Pearlite and (c) Martensite [Mehl (1972)].

point of view, do not receive the deserved attention in such approaches. This makes therefore difficult the prediction of the residual strains and stresses inside the specimen, as well as the estimation of possible failures arising from a badly designed heat treatment. There is the need for looking at steel heat treatment in its generality, within a thermodynamic consistent framework; this change in the point of view would allow thermal effects, mechanical effects and phase transformations to be accounted for concurrently.

To this end, there have been some attempts [Bouville and Ahluwalia (2007)] which look at the global phenomenon starting from a physical microscopic description of the material. The heat treatment evolution is traced accounting for the diffusive as well as the displacive transformations.

For the sake of simplicity, an eutectoid steel will be considered. In this case there can only be three phases: Austenite (fig.1.4a) with a face-centered cubic lattice (fcc), Pearlite (fig.1.4b) with a body-centered cubic (bcc) lattice and Martensite (fig.1.4c) with hexagonal close packed (hcp) lattice.

Figure 1.5 shows the phase diagram for iron varying the temperature and the pressure. Note that the pressure (i.e. the stress) is also a fundamental variable for the existence of a given phase within a steel.

1.2 State Of The Art In Phase Transitions Modelling

Solid-to-solid phase transitions can be, as stated in Sect.1.1, of different nature: diffusive and displacive. Diffusive transformations are second order phase transitions, and are

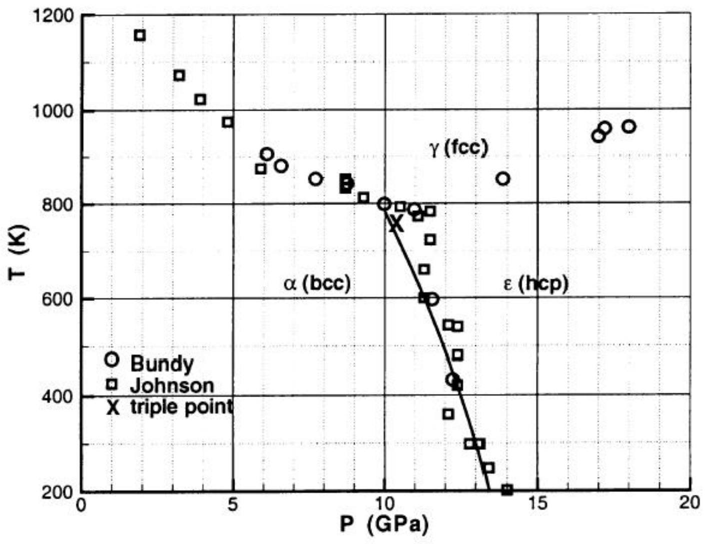


Figure 1.5: Phase diagram for iron, taken from Yano and Horie [Yano and Horie (2002)].

able to describe, among others, processes involving diffusion of atoms. Typical examples include spinodal decomposition, vapour-phase deposition, crystal growth during solidification and grain growth in single-phase and two-phase systems. Spinodal decomposition can occur, for instance, in an eutectoid steel: a diffusive transition from a high temperature stable phase, Austenite (fig. 1.4a), to Pearlite (fig. 1.4b) takes place at low cooling rate; as the temperature decreases slowly, carbon (the solute) can migrate outside iron cell and originate Cementite layers.

Displacive transformations are first order phase transitions involving a lattice distortion. Typical examples include the martensitic transformation of shape memory alloys or martensitic transformation of quenched steels: here the displacive transformation from Austenite to Martensite (fig. 1.4c) occurs at high cooling rate; the temperature decreases fast, and there is no time for diffusive phenomena to take place so that the new phase originates with a lattice deformation.

In a steel subjected to a heat treatment there may be different zones in which, due to a different cooling rate, the transformations are of different kind, leading to different phases.

As stated in Sect.1.1, a number of approaches may be adopted to model both the aforementioned phase transitions; for instance, use of phenomenological laws can be made:

Brokate and Sprekels [Brokate and Sprekels (1996)], following the approach proposed by Hömberg [Hömberg (1995)], present a model constituted by the Koistinen-Marburger rule for the prediction of the martensitic phase fraction, the Johnson-Mehl formula for the estimate of the perlitic phase fraction, together with the Scheil's additivity rule for non-isothermal conditions. However, such an approach suffers from the drawbacks explained in Sect.1.1. A different way to tackle the problem may consist in setting a framework in the context of nonlinear thermoelasticity including the Ginzurg-Landau theory of phase transitions [Ginzburg and Landau (1950)], [Tolédano and Tolédano (1987)]. Following the phase-field approach, a microstructure is identified by a variable named order parameter. The phase transitions rely on phase evolution equations; constitutive laws are expressed as the derivatives of the Helmholtz free energy density, which is given as a function of the order parameters.

In particular, a Ginzburg-Landau theory was proposed by Falk [Falk (1980)] for the martensitic phase transition in shape memory alloys using an order parameter dependent on the strain tensor. The same idea has been developed by Barsch and Krumhansl [Barsch and Krumhansl (1984)] to describe the microstructure of inhomogeneous materials and their proposed model has been extended to simulate martensitic structures in two-and three-dimensional domains. In the work by Ahluwalia et. al. [Ahluwalia et al. (2006)] this theory is generalised to describe a two-dimensional square to rectangle martensitic transition in a polycrystal with more than one lattice orientation. Other models which describe the martensitic transformation in a Ginzurg-Landau framework using a different order parameter not dependent on the strain tensor are proposed by Levitas and Preston [Levitas and Preston (2002)], Wang and Khachaturyan [Wang and Khachaturyan (1997)], Artemev et. al. [Artemev et al. (2001)], Berti et. al. [Berti et al. (in press)].

Ginzburg-Landau frameworks have also been applied to study diffusive phase transitions. In this context, Cahn [Cahn (1961)] presented a first model in the simplest case of diffusive phase transition without accounting for thermal effects. An extension to Cahn's model is presented - among others - by Alt and Pawlow [Alt and Pawlow (1992)]; in their paper the coupled phenomena of mass diffusion and heat conduction in a binary system subjected to thermal activation have been modelled. Other models accounting for the mechanical aspects related to the diffusive phase transitions are the ones by Onuki and Furukawa [Onuki and Furukawa (2001)] and Fried and Gurtin [Fried and Gurtin (1993)].

First attempts to study diffusive and displacive phase transitions together were made by Rao and Sengupta [Rao and Sengupta (2003)] and by Levitas [Levitas (2000)]; the same idea has been developed by Bouville and Ahluwalia [Bouville and Ahluwalia (2007)], who proposed a Ginzburg-Landau framework including the balance of linear momentum equation and the Cahn-Hilliard equation.

1.3 Outline Of The Thesis

The outline of the thesis is as follows:

- in Chapter 2 we will describe the order parameters used and we will declare the scale at which the model is intended to operate.
- In Chapter 3 the equations of the model will be derived within a thermodynamically consistent framework, which would allow to deal with the non-localities of the model in a satisfactory way and to derive proper constitutive relations by means of functional derivatives of the Helmholtz free energy; initial conditions, as well as suitable boundary conditions of standard and non-standard type will be proposed. An appropriate form for the Helmholtz free energy will be assumed and the explicit form of the equations of the model will be given.
- In Chapter 4 the equations of the model will be cast in a variational form and different numerical techniques will be proposed in order to treat the peculiarity of the model: we will make use of the operator-splitting approach, a discontinuous-Galerkin formulation, the theta-method and the generalised-alpha scheme; a sketch of the linearisation of each equation will also be given.
- In Chapter 5 we will explain how the model has been coded using *DOLFIN*, a *C++* library for the automated solution of partial differential equations by means of the finite element method.
- In Chapter 6 we will present the results of the simulations conducted for different test cases and we will highlight the basic features and the behaviour of the model.
- In Chapter 7 we will draw conclusions for the work done and presented in this thesis.

- Two appendices have been included: Appendix A contains the computer code developed for the solution of the system of equations, while Appendix B presents an experimental characterisation of the type of steel that the model aims to describe.

1.4 The Material Body

Let[1] $\mathcal{B}$ be a material body; in the following we will refer to the material body through its reference configuration, namely a configuration for the body free of loads and strains. In other words, we will make no distinction between the *material description* and the *reference description* [Malvern (1969)].

The material body, in its reference configuration, is supposed to occupy a simply connected open set of the Euclidean two-dimensional space; we will call Ω such region and $\partial\Omega$ its boundary, which is supposed to be sufficiently regular [2]. At a generic time t, the body will be in its *actual configuration* and will occupy a region which we will call Ω_t, an open set with a boundary $\partial\Omega_t$.

The actual configuration is described by means of a space-time mapping called *motion*:

$$\boldsymbol{x} = \chi\left(\boldsymbol{x_0}, t\right), \tag{1.1}$$

being $\boldsymbol{x_0}$ the position of a given particle in the reference configuration and $\boldsymbol{x}$ its actual position.

In the following, we will consider the *Lagrangian description* (based upon the reference configuration) to be coincident with the *Eulerian description* (based upon the actual configuration)[3]; this derives form the exploitation of the so-called *small perturbation hypothesis*, for which the determinant of the Jacobian matrix of χ is approximately equal to the identity tensor:

$$det\left(\nabla_{\boldsymbol{X}}\, \chi\right) \simeq \boldsymbol{I}; \tag{1.2}$$

this will cause integrals over subsets $\mathcal{S}$ of Ω to be equivalent to the corresponding integrals over subsets $\mathcal{S}_t$ of Ω_t.

[1]A reference for this section may be the book by Maugin [Maugin (1992)].

[2]For an analysis of the regularity requisites see the work by Del Piero [Del Piero (2007)] and the references therein.

[3]see for instance Sect. 3.3.

Moreover, we will denote with dw and ds the volume measure and the surface measure of the aforementioned integrals, respectively.

In Sect. 4.3 we will have to introduce a discontinuous Galerkin formulation for one of the equations of the model: we will call $\widetilde{\partial\Omega}$ the interior boundary of the domain.

Chapter 2

Order Parameters

2.1 Introduction

An order parameter can be defined as an indicator of the state - or a measure of the *inner order* - of a material medium; for phase change models it can act as a marker of the phase.

Its introduction dates back to the 50s, when a paper by Landau and Ginzburg [Ginzburg and Landau (1950)] utilising a variable to describe the state of a superconductor media appeared. In this paper, instead of explaining the microscopic behaviour mechanism of superconduction, they set a somewhat abstract quantity and proposed an equation describing the evolution of the order parameter (the Ginzburg-Landau equation). Moreover, they provided a polynomial expansion for the free energy of the superconductor as a function of the aforementioned order parameter.

These kind of frameworks are often referred to as *phenomenological models*, as, in principle, it is nor fundamental for the order parameter to have a physical meaning or to explain the microscopic mechanism of the phenomenon accounted for by the order parameter; however it seems important to stress that:

i. recent advances [Gurtin (1996)] have shown that the Ginzburg-Landau equation can be obtained through a microforce balance, strengthening the physical content of such models;

ii. if the method of virtual power is applied to derive the framework [Del Piero (2009)], some considerations on the nature - and the meaning, thus - of the order parameter are needed in order to establish how it transforms due to a change of observer;

iii. every constitutive law is of *phenomenological nature* in essence;

iv. an order parameter may be used to describe a behaviour whose inner features, even if are known, cannot be observed at the scale at which the model is set.

In the case of phase transition models, the scale of the problem has to be pointed out (this is a critical point), as the physical meaning of some of the order parameters may differ and different evolution equations have to be accounted for.

2.2 Displacive order parameter

In order to describe a displacive phase transition, a common choice for the order parameter is [Shenoy et al. (1999)], [Albers et al. (2004)], [Ahluwalia et al. (2006)]:

$$e_2 = \frac{\varepsilon_{11} - \varepsilon_{22}}{\sqrt{2}},$$

being

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right) \tag{2.1}$$

the linearised strain tensor and

$$\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}, t), \quad \text{for } (\boldsymbol{x}, t) \in \Omega \times [0, \infty) \tag{2.2}$$

the displacement field.

Besides the order parameter, two other displacive quantities need to be introduced in order to complete the description of the strain at a given point; these are:

$$e_1 = \frac{\varepsilon_{11} + \varepsilon_{22}}{\sqrt{2}}$$

and

$$e_3 = \varepsilon_{12}.$$

The given definitions for the order parameter and the complementary displacive quantities may be confusing because of the presence of the coefficient $\frac{1}{\sqrt{2}}$ [1]; here the following

[1]the reason for such a choice lays in the fact that the Helmholtz free energy density for the linear elastic case takes the very compact form $\psi = (\lambda + \mu){e_1}^2 + \mu {e_2}^2 + \mu {e_3}^2$.

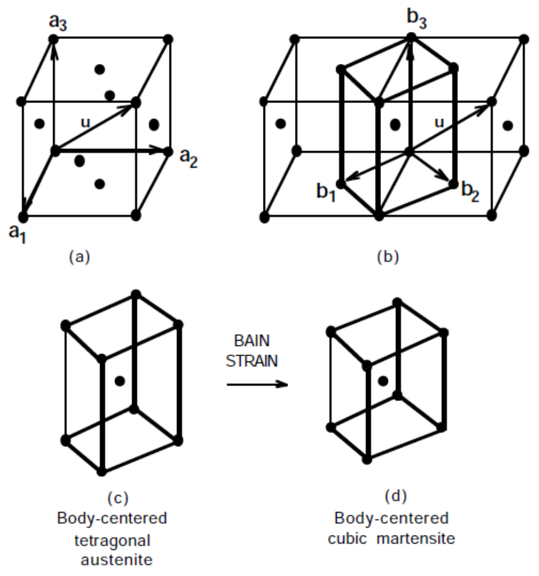

a3
u
a2
a1
(a)
b3
u
b1
b2
(b)
BAIN STRAIN
(c) Body-centered tetragonal austenite
(d) Body-centered cubic martensite

Figure 2.1: Bain strain (taken from Bhadeshia [Bhadeshia (1987)]).

positions are preferred [1]:

$$e_1 = \varepsilon_{11} + \varepsilon_{22}; \qquad e_2 = \varepsilon_{11} - \varepsilon_{22}; \qquad e_3 = \varepsilon_{12}. \tag{2.3}$$

It is important to notice that the quantities e_1, e_2 and e_3 are frame-dependent as the frame axes are meant to be coincident with the lattice axes ($\boldsymbol{b_1}$, $\boldsymbol{b_2}$ and $\boldsymbol{b_3}$ in fig. 2.1); therefore, the domain has to be constituted by a single grain or multiple grains having the same orientation.

The choice of the order parameter reflects the kinematic of the cubic-to-tetragonal martensitic phase transition [Lieberman et al. (1955)], [Bhadeshia (1987)]; its major feature is the *Bain strain*: a compression along two orthogonal axes and a dilatation along the third one. Figure 2.1 [Bhadeshia (1987)] clearly shows that the Bain strain can be described by a linearised strain tensor which is diagonal if (b_1, b_2, b_3) is assumed as the reference frame; in a two-dimensional setting the transition may be described as a square-to-rectangle deformation and calling α_1, α_2 the moduli of the principal strains we have:

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \alpha_1 & 0 \\ 0 & -\alpha_2 \end{bmatrix} \quad \Rightarrow \quad e_2 = \alpha_1 + \alpha_2; \quad e_1 = \alpha_1 - \alpha_2.$$

[1]the Helmholtz free energy density for the linear elastic case would be $\psi = \frac{\lambda+\mu}{2} {e_1}^2 + \frac{\mu}{2} {e_2}^2 + \mu {e_3}^2$, but this is of no importance, since the model proposed here cannot be referred to a linear elastic one.

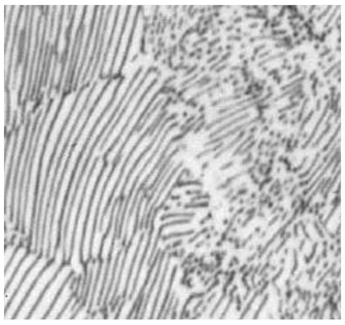


Figure 2.2: Optical microscope image of Pearlite.

The strains α_1 and α_2 are in general dependent on the temperature and on the chemical composition of the material [Kurdjumov and Kaminsky (1928)], [Xiao et al. (1995)].

2.3 Diffusive order parameter

An order parameter accounting for diffusive phenomena is set:

$$c = c(\boldsymbol{x}, t), \quad \text{for } (\boldsymbol{x}, t) \in \Omega \times [0, \infty). \tag{2.4}$$

The parameter describes the evolution from one phase to another one with different structure, and different inner order, thus; at the single-grain scale it may represent the density of solute divided by a reference density. This leads to a non-dimensional order parameter which can be used in a balance of mass equation, enforcing the physical meaning of the evolution equation for the diffusive part of the model.

In carbon steels, the solute is the carbon; Austenite can be identified by a null value of the diffusive order parameter ($c = 0$). The phase obtained through a diffusive phase transition is called Pearlite and appears as in figure 2.2. Pearlite is originated by the migration of the atoms of carbon; this process leads to a layered morphology in which it is possible to distinguish Cementite layers, i.e. zones where there is a great amount of carbon ($c > 0$) and ferrite layers, i.e. zones in which there is a lack of carbon ($c < 0$).

2.3.1 Diffusive Order Parameter Suitable For Macroscopic Scale

For *macroscopic scale* we mean a scale at which is no longer possible to distinguish the single grain. The physical meaning of the order parameter (and hence some of the equations of the model) has to change; for instance, the diffusive order parameter cannot represent the amount of carbon anymore, as no migration can be observed at this scale, and therefore needs to be a non-conserved one, identifying the transition from a phase to another with different inner structure.

Instead of using a Cahn-Hilliard equation as the evolution law (see Sect. 3.2.1), at this scale a Ginzburg-Landau equation seems more appropriate; the two scenarios, anyway, can still be obtained by means of same thermodynamic frame [Gurtin (1996)].

Chapter 3

Equations Of The Model And Thermodynamic Restrictions

3.1 Introduction

This chapter describes the equations used in the model; these are three balance equations: for solute mass, linear momentum and energy. The thermodynamic consistency of this set of equations will be investigated, as proper constitutive relations will be assumed, and a way to treat the terms accounting for the non-local effects will be proposed.

The set of the state variables is given by the following vector:

$$\Gamma = (\boldsymbol{\varepsilon}, c, T, \nabla\boldsymbol{\varepsilon}, \nabla c),$$

in which $\nabla\boldsymbol{\varepsilon}$ and ∇c account for non-locality effects and

$$\Gamma_{loc} = (\boldsymbol{\varepsilon}, c, T)$$

is the set of the local state variables.

3.2 Balance Equations

This section is devoted to the description of the balance equations used in the model, as well as to the proper distinction between external and internal powers; this is a critical point for non-local model with order parameters [Del Piero (2009)] and has to be carefully considered in order to correctly state the balance of energy.

3.2.1 Conservation of mass and chemical powers

We consider c to be a conserved order parameter, therefore:

$$\int_{\mathcal{S}_t} \dot{c}\, dw_t = -\int_{\partial\mathcal{S}_t} \boldsymbol{j}_c \cdot \boldsymbol{\nu}\, d\partial s_t = -\int_{\mathcal{S}_t} \nabla \cdot \boldsymbol{j}_c\, dw_t, \tag{3.1}$$

where $\boldsymbol{j}_c$ is the mass flux of c. As eq.(3.1) must hold for any sub-body $\mathcal{S}_t$, the mass balance equation can be expressed point-wise as:

$$\dot{c} = -\nabla \cdot \boldsymbol{j}_c. \tag{3.2}$$

In order to write the powers associated to the balance equation, the *chemical potential* μ, which is the variable conjugated to $\dot{c}$, has to be introduced; to account for the non-local nature of this thermodynamic force we consider the relation:

$$\mu = \mu_{loc}(\Gamma_{loc}) - \nabla \cdot \boldsymbol{M}(\Gamma), \tag{3.3}$$

in which the local part of the chemical potential is separated from the non-local one; in fact, $\nabla \cdot \boldsymbol{M}$ represents the part of the chemical potential at a given point due to the long-distance interaction.

Multiplying both terms in eq.(3.2) by the chemical potential and integrating over $\mathcal{S}_t$ we have:

$$\int_{\mathcal{S}_t} \mu\left(\dot{c} + \nabla \cdot \boldsymbol{j}_c\right) dw_t =$$
$$\int_{\mathcal{S}_t} \left[\mu_{loc}\dot{c} - \nabla \cdot (\boldsymbol{M}\dot{c}) + \boldsymbol{M} \cdot \nabla\dot{c} + \nabla \cdot (\mu\boldsymbol{j}_c) - \nabla\mu \cdot \boldsymbol{j}_c\right] dw_t = 0;$$

re-arranging the terms we obtain:

$$\int_{\mathcal{S}_t} \left(\mu_{loc}\dot{c} + \boldsymbol{M} \cdot \nabla\dot{c} - \nabla\mu \cdot \boldsymbol{j}_c\right) dw_t = \int_{\partial\mathcal{S}_t} \left(\boldsymbol{M}\dot{c} - \mu\boldsymbol{j}_c\right) \cdot \boldsymbol{\nu}\, ds_t. \tag{3.4}$$

The balance of chemical powers (eq.(3.4)) allows to identify the internal chemical power and the external chemical power, respectively:

$$\mathcal{P}^i_c(\mathcal{S}_t) = \int_{\mathcal{S}_t} \left(\mu_{loc}\dot{c} + \boldsymbol{M} \cdot \nabla\dot{c} - \nabla\mu \cdot \boldsymbol{j}_c\right) dw_t, \tag{3.5}$$

$$\mathcal{P}^e_c(\mathcal{S}_t) = \int_{\partial\mathcal{S}_t} \left(\boldsymbol{M}\dot{c} - \mu\boldsymbol{j}_c\right) \cdot \boldsymbol{\nu}\, ds_t. \tag{3.6}$$

Note that the internal chemical power sums up the contribution of three terms; the first and the second depend on the local and the non-local part of μ, respectively, and are related to the reversible part of the chemical process; the third one is dependent on $\nabla\mu$ and will give rise to the chemical dissipation.

3.2.2 Balance of momentum and mechanical powers

Balance of linear momentum reads:

$$\frac{d}{dt}\int_{\mathcal{S}_t} \rho\boldsymbol{v}\, dw_t = \int_{\partial\mathcal{S}_t} \boldsymbol{t}\, ds_t + \int_{\mathcal{S}_t} \boldsymbol{b}\, dw_t, \tag{3.7}$$

where ρ is the density, $\boldsymbol{v}$ is the velocity, $\boldsymbol{t}$ is a surface force term, $\boldsymbol{b}$ is a body force term. By means of the Noll's and the Cauchy's theorems it is possible to state that $\boldsymbol{t} = \boldsymbol{\sigma}\cdot\boldsymbol{\nu}$; hence, applying the divergence theorem we have:

$$\frac{d}{dt}\int_{\mathcal{S}_t} \rho\boldsymbol{v}\, dw_t = \int_{\mathcal{S}_t} \nabla\cdot\boldsymbol{\sigma}\, dw_t + \int_{\mathcal{S}_t} \boldsymbol{b}\, dw_t, \tag{3.8}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor; as eq.(3.8) must hold for any sub-body $\mathcal{S}_t$, we can write the balance equation in its point-wise formulation:

$$\rho\frac{\partial\boldsymbol{v}}{\partial t} = \nabla\cdot\boldsymbol{\sigma} + \boldsymbol{b}. \tag{3.9}$$

In order to write the powers associated to this balance equation, the local and the non-local parts of the stress tensor must be highlighted; this is done by assuming the following decomposition of the stress tensor:

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{loc}(\Gamma_{loc}) - \nabla\cdot\Sigma(\Gamma), \tag{3.10}$$

in which, again, $\nabla\cdot\boldsymbol{\Sigma}$ represents the part of the stress tensor at a given point due to the long-distance interaction; both $\boldsymbol{\sigma}_{loc}$ and $\nabla\cdot\boldsymbol{\Sigma}$ are supposed to be symmetric second order tensors.

Multiplying all the terms in the balance of linear momentum (eq.(3.9)) by $\boldsymbol{v}$ and integrating over $\mathcal{S}_t$, we have:

$$\begin{aligned}\frac{1}{2}\frac{d}{dt}\int_{\mathcal{S}_t} \rho\|\boldsymbol{v}\|^2\, dw_t &= \int_{\mathcal{S}_t} [\nabla\cdot(\boldsymbol{\sigma}_{loc} - \nabla\cdot\boldsymbol{\Sigma}) + \boldsymbol{b}]\cdot\boldsymbol{v}\, dw_t = \\ &+ \int_{\mathcal{S}_t} \left[-\left(\boldsymbol{\sigma}_{loc} : \dot{\boldsymbol{\varepsilon}} + \boldsymbol{\Sigma}\,\vdots\,\nabla\dot{\boldsymbol{\varepsilon}}\right) + \nabla\cdot(\boldsymbol{\sigma}\cdot\boldsymbol{v} + \boldsymbol{\Sigma} : \dot{\boldsymbol{\varepsilon}}) + \boldsymbol{b}\cdot\boldsymbol{v}\right] dw_t,\end{aligned}$$

having enforced the symmetry condition for the tensors and therefore substituted $\dot{\boldsymbol{\varepsilon}}$ with $\nabla \boldsymbol{v}$; splitting the local part of the stress tensor into two parts and re-arranging the terms we obtain:

$$\frac{1}{2}\frac{d}{dt}\int_{\mathcal{S}_t} \rho \|\boldsymbol{v}\|^2 \, dw_t + \int_{\mathcal{S}_t} \left(\boldsymbol{\sigma}^{el}_{loc} : \dot{\boldsymbol{\varepsilon}} + \boldsymbol{\sigma}^{an}_{loc} : \dot{\boldsymbol{\varepsilon}} + \boldsymbol{\Sigma} \vdots \nabla \dot{\boldsymbol{\varepsilon}} \right) dw_t = \int_{\mathcal{S}_t} \boldsymbol{b} \cdot \boldsymbol{v} \, dw_t + \int_{\partial \mathcal{S}_t} (\boldsymbol{\sigma} \cdot \boldsymbol{v} + \boldsymbol{\Sigma} : \dot{\boldsymbol{\varepsilon}}) \cdot \boldsymbol{\nu} \, ds_t. \quad (3.11)$$

The balance of mechanical energy (eq.(3.11)) allows to identify the kinetic energy, the internal mechanical power and the external mechanical power, respectively:

$$K(\mathcal{S}_t) = \int_{\mathcal{S}_t} \frac{1}{2} \rho \|\boldsymbol{v}\|^2 \, dw_t, \quad (3.12)$$

$$\mathcal{P}^i_m(\mathcal{S}_t) = \int_{\mathcal{S}_t} (\boldsymbol{\sigma}^{el}_{loc} : \dot{\boldsymbol{\varepsilon}} + \boldsymbol{\sigma}^{an}_{loc} : \dot{\boldsymbol{\varepsilon}} + \boldsymbol{\Sigma} \vdots \nabla \dot{\boldsymbol{\varepsilon}}) \, dw_t, \quad (3.13)$$

$$\mathcal{P}^e_m(\mathcal{S}_t) = \int_{\mathcal{S}_t} \boldsymbol{b} \cdot \boldsymbol{v} \, dw_t + \int_{\partial \mathcal{S}_t} (\boldsymbol{\sigma} \cdot \boldsymbol{v} + \boldsymbol{\Sigma} : \dot{\boldsymbol{\varepsilon}}) \cdot \boldsymbol{\nu} \, ds_t. \quad (3.14)$$

Note that the internal mechanical power sums up the contribution of three terms; the first and the third depend on, respectively, the local elastic part and the non-local part of $\boldsymbol{\sigma}$ and are related to the reversible part of the mechanical process; the third one is dependent on $\boldsymbol{\sigma}^{an}_{loc}$ - the *anelastic* contribution to the local part of the stress tensor - and will give rise to the mechanical dissipation.

3.2.3 Balance of energy

The first principle of thermodynamics, conservation of energy, requires that:

$$\frac{d}{dt}\int_{\mathcal{S}_t} \left(\frac{1}{2} \rho \|\boldsymbol{v}\|^2 + \rho u \right) dw_t = -\int_{\partial \mathcal{S}_t} \boldsymbol{q} \cdot \boldsymbol{\nu} \, ds_t + \int_{\mathcal{S}_t} \boldsymbol{b} \cdot \boldsymbol{v} \, dw_t + \int_{\partial \mathcal{S}_t} (\boldsymbol{\sigma} \cdot \boldsymbol{v} + \boldsymbol{\Sigma} : \dot{\boldsymbol{\varepsilon}}) \cdot \boldsymbol{\nu} \, ds_t + \int_{\partial \mathcal{S}_t} (\boldsymbol{M} \dot{c} - \mu \boldsymbol{j}_c) \cdot \boldsymbol{\nu} \, ds_t \quad (3.15)$$

where u is the specific internal energy (per unit volume) and $\boldsymbol{q}$ is the heat flux; inserting the mechanical power balance eq.(3.11) and the chemical power balance eq.(3.4) into eq.(3.15), we obtain:

$$\frac{d}{dt}\int_{\mathcal{S}_t} \rho u \, dw_t = -\int_{\mathcal{S}_t} \nabla \cdot \boldsymbol{q} \, dw_t + \int_{\mathcal{S}_t} (\boldsymbol{\sigma}^{el}_{loc} : \dot{\boldsymbol{\varepsilon}} + \boldsymbol{\Sigma} \vdots \nabla \dot{\boldsymbol{\varepsilon}}) \, dw_t + \int_{\mathcal{S}_t} (\mu_{loc} \dot{c} + \boldsymbol{M} \cdot \nabla \dot{c}) \, dw_t + \int_{\mathcal{S}_t} \boldsymbol{\sigma}^{an}_{loc} : \dot{\boldsymbol{\varepsilon}} \, dw_t - \int_{\mathcal{S}_t} \nabla \mu \cdot \boldsymbol{j}_c \, dw_t, \quad (3.16)$$

which is the energy balance equation. As eq.(3.16) must hold for any sub-body $\mathcal{S}_t$, we can write:

$$\rho \dot{u} = p^i_m + p^i_c - \nabla \cdot \boldsymbol{q}, \quad (3.17)$$

being:

$$p^i_m = \boldsymbol{\sigma}^{el}_{loc} : \dot{\boldsymbol{\varepsilon}} + \boldsymbol{\Sigma} \vdots \nabla \dot{\boldsymbol{\varepsilon}} + \boldsymbol{\sigma}^{an}_{loc} : \dot{\boldsymbol{\varepsilon}} \quad (3.18)$$

$$p^i_c = \mu_{loc} \dot{c} + \boldsymbol{M} \cdot \nabla \dot{c} - \nabla \mu \cdot \boldsymbol{j}_c \quad (3.19)$$

the specific mechanical and chemical internal powers, respectively.

3.3 Thermodynamic Restrictions For Admissible Processes: The Second Law Of Thermodynamics

In order to write the equations of the model in a definitive fashion, constitutive relations need to be assumed; these must be compatible with the principles of thermodynamics.

In this section we will derive the constitutive equations by exploiting the second law of thermodynamics in the form of the Clausius-Duhem equation [Lemaitre and Chaboche (1990)], [Maugin (1998)]. Here we do not include any *extra-entropy fluxes* [Maugin (1990)].

The second law of thermodynamics, the entropy inequality, requires that:

$$\int_{\mathcal{S}_t} \rho \dot{\eta} \, dw_t \geq -\int_{\partial \mathcal{S}_t} \frac{\boldsymbol{q} \cdot \boldsymbol{n}}{T} \, ds_t, \quad (3.20)$$

where η is the specific entropy. Applying the divergence theorem we obtain:

$$\int_{\mathcal{S}_t} \rho \dot{\eta} \, dw_t \geq -\int_{\mathcal{S}_t} \nabla \cdot \left(\frac{\boldsymbol{q}}{T} \right) dw_t;$$

as eq.(3.3) must hold for any sub-body $\mathcal{S}_t$, we can write the point-wise form of the Clausius-Duhem inequality:

$$\rho\dot{\eta} \geq -\nabla \cdot \frac{\boldsymbol{q}}{T}. \tag{3.21}$$

Assuming that the displacements and the displacements gradient are very small, no distinction is made between the Lagrangian and the Eulerian description (see Sect. 1.4); it is therefore possible to use the density of the media in the reference configuration and re-write the Clausius-Duhem inequality as follows [1]:

$$\dot{\eta} \geq -\nabla \cdot \frac{\boldsymbol{q}}{T}. \tag{3.22}$$

By using the first law of thermodynamics (eq.(3.17)) and introducing the *Helmholtz free energy* $\psi = u - T\eta$, the Clausius-Duhem inequality can be expressed in the more convenient form (*reduced inequality*):

$$\dot{\psi} + \dot{T}\eta - p_c^i - p_m^i \leq -\frac{\boldsymbol{q}}{T}\nabla T. \tag{3.23}$$

By substituting the expressions for the internal powers and remembering that ψ is a function of the variables $\Gamma = (\boldsymbol{\varepsilon}, c, T, \nabla\boldsymbol{\varepsilon}, \nabla c)$, the inequality becomes:

$$\begin{aligned}(\psi_T + \eta)\dot{T} + (\psi_c - \mu_{loc})\dot{c} + (\psi_{\nabla c} - \boldsymbol{M}) \cdot \nabla\dot{c} \\ + (\psi_{\boldsymbol{\varepsilon}} - \boldsymbol{\sigma}_{loc}^{el}) : \dot{\boldsymbol{\varepsilon}} + (\psi_{\nabla\boldsymbol{\varepsilon}} - \boldsymbol{\Sigma}) \vdots \nabla\dot{\boldsymbol{\varepsilon}} \\ \leq \boldsymbol{\sigma}_{loc}^{an} : \dot{\boldsymbol{\varepsilon}} - \nabla\mu \cdot \boldsymbol{j}_c - \frac{\boldsymbol{q}}{T} \cdot \nabla T,\end{aligned} \tag{3.24}$$

where the subscripts represent partial derivatives. This inequality has to be satisfied for all processes $(\dot{\boldsymbol{\varepsilon}}, \dot{c}, \dot{T}, \nabla\dot{\boldsymbol{\varepsilon}}, \nabla\dot{c})$.

Note that in eq.(3.24) the dissipations have been clearly separated from the parts of the stress tensor and of the chemical potential that are of non-dissipative nature.

[1] strictly operating, we should introduce the product of the reference density by the specific entropy as a new quantity, e.g. $\tilde{\eta} = \rho\eta$; however, in order to keep the notation light, we will not write $\tilde{\eta}$ but just η in the following. The same policy will be pursued for all the other quantities.

The second principle of thermodynamics is satisfied assuming:

$$\eta = -\psi_T, \tag{3.25}$$
$$\mu_{loc} = \psi_c, \tag{3.26}$$
$$\boldsymbol{M} = \psi_{\nabla c}, \tag{3.27}$$
$$\boldsymbol{\sigma}^{el}_{loc} = \psi_{\boldsymbol{\varepsilon}}, \tag{3.28}$$
$$\boldsymbol{\Sigma} = \psi_{\nabla\boldsymbol{\varepsilon}}, \tag{3.29}$$
$$\nabla\mu \cdot \boldsymbol{j}_c \leq 0, \tag{3.30}$$
$$\boldsymbol{\sigma}^{an}_{loc} : \dot{\boldsymbol{\varepsilon}} \geq 0 \tag{3.31}$$
$$\boldsymbol{q} \cdot \nabla T \leq 0. \tag{3.32}$$

Obviously, this by no means determines univocally the constitutive expressions; for our purposes it will be sufficient to assume the simplest choice which ensures the dissipations to be always positive:

$$\boldsymbol{j}_c = -\gamma\nabla\mu, \qquad \gamma(\Gamma) > 0, \tag{3.33}$$
$$\boldsymbol{\sigma}^{an}_{loc} = \beta\dot{e}_2\left(\boldsymbol{i}\otimes\boldsymbol{i} - \boldsymbol{j}\otimes\boldsymbol{j}\right), \qquad \beta(\Gamma) > 0 \tag{3.34}$$
$$\boldsymbol{q} = -\kappa\nabla T, \qquad \kappa(\Gamma) > 0. \tag{3.35}$$

Notice that the constitutive choice for $\boldsymbol{j}_c$ (together with a convenient form of the free energy) will cause the balance of solute mass to take the form of the standard Cahn-Hilliard equation; γ is called *mobility*.

The choice of $\boldsymbol{\sigma}^{an}_{loc}$ introduces a damping only for the deformation which characterises the displacive phase transition (i.e. a square-to-rectangle deformation), β being the damping coefficient.

For the heat flux we have chosen the Fourier's law, κ being the conductivity.

The resulting dissipations, namely *chemical*, *mechanical* and *thermal* [Maugin (1998)] are:

$$\Phi_c = \alpha\|\nabla\mu\|^2, \tag{3.36}$$
$$\Phi_m = \beta\dot{e}_2^2 \tag{3.37}$$
$$\Phi_t = \frac{1}{T}\kappa\|\nabla T\|^2. \tag{3.38}$$

We remark that the chemical potential and the elastic stress can be written accounting for their local and non-local parts in terms of functional derivatives of the Helmholtz free energy $\Psi = \int \psi \, dw$:

$$\mu = \psi_c - \nabla \cdot \psi_{\nabla c} \equiv \frac{\delta \Psi}{\delta c}, \qquad \boldsymbol{\sigma}^{el} = \psi_{\boldsymbol{\varepsilon}} - \nabla \cdot \psi_{\nabla \boldsymbol{\varepsilon}} \equiv \frac{\delta \Psi}{\delta \boldsymbol{\varepsilon}} \tag{3.39}$$

(this will be consistent with the choice of the boundary conditions made in Sect.3.5.2).

3.4 Helmholtz Free Energy

Central to our thermodynamic approach is the definition of the Helmholtz free energy. The total free energy of the system is given by:

$$\Psi = \int_{\mathbb{S}} \psi \, dw, \tag{3.40}$$

where ψ is the specific free-energy per unit mass.

The free energy is decomposed additively into four contributions, with each representing a particular physical aspect of the model, namely diffusive processes, displacive processes, interfacial energy, and coupling effects. Hence, we express ψ as:

$$\psi = \psi_{\text{diff}}\left(T, c, \nabla c\right) + \psi_{\text{displ}}\left(T, e_1, e_2, e_3, \nabla e_2\right) + \psi_{\text{therm}}\left(T\right) + \psi_{\text{cpl}}\left(c, e_2\right), \tag{3.41}$$

where:

- ψ_{diff} is the free energy associated with diffusive processes,
- ψ_{displ} is the free energy associated with displacive transformations and mechanical deformation,
- ψ_{therm} is the part of the free energy depending only on the temperature,
- ψ_{cpl} is the energy associated with the interaction of phases, typically the penalisation of the the formation of one in the presence of the other.

We provide explicit forms for each contribution to the free energy in the following.

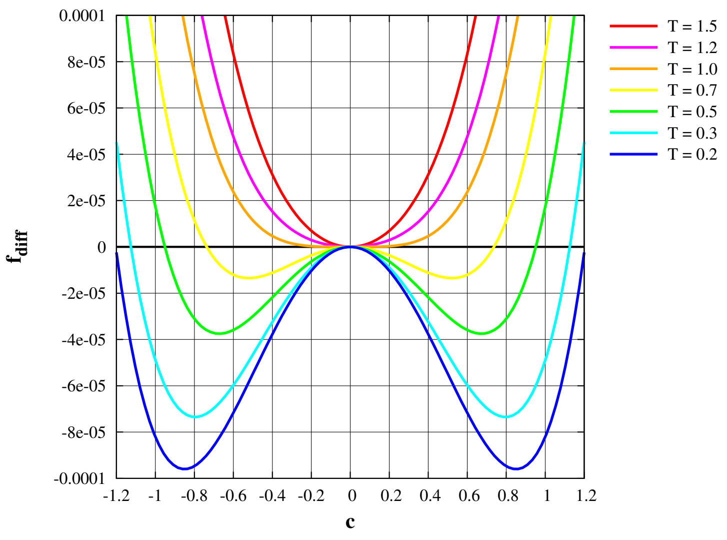


Figure 3.1: Diffusive free energy ($A = 7.3 \cdot 10^{-4}$, $B = 6.6 \cdot 10^{-4}$, $T_P = 1$)

3.4.1 Diffusive part of the free energy

We postulate the diffusive free energy functions of the form [Cahn (1961)], [Bouville and Ahluwalia (2007)]:

$$\psi_{\text{diff}} = \frac{A}{4}c^4 + \frac{B}{2}\frac{T - T_P}{T_P}c^2 + \frac{1}{2}K_c\|\nabla c\|^2, \tag{3.42}$$

where A, B and K_c are positive constants, and T_P is the temperature above which only Austenite is stable. The diffusive free energy as a function of c is shown in figure 3.1 for various temperatures T. At high temperature the free energy is convex, which indicates that there is only one stable phase, and this phase is associated with the minimum point $c = 0$. For a lower temperature, below T_P, the free energy function becomes non-convex and two wells appear, each corresponding to a stable phase. This is consistent with the formation of Pearlite - alternating layer with different carbon concentration.

3.4.2 Displacive part of the free energy

The displacive part of the free energy is postulated as follows:

$$\psi_{\text{displ}} = \frac{D}{6}e_2^6 - \frac{E}{4}e_2^4 + \frac{F}{2}\frac{T - T_M}{T_M}e_2^2 + \frac{G}{2}e_1\left[e_1 - 4\left(\alpha\left(T - T_{\text{ref}}\right) + x_{1c}c + x_{12}e_2^2\right)\right] + \frac{H}{2}e_3^2 + \frac{1}{2}K_e|\nabla e_2|^2, \quad (3.43)$$

where D, E, F, G, H and K_e are positive constants, x_{1c} and x_{12} are constant coupling parameters, T_M is the temperature below which Martensite becomes a stable phase, α is the thermal dilatation coefficient and T_{ref} is a reference temperature for thermal dilatation.

The displacive part of the free energy is illustrated in figure 3.2 as a function of e_2 ($e_1 = e_3 = 0$) for various temperatures and $T_{\text{ref}} = T$. At high temperature ($T > T_M$) only one phase is stable, which can be identified with the minimum at $e_2 = 0$. For $T < T_M$, another phase, Martensite, becomes stable, as the free energy shows two minima. The phase corresponding to $e_2 = 0$ is unstable, while the other one, corresponding to non-zero values of e_2, is stable.

3.4.3 Thermal part of the free energy

The thermal part of the free energy is assumed as follows:

$$\psi_{\text{therm}} = -\eta_0\left(T - T_{\text{ref}}\right) - \frac{c_p}{2T_{\text{ref}}}\left(T - T_{\text{ref}}\right)^2, \quad (3.44)$$

where c_p is the specific heat, η_0 is the specific entropy and T_{ref} is the reference temperature for thermal dilatation.

3.4.4 Phase-coupling part of the free energy

Finally, we consider a coupling contribution to the free energy. It serves to couple diffusive and deformation processes. We will consider a coupling free energy which is quadratic in both c and e_2:

$$f_{\text{cpl}} = x_{2c}c^2e_2^2, \quad (3.45)$$

where x_{2c} is a positive constant.

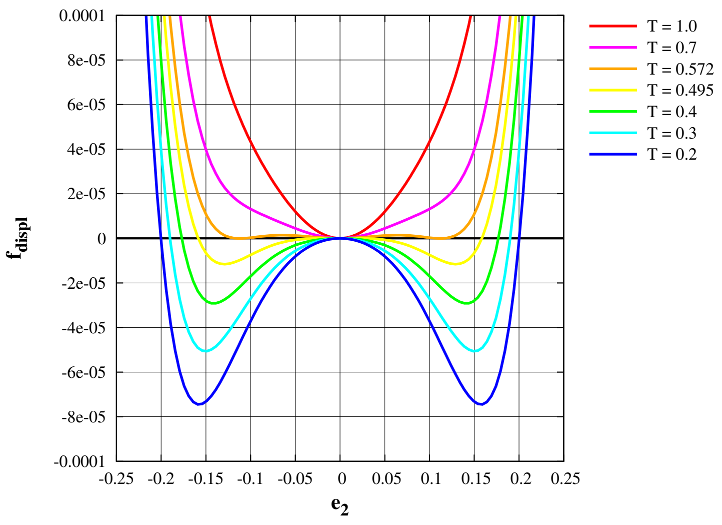


Figure 3.2: Displacive free energy as a function of e_2 only ($D = 29.5$, $E = 0.49$, $F = 9.9 \cdot 10^{-3}$, $x_{12} = 0$, $T_M = 0.495$).

3.5 Governing Equations

It is possible to substitute the constitutive relations eqs. (3.26) ÷ (3.29) and eqs. (3.33) ÷ (3.34) in the equation for the internal powers (eqs.(3.18) and (3.19)), obtaining the following specific internal powers:

$$p^i_m = \psi_{\boldsymbol{\varepsilon}} : \dot{\boldsymbol{\varepsilon}} + \psi_{\nabla\boldsymbol{\varepsilon}} \vdots \nabla\dot{\boldsymbol{\varepsilon}} + \beta \dot{e}_2^2 \tag{3.46}$$

$$p^i_c = \psi_c \dot{c} + \psi_{\nabla c} \cdot \nabla \dot{c} + \gamma |\nabla\mu|^2 \tag{3.47}$$

Besides, by using the constitutive relation for the specific entropy and for the heat flux (eqs.(3.25) and (3.35) respectively), we can rewrite eq.(3.17):

$$-T\dot{\psi_T} = \gamma|\nabla\mu|^2 + \beta \dot{e}_2^2 + \nabla \cdot (\kappa \nabla T). \tag{3.48}$$

If now we define:

$$c_s(T) = -T\psi_{TT} \tag{3.49}$$

$$c_{s,c} = \psi_{Tc} \tag{3.50}$$

$$c_{s,e1} = \psi_{Te_1} \tag{3.51}$$

$$c_{s,e2} = \psi_{Te_2} \tag{3.52}$$

eq.(3.48) takes the following form:

$$c_s(T)\dot{T} - T\left[c_{s,c}\dot{c} + c_{s,e2}\dot{e}_2 + c_{s,e1}\dot{e}_1\right] = \gamma|\nabla\mu|^2 + \beta\dot{e}_2^2 + \nabla\cdot(\kappa\nabla T), \tag{3.53}$$

which can be referred to as the *complete heat equation* and in which the thermo-chemical coupling, the thermo-mechanical coupling and the chemical, mechanical, thermal dissipations can be clearly noticed. We can observe that:

- both coupling terms and dissipations act as a heat source;
- dissipations are *always* a positive heat sources, and result in a temperature increase, thus (note that γ and β are requested to be always positive, see Sect.3.3);
- coupling terms can be positive or negative heat sources, depending on the process;
- for a process, in general, temperature will instantaneously increase or decrease depending on the concurrent effect of the dissipation and of the coupling terms.

We now summarise the equations of the model by collecting the three balance equations:

$$\begin{cases} \dot{c} = \nabla\cdot(\gamma\nabla\mu) \\ \rho\ddot{\boldsymbol{u}} = \nabla\cdot\boldsymbol{\sigma} + \boldsymbol{b} \\ c_s(T)\dot{T} - T\left[c_{s,c}\dot{c} + c_{s,e2}\dot{e}_2 + c_{s,e1}\dot{e}_1\right] = \gamma|\nabla\mu|^2 + \beta\dot{e}_2^2 + \nabla\cdot(\kappa\nabla T). \end{cases} \tag{3.54}$$

3.5.1 Explicit constitutive relations

The constitutive laws of stress and chemical potential can be put in an explicit form.

The stress tensor can be written as:

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^{el}_{loc} + \boldsymbol{\sigma}^{an}_{loc} - (K_e\nabla^2 e_2)\left(\boldsymbol{i}\otimes\boldsymbol{i} - \boldsymbol{j}\otimes\boldsymbol{j}\right), \tag{3.55}$$

with:

$$\begin{aligned} \boldsymbol{\sigma}^{el}_{loc} = &\left[D{e_2}^5 - E{e_2}^3 + \left(F\frac{T-T_M}{T_M} + 2x_{2c}c^2\right)e_2\right]\left(\boldsymbol{i}\otimes\boldsymbol{i} - \boldsymbol{j}\otimes\boldsymbol{j}\right) \\ &+ G\left[e_1 - 2\left[\alpha(T - T_{ref}) + x_{1c}c + x_{12}e_2^2\right]\right]\left(\boldsymbol{i}\otimes\boldsymbol{i} + \boldsymbol{j}\otimes\boldsymbol{j}\right) \\ &- 4Gx_{12}e_1e_2\left(\boldsymbol{i}\otimes\boldsymbol{i} - \boldsymbol{j}\otimes\boldsymbol{j}\right) + He_3\left(\boldsymbol{i}\otimes\boldsymbol{j} - \boldsymbol{j}\otimes\boldsymbol{i}\right), \end{aligned} \tag{3.56}$$

$$\boldsymbol{\sigma}_{loc}^{an} = \beta \dot{e}_2 \left(\boldsymbol{i} \otimes \boldsymbol{i} - \boldsymbol{j} \otimes \boldsymbol{j} \right). \tag{3.57}$$

The chemical potential is the following:

$$\mu = \mu_{loc} - K_c \nabla^2 c, \tag{3.58}$$

where:

$$\mu_{loc} = Ac^3 + B\frac{T - T_P}{T_P}c - 2Gx_{1c}e_1 + 2x_{2c}{e_2}^2 c. \tag{3.59}$$

The terms appearing in the heat equation are:

$$c_s(T) = c_p \frac{T}{T_{ref}} \tag{3.60}$$

$$c_{s,c} = \frac{B}{T_P}c \tag{3.61}$$

$$c_{s,e1} = -2G\alpha \tag{3.62}$$

$$c_{s,e2} = \frac{F}{T_M}e_2 \tag{3.63}$$

These expressions have to be substituted in eqs.(3.54).

3.5.2 Boundary and initial conditions for the differential system

Well-posed boundary and initial conditions are necessary to complement eqs.(3.54). To state the boundary conditions in a general mixed form, we assume that the boundary $\partial\Omega$ is partitioned into two subsets in two different ways:

$$\partial\Omega = \partial\Omega_s \cup \partial\Omega_u = \partial\Omega_q \cup \partial\Omega_T,$$

with:

$$\partial\Omega_s \cap \partial\Omega_u = \varnothing, \qquad \partial\Omega_q \cap \partial\Omega_T = \varnothing.$$

We assume the following boundary conditions:

$$\begin{cases} \nabla\mu(\boldsymbol{x},t)\cdot\boldsymbol{\nu}\Big|_{\partial\Omega} = 0 \\ \nabla c(\boldsymbol{x},t)\cdot\boldsymbol{\nu}\Big|_{\partial\Omega} = 0 \\ \nabla e_2(\boldsymbol{x},t)\cdot\boldsymbol{\nu}\Big|_{\partial\Omega} = 0 \\ \nabla T(\boldsymbol{x},t)\cdot\boldsymbol{\nu}\Big|_{\partial\Omega_q} = g(\boldsymbol{x}) \\ \boldsymbol{\sigma}_{loc}(\boldsymbol{x},t)\cdot\boldsymbol{\nu}\Big|_{\partial\Omega_s} = \boldsymbol{t}(\boldsymbol{x}) \\ \boldsymbol{u}(\boldsymbol{x},t)\Big|_{\partial\Omega_u} = \tilde{\boldsymbol{u}}(\boldsymbol{x}) \\ T(\boldsymbol{x},t)\Big|_{\partial\Omega_T} = \tilde{T}(\boldsymbol{x}). \end{cases} \tag{3.64}$$

We remark that condition $(3.64)_1$ ensures the global conservation of solute mass in the domain:

$$\frac{d}{dt}\int_\Omega c\,dw = -\int_{\partial\Omega}\boldsymbol{j}_c\cdot\boldsymbol{\nu}\,ds = \int_{\partial\Omega}\gamma\nabla\mu\cdot\boldsymbol{\nu}\,ds = 0. \tag{3.65}$$

The non-local boundary conditions (eqs.$(3.64)_2$, $(3.64)_3$) reflect the consideration that no long-distance interactions are allowed between the medium and its surrounding [Polizzotto (2003)]; hence, recalling eqs.(3.3) and (3.14) we assume:

$$\int_{\partial\Omega}\boldsymbol{M}\cdot\boldsymbol{\nu}\,ds = \int_{\partial\Omega}K_c\nabla c\cdot\boldsymbol{\nu}\,ds = 0 \tag{3.66}$$

$$\int_{\partial\Omega}(\boldsymbol{\Sigma}:\dot{\boldsymbol{\varepsilon}})\cdot\boldsymbol{\nu}\,ds = \int_{\partial\Omega}K_e\dot{e}_2\nabla e_2\cdot\boldsymbol{\nu}\,ds = 0, \tag{3.67}$$

which are satisfied if the conditions $(3.64)_2$ and $(3.64)_3$ are assumed.

The other conditions are classical.

The initial conditions are the following:

$$\begin{cases} \boldsymbol{u}(\boldsymbol{x},0) = \boldsymbol{u}_0(\boldsymbol{x}) \\ \dot{\boldsymbol{u}}(\boldsymbol{x},0) = \boldsymbol{v}_0(\boldsymbol{x}) \\ c(\boldsymbol{x},0) = c_0(\boldsymbol{x}) \\ T(\boldsymbol{x},0) = T_0(\boldsymbol{x}). \end{cases} \tag{3.68}$$

Chapter 4

Weak Form Of The Equations Of The Model

4.1 Introduction

This chapter is devoted to present the variational formulations of the equations of the model we derived in Chapter 3. The structure will be the same for each equation: first we will formulate the variational expression in the semi-discrete case, then we will treat the fully discrete case and last we will linearise the resulting form.

As the equations are of various nature, different techniques will be used; for the Cahn-Hilliard equation we will make use of the operator-splitting approach and treat the time derivative by means of the theta-method; for the balance of linear momentum we will propose a discontinuous-Galerkin formulation and obtain the fully discrete version of the equation through the generalised-alpha method; for the heat equation we will again enforce the theta-method to deal with the time derivatives.

The test and the trial functions resulting from the formulation must possess some properties that will be explained in Sect. 5.3.1, in which we will comment upon the code developed for the solution of the system of equations.

4.2 Cahn-Hilliard Equation

The Cahn-Hilliard equation is a parabolic PDE containing derivatives up to the fourth order; for this reason we consider an operator splitting approach for the derivatives in space, while the time derivative will be handled by means of the *theta-method*.

4.2.1 Semi-discrete case

We recall, for convenience, the Cahn-Hilliard equation:

$$\dot{c} = \nabla \cdot \left(\gamma \nabla \left(\psi_c - K_c \nabla^2 c\right)\right).$$

Implementing the operator splitting approach, we obtain the following two equations:

$$\dot{c} = \nabla \cdot (\gamma \nabla \mu), \tag{4.1}$$

$$\mu = \psi_c - K_c \nabla^2 c, \tag{4.2}$$

which are now of second order with respect to the derivatives in space.

Being (p, q) the test functions, the problem reduces to find (μ, c) such that:

$$\int_\Omega p\, \dot{c}\, dw + \int_\Omega \gamma \nabla p \cdot \nabla \mu\, dw - \int_{\partial\Omega} \gamma\, p \nabla \mu \cdot \boldsymbol{\nu}\, ds = 0$$
$$\int_\Omega q\, \mu\, dw - \int_\Omega q \psi_c\, dw - \int_\Omega K_c \nabla q \cdot \nabla c\, dw + \int_{\partial\Omega} K_c\, q \nabla c \cdot \boldsymbol{\nu}\, ds = 0.$$

Note that the boundary conditions that we require from the thermodynamics fall naturally out of the variational form; enforcing them we obtain:

$$\int_\Omega p\, \dot{c}\, dw + \int_\Omega \gamma \nabla p \cdot \nabla \mu\, dw = 0 \tag{4.3}$$

$$\int_\Omega q\, \mu\, dw - \int_\Omega q \psi_c\, dw - \int_\Omega K_c \nabla q \cdot \nabla c\, dw = 0 \tag{4.4}$$

4.2.2 Fully-discrete case

Using the theta-method to treat the time derivative, the two equations become:

$$\int_\Omega p\, \frac{c^{n+1} - c^n}{\Delta t}\, dw + \int_\Omega \nabla \gamma\, p \cdot \nabla \mu^{n+\theta}\, dw = 0 \tag{4.5}$$

$$\int_\Omega q\, \mu^{n+1}\, dw - \int_\Omega q\, {\psi_c}^{n+1}\, dw - \int_\Omega K_c \nabla q \cdot \nabla c^{n+1}\, dw = 0, \tag{4.6}$$

where the mobility is considered constant, Δt is the time step, $\cdot^i$ is a generic quantity evaluated at the step i and:

$$\mu^{n+\theta} = (1-\theta)\,\mu^{n+1} + \theta\mu^n.$$

Notice that the second equation is evaluated at the step $n+1$, as this will be the current step for our implementation (see Sect. 5.3.1).

The equations appear to be non-linear, due to the non-linearity of ψ_c; in a general fashion we can write:

$$\Lambda_C\left(\mu^{n+1}, c^{n+1}\right) = 0;$$

hence, the linearisation leads to the following equation to be solved:

$$\nabla_{(\mu^{n+1},c^{n+1})}\Lambda_C\left(\mu_0^{n+1}, c_0^{n+1}\right)\cdot\left(d\mu^{n+1}, dc^{n+1}\right) = -\Lambda_C\left(\mu_0^{n+1}, c_0^{n+1}\right), \tag{4.7}$$

where the left-hand side of eq.(4.7) is the bi-linear form, the right-hand side is the linear form of the problem, the vector $(d\mu^{n+1}, dc^{n+1})$ is the unknown and represents the increment in the solution for the step $n+1$, while $\left(\mu_0^{n+1}, c_0^{n+1}\right)$ is the solution (for the time step $n+1$) vector at the previous linear solver step.

4.3 Balance Of Linear Momentum Equation

For the model considered here, the balance of linear momentum yields to an equation containing fourth order derivatives in space and second order derivatives in time; however, an operator splitting approach leading to two second order partial derivatives [1] equation would not be feasible, as the natural boundary conditions would not fall naturally out of the formulation. Hence, we will formulate the problem through a discontinuous Galerkin approach for the non-local part of the stress tensor, while the local part will be handled in a standard fashion.

4.3.1 Semi-discrete case

We recall, for convenience, the balance of linear momentum equation:

$$\rho\ddot{\boldsymbol{u}} - \beta\nabla\cdot\left(\dot{e_2}\;\boldsymbol{\epsilon}_2\right) - \nabla\cdot\boldsymbol{\sigma}^{el}_{loc} + K_e\nabla\cdot\left(\nabla^2 e_2\;\boldsymbol{\epsilon}_2\right) - \boldsymbol{b} = 0,$$

[1] in space.

where:

$$\boldsymbol{\epsilon}_2 = \boldsymbol{i} \otimes \boldsymbol{i} - \boldsymbol{j} \otimes \boldsymbol{j}.$$

Being $\boldsymbol{r}$ the test function, the variational formulation reads:

$$\int_\Omega \rho\, \boldsymbol{r} \cdot \ddot{\boldsymbol{u}}\, dw + \int_\Omega \beta\, \dot{e_2} \left(r_{1,1} - r_{2,2}\right)\, dw + \int_\Omega \nabla \boldsymbol{r} : \boldsymbol{\sigma}^{el}_{loc}\, dw \\ - \int_\Omega K_e \nabla^2 e_2 \left(r_{1,1} - r_{2,2}\right)\, dw - \int_\Omega \boldsymbol{r} \cdot \boldsymbol{b}\, dw - \int_{\partial\Omega} \boldsymbol{r} \cdot \boldsymbol{t}\, ds = 0, \quad (4.8)$$

where:

$$\left(r_{1,1} - r_{2,2}\right) = \nabla \boldsymbol{r} : \boldsymbol{\epsilon}_2$$

and being $r_{i,j} = \partial_j r_i$.

Focussing now only on the integral representing the non-local part of the stress tensor and applying integration by parts, we have:

$$\int_\Omega K_e \nabla^2 e_2 \left(r_{1,1} - r_{2,2}\right)\, dw = - \int_\Omega K_e \nabla e_2 \cdot \nabla \left(r_{1,1} - r_{2,2}\right)\, dw,$$

having accounted for the non-local natural boundary condition $(3.64)_3$ and set, therefore,

$$\int_{\partial\Omega} K_e \left(r_{1,1} - r_{2,2}\right) \nabla e_2 \cdot \boldsymbol{\nu}\, ds = 0.$$

We can now apply a discontinuous Galerkin scheme [Ølgaard et al. (2008)] and write:

$$\int_\Omega K_e \nabla e_2 \cdot \nabla \left(r_{1,1} - r_{2,2}\right)\, dw - \int_{\tilde{\partial\Omega}} K_e \langle \nabla e_2 \rangle \cdot [\![\left(r_{1,1} - r_{2,2}\right)]\!]\, d\tilde{s} \\ - \int_{\tilde{\partial\Omega}} K_e [\![e_2]\!] \cdot \langle \nabla \left(r_{1,1} - r_{2,2}\right) \rangle\, d\tilde{s} + \int_{\tilde{\partial\Omega}} \frac{\xi}{h}\, K_e [\![e_2]\!] \cdot [\![\left(r_{1,1} - r_{2,2}\right)]\!]\, d\tilde{s}, \quad (4.9)$$

where: $[\![a]\!] = a^+ \boldsymbol{n}^+ + a^- \boldsymbol{n}^-$ is the jump operator, $\langle b \rangle = \left(b^+ + b^-\right)/2$ is the average operator, $\tilde{\partial\Omega}$ is the inner boundary, h is the mesh size and ξ is a stabilisation parameter.

We recall here that the second integral in the formulation (4.9) is needed for the consistency of the method, the third ensures the problem to have symmetry property and the fourth gives stability to the formulation.

Remembering that $e_2 = u_{1,1} - u_{2,2}$ and substituting the formulation (4.9) of the integral accounting the non-local stress into eq.(4.8), we obtain:

$$
\begin{aligned}
\int_\Omega \rho\, \boldsymbol{r} \cdot \ddot{\boldsymbol{u}}\, dw + \int_\Omega \beta \left(\dot{u}_{1,1} - \dot{u}_{2,2}\right)\left(r_{1,1} - r_{2,2}\right)\, dw + \int_\Omega \nabla \boldsymbol{r} : \boldsymbol{\sigma}^{el}_{loc}\, dw \\
+ \int_\Omega K_e \nabla \left(u_{1,1} - u_{2,2}\right) \cdot \nabla \left(r_{1,1} - r_{2,2}\right)\, dw \\
- \int_{\tilde{\partial\Omega}} K_e \langle \nabla \left(u_{1,1} - u_{2,2}\right) \rangle \cdot [\![(r_{1,1} - r_{2,2})]\!]\, d\tilde{s} \\
- \int_{\tilde{\partial\Omega}} K_e [\![(u_{1,1} - u_{2,2})]\!] \cdot \langle \nabla \left(r_{1,1} - r_{2,2}\right) \rangle\, d\tilde{s} \\
+ \int_{\tilde{\partial\Omega}} \frac{\xi}{h} K_e [\![(u_{1,1} - u_{2,2})]\!] \cdot [\![(r_{1,1} - r_{2,2})]\!]\, d\tilde{s} \\
- \int_\Omega \boldsymbol{r} \cdot \boldsymbol{b}\, dw - \int_{\partial\Omega} \boldsymbol{r} \cdot \boldsymbol{t}\, ds = 0. \quad (4.10)
\end{aligned}
$$

4.3.2 Fully-discrete case

In solving the balance of linear momentum equation it is of fundamental importance to eliminate - or at least reduce - high-frequency spurious modes, arising mainly from the finite-element spatial discretisation of the domain. For this reason, a *generalised-alpha* method has been applied; the method has some advantages: it is the one, amongst a variety of dissipative algorithms, that for given high-frequency dissipation has the lowest numerical dissipation at low frequencies; it does not exhibit any overshoots in displacements; it allows the user to control the amount of numerical dissipation introduced in the implementation [Chung and Hulbert (1993)], [Erlicher et al. (2002)].

The method consists in solving a modified balance equation in which the inertial term is evaluated at an intermediate point which is different from the intermediate point used for the evaluation of the stress and external force terms; for our purposes, the balance of linear momentum equation becomes:

$$
\begin{aligned}
\int_{\Omega} \rho \, \boldsymbol{r} \cdot \boldsymbol{a}^{\alpha_m} \, dw + \int_{\Omega} \beta \left(v_{1,1}^{\alpha_s} - v_{2,2}^{\alpha_s} \right) \left(r_{1,1} - r_{2,2} \right) \, dw & \\
+ \int_{\Omega} \nabla \boldsymbol{r} : \boldsymbol{\sigma}_{loc}^{el} \left(\boldsymbol{u}^{\alpha_s} \right) \, dw & \\
+ \int_{\Omega} K_e \nabla \left(u_{1,1}^{\alpha_s} - u_{2,2}^{\alpha_s} \right) \cdot \nabla \left(r_{1,1} - r_{2,2} \right) \, dw & \\
- \int_{\partial \tilde{\Omega}} K_e \langle \nabla \left(u_{1,1}^{\alpha_s} - u_{2,2}^{\alpha_s} \right) \rangle \cdot [\![\left(r_{1,1} - r_{2,2} \right)]\!] \, d\tilde{s} & \\
- \int_{\partial \tilde{\Omega}} K_e [\![\left(u_{1,1}^{\alpha_s} - u_{2,2}^{\alpha_s} \right)]\!] \cdot \langle \nabla \left(r_{1,1} - r_{2,2} \right) \rangle \, d\tilde{s} & \\
+ \int_{\partial \tilde{\Omega}} \frac{\xi}{h} \, K_e [\![\left(u_{1,1}^{\alpha_s} - u_{2,2}^{\alpha_s} \right)]\!] \cdot [\![\left(r_{1,1} - r_{2,2} \right)]\!] \, d\tilde{s} & \\
- \int_{\Omega} \boldsymbol{r} \cdot \boldsymbol{b}^{\alpha_s} \, dw - \int_{\partial \Omega} \boldsymbol{r} \cdot \boldsymbol{t}^{\alpha_s} \, ds = 0, & \quad (4.11)
\end{aligned}
$$

where:

$$
\begin{aligned}
\boldsymbol{a}^{\alpha_m} &= \left(1 - \alpha_m \right) \ddot{\boldsymbol{u}}^{n+1} + \alpha_m \, \ddot{\boldsymbol{u}}^{n} \\
\boldsymbol{v}^{\alpha_s} &= \left(1 - \alpha_s \right) \dot{\boldsymbol{u}}^{n+1} + \alpha_s \, \dot{\boldsymbol{u}}^{n} \\
\boldsymbol{u}^{\alpha_s} &= \left(1 - \alpha_s \right) \boldsymbol{u}^{n+1} + \alpha_s \, \boldsymbol{u}^{n} \\
\boldsymbol{b}^{\alpha_s} &= \left(1 - \alpha_s \right) \boldsymbol{b}^{n+1} + \alpha_s \, \boldsymbol{b}^{n} \\
\boldsymbol{t}^{\alpha_s} &= \left(1 - \alpha_s \right) \boldsymbol{t}^{n+1} + \alpha_s \, \boldsymbol{t}^{n}
\end{aligned}
$$

are the acceleration at the step $n + \alpha_m$ and the velocity, the displacement, the body force and the surface force at the step $n + \alpha_s$, respectively.

The Newmark approximation is used to evaluate the acceleration and the velocity at the step $n + 1$ [Erlicher et al. (2002)]:

$$
\boldsymbol{a}^{n+1} = \frac{1}{\beta_a \, \Delta t^2} \left[\boldsymbol{u}^{n+1} - \boldsymbol{u}^{n} - \Delta t \, \boldsymbol{v}^{n} - \left(\frac{1}{2} - \beta_a \right) \Delta t^2 \, \boldsymbol{a}^{n} \right] \quad (4.12)
$$

$$
\boldsymbol{v}^{n+1} = \boldsymbol{v}^{n} + \left(1 - \gamma_v \right) \Delta t \, \boldsymbol{a}^{n} + \gamma_v \, \Delta t \, \boldsymbol{a}^{n+1}; \quad (4.13)
$$

the only remaining unknown in eq.(4.11) is $\boldsymbol{u}^{n+1}$. Once $\boldsymbol{u}^{n+1}$ is calculated, $\boldsymbol{v}^{n+1}$ and $\boldsymbol{a}^{n+1}$ can be determined with eqs.(4.12), (4.13) and all the three quantities are given in input for the next time step.

Note that in order to correctly apply the proposed algorithm, the initial displacement and velocity have to be specified [1], while the initial acceleration needs to be evaluated solving eq.(4.11) with $\alpha_m = \alpha_s = 0$ [2].

In order to evaluate the elastic part of the stress tensor $\boldsymbol{\sigma}^{el}_{loc}\left(\boldsymbol{u}^{\alpha_s}\right)$ in eq.(4.11) different quadrature rules may be applied [3]; we have used the generalised mid-point rule:

$$\boldsymbol{\sigma}^{el}_{loc}\left(\boldsymbol{u}^{\alpha_s}\right) = \boldsymbol{\sigma}^{el}_{loc}\left(\left(1-\alpha_s\right)\boldsymbol{u}^{n+1} + \alpha_s\ \boldsymbol{u}^{n}\right).$$

A feature of the generalised-alpha method is the capability to control the numerical dissipation introduced; in fact, all the algorithm parameters (α_m, α_s, β_a, γ_v) can be expressed as functions of the spectral radius in the high frequency limit; as we will see in Sect.5.3.2, we have chosen values for the parameters which ensure stability and second order accuracy of the method in the case of *asymptotic annihilation*, i.e. posing the spectral radius equal to zero (this case is the opposite with respect to the no-dissipation case) [Chung and Hulbert (1993)].

Equation (4.11) is non-linear and in a general and compact fashion we can write:

$$\Lambda_M\left(\boldsymbol{u}^{n+1}\right) = 0;$$

hence, the linearisation leads to the following equation to be solved:

$$\nabla_{\boldsymbol{u}^{n+1}}\Lambda_M\left(\boldsymbol{u}_0^{n+1}\right)\cdot \boldsymbol{du}^{n+1} = -\,\Lambda_M\left(\boldsymbol{u}_0^{n+1}\right), \tag{4.14}$$

where the left-hand side of eq.(4.14) is the bi-linear form, the right-hand side is the linear form of the problem, the vector $\boldsymbol{du}^{n+1}$ is the unknown and represents the increment in the solution for the step $n+1$, while $\boldsymbol{u}_0^{n+1}$ is the solution (for the time step $n+1$) vector at the previous linear solver step.

4.4 Heat Equation

The heat equation results from the balance of energy; for the model proposed here, it is an equation containing second order derivatives in space and first order derivatives in time. To handle the latter, the theta-method will be applied.

[1]they are natural initial conditions indeed (see Sect. 3.5.2).

[2]as the current unknown becomes the acceleration, the equation is linear.

[3]note that all the possible rules differ because the elastic part of the stress tensor, for our choice of the free energy, is non-linear.

4.4.1 Semi-discrete case

We recall, for convenience, the heat equation:

$$c_p \, \frac{T}{T_{ref}} \, \dot{T} - T \left[\frac{B}{T_P} \, c \, \dot{c} + \frac{F}{T_M} \, e_2 \, \dot{e}_2 - 2 \, G \, \alpha \, \dot{e}_1 \right] - \alpha |\nabla \mu|^2 - \beta \, \dot{e}_2^2 - \nabla \cdot (\kappa \nabla T) = 0.$$

Being s the test function, the variational formulation reads:

$$\begin{aligned}\int_\Omega c_p \, \frac{T}{T_{ref}} \, s \, \dot{T} \, dw - \int_\Omega c_p \, s \, T \, \frac{B}{T_P} \, c \, \dot{c} \, dw \\ - \int_\Omega \frac{F}{T_M} \left(u_{1,1} - u_{2,2}\right) \left(\dot{u}_{1,1} - \dot{u}_{2,2}\right) \, dw \\ + \int_\Omega 2 \, G \, \alpha \left(\dot{u}_{1,1} + \dot{u}_{2,2}\right) \, dw \\ - \int_\Omega \alpha_s \nabla \mu \cdot \nabla \mu \, dw - \int_\Omega \beta \, s \left(\dot{u}_{1,1} - \dot{u}_{2,2}\right)^2 \, dw \\ + \int_\Omega \kappa \nabla q \cdot \nabla T \, dw - \int_{\partial\Omega} q \, g \, ds = 0. \quad (4.15)\end{aligned}$$

4.4.2 Fully-discrete case

Using the theta-method to treat the time derivative, the equation becomes:

$$\begin{aligned}\int_\Omega c_p \, \frac{T^{n+\theta}}{T_{ref}} \, s \, \frac{T^{n+1} - T^n}{\Delta t} \, dw - \int_\Omega c_p \, s \, T^{n+\theta} \, \frac{B}{T_P} \, c^{n+\theta} \, \frac{c^{n+1} - c^n}{\Delta t} \, dw \\ - \int_\Omega \frac{F}{T_M} \left(u_{1,1}^{n+\theta} - u_{2,2}^{n+\theta}\right) \left(\frac{u_{1,1}^{n+1} - u_{1,1}^n}{\Delta t} - \frac{u_{2,2}^{n+1} - u_{2,2}^n}{\Delta t} \right) \, dw \\ + \int_\Omega 2 \, G \, \alpha \left(\frac{u_{1,1}^{n+1} - u_{1,1}^n}{\Delta t} + \frac{u_{2,2}^{n+1} - u_{2,2}^n}{\Delta t} \right) \, dw \\ - \int_\Omega \alpha_s \nabla \mu^{n+\theta} \cdot \nabla \mu^{n+\theta} \, dw - \int_\Omega \beta \, s \left(\frac{u_{1,1}^{n+1} - u_{1,1}^n}{\Delta t} - \frac{u_{2,2}^{n+1} - u_{2,2}^n}{\Delta t} \right)^2 \, dw \\ + \int_\Omega \kappa \nabla q \cdot \nabla T^{n+\theta} \, dw - \int_{\partial\Omega} q \, g^{n+\theta} \, ds = 0. \quad (4.16)\end{aligned}$$

Note that all the time derivatives have been computed with the difference between the value at the time steps $n+1$ and at n, while the variables themselves have been evaluated at the step $n+\theta$.

Equation (4.16) is non-linear; in a general and compact fashion we can write:

$$\Lambda_H \left(T, \boldsymbol{u}, \mu, c\right) = 0;$$

hence, the linearisation leads to the following equation to be solved:

$$\begin{aligned}\nabla_{(T^{n+1},\boldsymbol{u}^{n+1},\mu^{n+1},c^{n+1})}\Lambda_H \left(T_0^{n+1}, \boldsymbol{u}_0^{n+1}, \mu_0^{n+1}, c_0^{n+1}\right) \cdot \\ \cdot \left(dT^{n+1}, \boldsymbol{du}^{n+1}, d\mu^{n+1}, dc^{n+1}\right) = \\ - \Lambda_H \left(T_0^{n+1}, \boldsymbol{u}_0^{n+1}, \mu_0^{n+1}, c_0^{n+1}\right), \quad (4.17)\end{aligned}$$

where the left-hand side of eq.(4.17) is the bi-linear form, the right-hand side is the linear form of the problem, the vector $(dT^{n+1}, \boldsymbol{du}^{n+1}, d\mu^{n+1}, dc^{n+1})$ is the unknown and represents the increment in the solution for the step $n+1$, while $\left(T_0^{n+1}, \boldsymbol{u}_0^{n+1}, \mu_0^{n+1}, c_0^{n+1}\right)$ is the solution (for the time step $n+1$) vector at the previous linear solver step.

Chapter 5

Computer Code For The Equations Of The Model

5.1 Introduction

This chapter explains how the fully-discrete, linearised forms of the equations presented in Chapter 4 have been coded. We used a collection of *C++* libraries that are part of the *FEniCS Project* which will be, for this reason, briefly explained in Sect. 5.2.

The interested reader can found further details about the *FEniCS project* in the work by Logg and Wells [Logg and Wells (2010)] and in the references contained therein; however, in order to clarify some concepts exposed here without forcing the reader to refer to the aforementioned paper, some of the figures of the work by Logg and Wells will be reported here (as explicitly stated in the related captions).

The solution of the equations of the model puts many computational issues. First of all, it is important to highlight that the diffusive and the displacive phenomena occur at two very different time scales; as stated in Sect. 1.1, displacive transitions occur in short time, while diffusive ones take very long time to exhibit; this fact cannot be ignored, particularly when the temperature is around (but below) the martensitic transition one, and the two phenomena are clearly concurrent, thus. Moreover, the variational formulations we proposed are both continuous (Cahn-Hilliard and heat equations) and discontinuous (balance of linear momentum equation), refer to equations of very different nature and the variables are many, namely $(\boldsymbol{u}, T, c, \mu)$.

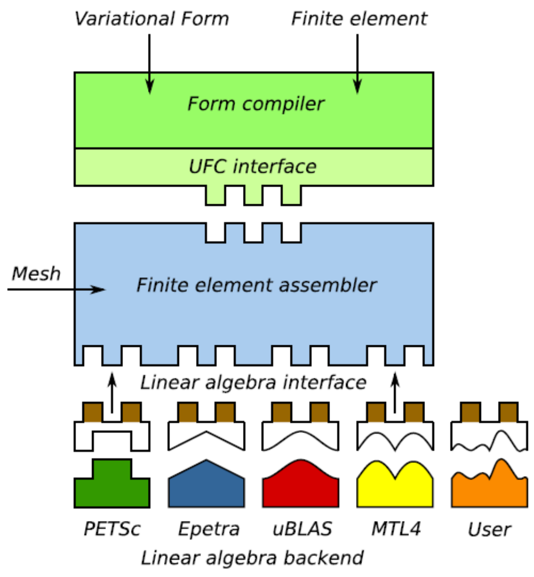


Figure 5.1: Dolfin relations with external components (taken from Logg and Wells [Logg and Wells (2010)]).

Therefore, the need for high-efficiency computational tools is urgent; however, special-purpose codes, despite being very efficient, typically lack flexibility, which makes their use for the solution of new problems not suitable. For this reason we decided to use *DOLFIN*, a library that encompasses computational efficiency, flexibility and a high level of expressiveness.

5.2 The *FEniCS* Project

The *FEniCS* Project represents an attempt to develop innovative tools for the automation of mathematical computation modelling; this aspect will be emphasised in the current section, while explaining some features of the components of the project.

DOLFIN is the main programming interface and the problem solving environment of the FEniCS project [Logg and Wells (2010)] and is a library for the automated solution of partial differential equations through the finite element method; DOLFIN relies on other tools (internal and external to the FEniCS project) in order to accomplish this mission. As depicted in fig.5.1, DOLFIN takes as input:

- a problem-specific code for computing the element vector and matrix
- a mesh
- linear algebra backend

and performs the assembly relying on highly-efficient algorithms, as this procedure can be executed in the same manner independently on the content of the above listed input elements (i.e. for different variational forms, element type, mesh, linear algebra backend).

The problem-specific code comes as C++ code containing a number of classes that can be instantiated by the user and passed to DOLFIN; for example, the bilinear and the linear forms of the problem are instantiated as objects of such classes and passed as input arguments to the assembly function. Such problem-specific code is passed through an interface called UFC (Unified Form-assembly Code) [Alnæs et al. (2009)]. DOLFIN can be fed with any UFC-compliant code, even if within the context of the FEniCS project some form compilers have been developed.

For the generation of the problem-specific code we used *FFC* (Fenics Form Compiler), a form compiler which automatically generates code compatible with UFC; it takes as inputs:

- the variational form of the problem
- the specification of the finite element to be used.

The finite element basis functions are evaluated by another FEniCS component, namely FIAT (FInite element Automatic Tabulator) [Kirby (2004)].

The principal advantages of using FFC are in that the form can be written in a nearly-mathematical notation and the automatic code generation makes the process less tedious and error-prone; the former argument brightly highlights the mission of automatic mathematical modelling of the FEniCS project and will be elucidated in Sect. 5.3.1.

In this way, very efficient code is automatically generated while the user has only to write the variational form of the problem in a very simple language (see Sect. 5.3.1).

The compilation of the variational form is done by calling the compiler with a line command, and must be performed prior to compiling the DOLFIN code.

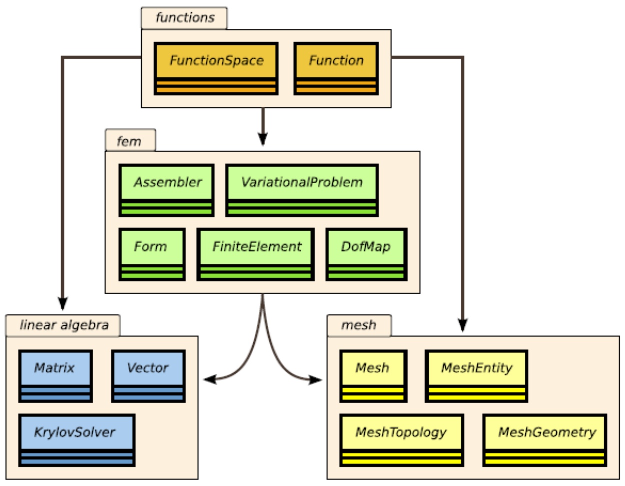


Figure 5.2: UML diagram for the central components and classes of DOLFIN (taken from Logg and Wells [Logg and Wells (2010)]).

As explained before, DOLFIN is a library containing a number of classes, which are all members of the C++ namespace *dolfin*; the most important ones are depicted in fig. 5.2 [Logg and Wells (2010)].

For the solution of linear systems, DOLFIN relies on external libraries (fig.5.1); hence, the related DOLFIN classes (e.g. *KrylovSolver*, fig. 5.2) are mostly wrappers for the linear algebra backends, allowing the user to easily set sensible options for the external libraries.

We remark that all FEniCS components are released under the GNU General Public License or the GNU Lesser General Public License and are freely available at

http://www.fenics.org.

5.3 Equations Coding In *DOLFIN*

Figure 5.3 shows a diagram of the files structure regarding the model code for DOLFIN; the functionality of each file will be briefly exposed in this introduction and only the most relevant features will be explained in detail throughout the current section.

For further details see Appendix A, where all the developed code is contained.

- *ThermoChemicalMechanical.ufl*, *ChemicalMechanical.ufl*, *ThermoMechanical.ufl*, *Mechanical.ufl*: FFC files containing the variational formulations for the complete

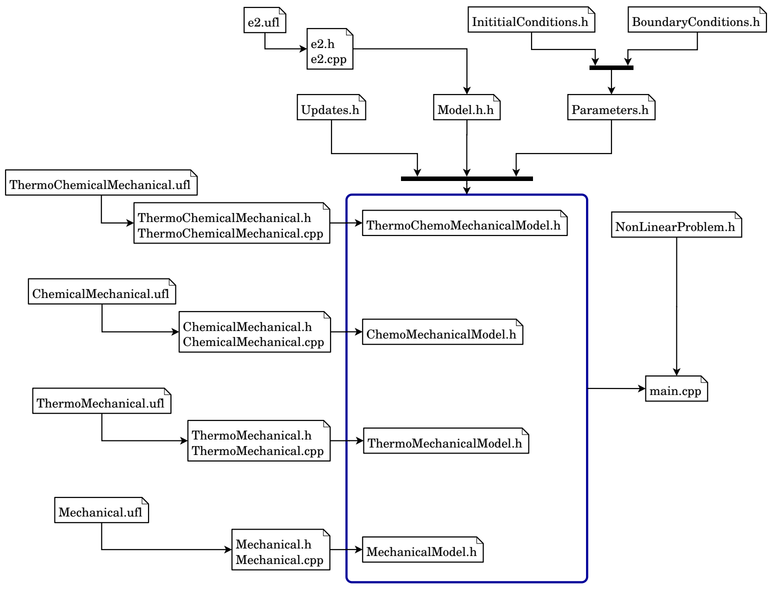


Figure 5.3: UML diagram for the files produced.

model (Cahn-Hilliard equation, balance of linear momentum equation and heat equation) and for reduced models (Sect. 5.3.1);

- *ThermoChemicalMechanical.h*, *ThermoChemicalMechanical.cpp*, *ChemicalMechanical.h*, *ChemicalMechanical.cpp*, *ThermoMechanical.h*, *ThermoMechanical.cpp*, *Mechanical.h*, *Mechanical.cpp*: FFC-generated files for the aforementioned variational formulations;

- *ThermoChemoMechanicalModel.h*, *ChemoMechanicalModel.h*, *MechanicalModel.h*, *ThermoMechanicalModel.h*: header files for the setting of the coefficients of the variational formulation, initial conditions, boundary conditions and for the acceleration and velocity updates. There is one file for each version of the model (and for each FFC file, thus);

- *Model.h*: header file for the definition of the general structure and features of each of the aforementioned files;

- *Parameters.h*, *InitialConditions.h*, *BoundaryConditions.h*: header files for the physical parameters of the model, the time stepping parameters and all the other parameters involved in the simulation, for the specification of the mesh, of the initial conditions and of the boundary conditions (Sects. 5.3.2, 5.3.3);

- *Updates.h*: header file containing the functions that perform the update of the acceleration and of the velocity after each time step (Sect. 5.3.4);
- *e2.ufl*, *e2.h*, *e2.cpp*: FFC and FFC-generated files for evaluating the displacive order parameter e_2 from the given displacements;
- *NonLinearProblem.h*: header file for the assembly of the residual vector and the Jacobian matrix of the linearised variational problem (Sect.5.3.5);
- *main.cpp*: main file, containing the time stepping loop and the commands for saving the results of the simulation.

5.3.1 Variational form

The variational form of the problem has been coded in FFC; first comes the definition of the finite element to be used:

```
P2v = VectorElement("Lagrange", "triangle", 2)
P1  = FiniteElement("Lagrange", "triangle", 1)
ME  = MixedElement([P2v, P1, P1, P1])
```

The syntax shows the capability of defining a mixed element made up of different elements, which can be either scalar or vectors; in the present case we use continuous Lagrange elements of second order for the vector element and of the first order for the scalar elements.

After this, the test function and the approximate solution function are defined in the following way:

```
# Displacement, temperature, concentration, chemical potential
(r,   q,   s,   y) = TestFunctions(ME)
dU = TrialFunction(ME)
U1 = Coefficient(ME)                # Current solution
```

```
U0 = Coefficient(ME)            # Previous converged step
du, dT, dc, dmu = split(dU)
u1, T1, c1, mu1 = split(U1)
u0, T0, c0, mu0 = split(U0)
```

The solution vector is build up with the functions related to the displacements (second order vector function), the temperature, the concentration and the chemical potential.

The displacements have to be computed using at least a second order function, as in the weak form of the balance of linear momentum (eq.(4.11)) second order derivatives persist in the discontinuous-Galerkin part of the formulation; notice that the displacements function is a continuous piece-wise polynomial of second order and its derivative is a discontinuous piece-wise polynomial of first order which has to be derived once more, justifying the discontinuous formulation.

The solution vector contains both the concentration and the chemical potential; this is due to the operator splitting approach used to solve the Cahn-Hilliard equation. In both cases, with reference to eq.(4.5) and eq.(4.6), the gradient of one function is a piecewise constant function and is projected onto a linear basis.

Notice that the command `split(.)` allows to operate with only a part of the complete vector of functions and that because of the non-local, time-dependent nature of the equations, the functions to account for are the solution at the previous converged time step, the current solution and its increment.

The following box contains an example of how parameters (either constants or functions) can be passed from outside FFC; this will enable simulations with different values of the parameters to be performed without having to re-compile the variational formulation code.

```
cT      = Constant(triangle)  # heat capacity
kT      = Constant(triangle)  # conductivity
T_ref   = Coefficient(P1)     # reference temperature for
                              # undeformed configuration
```

The drawback of passing parameters from outside consists on a higher compilation time; for this reason not all the simulation parameters are passed from outside FFC; for example, the coefficients of the local part of the free energy are defined inside the form file, as their values will not change:

```
D  = 29.535
E  = 0.49482
F  = 0.009948
G  = 0.004974
H  = 0.004974
TM = 0.495
```

FFC allows also to easily define functions in the following way:

```
# Shear strain (1/2)(dv/dx + du/dy)
def e3(r):
    return 0.5*(r[0].dx(1) + r[1].dx(0))
```

in which `r[i].dx(j)` corresponds to $\partial_{x_i} r_j(\boldsymbol{x})$.

The variational formulation is then coded basically using the commands `diff (.,.)` and `derivative (.,.,.)`. These are commands of major importance, that allow to write the code in the spirit of the automatic mathematical modelling: in Sect. 3.3 we gave constitutive equations in terms of functional derivatives of the free energy; the command `diff (.,.)` avoids these derivatives to be calculated by the user (a process which may introduce errors in the code), who has only to specify, as we did in Sect. 3, the free energy and write the constitutive relations in the form of its derivatives. With reference to the chemical potential:

```
# Chemical free energy
def f_diff(s, l, q):
    return (B/4.0)*s*s*s*s + A*(q-Tp)*s*s/(2.0*Tp)\
       + (lmbda/2.0)*inner(l, l)

# Chemical potential df/dc
mu_c    = diff(f_diff(c1, grad_c1, T1), c1) \
       + diff(f_cpl(c1, e2_1), c1) \
          + diff(f_disp(e1(u1), e2(u1), c1, T1), c1)
mu_grad = diff(f_diff(c1, grad_c1, T1), grad_c1)
```

being f_{cpl} and f_{disp} the coupling and the displacive part of the free energy (see App. A for details).

Once the constitutive relations are written in terms of derivatives of the free energy, we write the linear form and let FFC to compute the bilinear form (as the derivative of the linear form). With reference to the Cahn-Hilliard equation, linearisation of eqs.(4.5),(4.6) leads to eq.(4.7); in FFC we just write:

```
# Weak form
L_c_mass      =  s*c_rate*dx + Mob_T*inner(grad(s),grad(mu_mid))*dx
L_c_potential =  y*mu1*dx - y*mu_c*dx - inner(grad(y), mu_grad)*dx

L_mass = L_c_mass  + L_c_potential
a_mass = derivative(L_mass, U1, dU)
```

which is very compact and eliminates many sources of user-induced errors.

As can be seen, the variational formulation has been introduced in FFC using a near mathematical syntax and following a procedure that closely resembles the way in which we derived the model mathematically.

5.3.2 Parameters of the model and linearised system solver

The file Parameters.h contains the parameters for the model; the type of model, the linear solver and the Newton solver parameters are specified in the first part:

```
// Specify model type
std::string model_type("ThermoChemoMechanical");

// Create linear solver and set parameters
KrylovSolver linear_solver("bicgstab", "sor");
static bool   linearsolv_conv(true);
static double linearsolv_reltol = 1.0e-4;
```

```
// Create nonlinear solver and set parameters
DefaultFactory factory;
NewtonSolver newton_solver(linear_solver, factory);
static int    newton_maxit  = 10;
static double newton_reltol = 1e-6;
static double newton_abstol = 1e-12;
```

In the piece of code above, the complete model (balance of linear momentum equation, Cahn-Hilliard equation and heat equation) is chosen for the simulation through the string "ThermoChemoMechanical".

As mentioned in Sect. 5.2, DOLFIN allows to easily tune the value of some parameters of the linear algebra backend; in this case the type of solver, the tolerance and the monitoring of the convergence have been chosen; the linear solver for the linearised problem is an iterative one provided by the PETSc backend, namely the *biconjugate gradient stabilised* solver with an appropriate preconditioner. This makes the problem to be solved using an *inexact Newton* method, which is faster - in terms of time needed for the computation of one step - than an exact Newton method if the dimension of the problem is large [Dembo et al. (1982)].

The parameters for the Newton algorithm are set in the third part of the piece of code reported above: the type of linear solver is passed to the Newton solver, as well as the maximum number of iterations allowed and the relative and absolute tolerance. In Sect. 5.3.5 the class will be explained in more detail.

The values for the time step, for the generalised-alpha scheme and for the theta-method scheme are set with the following lines of the code:

```
// Time stepping parameters
static Constant dt(2.5e-6);      // time step
static double t = 0.0;           // initial simulation time
static double T = 10000*dt;      // total simulation time
static int save_step = 2;        // save results every # steps
static Constant theta(0.5);      // evaluation point
static Constant theta_m(-1.0);   // evaluation point g-alpha
static Constant theta_s(0.0);    // evaluation point g-alpha
static Constant beta(1.0);       // acceleration weight
static Constant gamma(1.5);      // velocity weight
```

The evaluation point chosen for the theta-method is the mid point ($\theta = 0.5$, see Sects. 4.2.2, 4.4.2); for the generalised-alpha scheme (see Sect. 4.3.2) the *asymptotic annihilation case* is enforced. As observed by Chung and Hulbert [Chung and Hulbert (1993)], with the generalised-alpha method the amount of numerical dissipation can be controlled by the user; in the case adopted here, the value of the spectral radius in the high frequency limit (ρ_∞) is set to zero and the sensible parameters of the scheme are determined by the following relations [Chung and Hulbert (1993)]:

$$\begin{aligned}
\alpha_m &= \frac{2\rho_\infty - 1}{\rho_\infty + 1} \\
\alpha_s &= \frac{\rho_\infty}{\rho_\infty + 1} \\
\beta_a &= \frac{1}{4}\left(1 - \alpha_m + \alpha_s\right)^2 \\
\gamma_v &= \frac{1}{2} - \alpha_m + \alpha_s
\end{aligned}$$

in order to ensure stability and second-order accuracy to the method in this condition.

The file Parameters.h, in its last part, contains some specifications for the number of initial conditions needed, the value for the Dirichlet boundary conditions and the specification of the part of boundary for the latter to be applied:

```
uint ic_dim = 5;                    // number of initial conditions

// Boundary conditions
std::vector<double> _zero_vector = boost::assign::list_of(0)(0);
static Constant zero_vector(0.0, 0.0);
static Constant zero_scalar(0.0);
static Left left;
static CentreLeft centre_left;
```

The values for the initial conditions and the classes describing the boundary are cast into separate files (see Sect. 5.3.3).

For the values of the physical parameters of the model the reader is referred to Sect. 6.2.3.

5.3.3 Initial conditions and boundary conditions

The initial conditions are specified in the file InitialConditions.h; the code consists of the definition of the class InitialConditions:

```
class InitialConditions : public Expression
{
public:

  InitialConditions(uint ic_dim)
       : Expression(ic_dim), engine(2), distribution(-0.01, 0.01),
         rng(engine, distribution)
  {
  }

  void eval(Array<double>& values, const Array<const double>& x) const
  {
    values[0]= 0.0001*rng();   // x displacement
    values[1]= 0.0001*rng();   // y displacement
    values[2]= 0.4;            // temperature
    values[3]= 0.001*rng();    // chemical concentration
    values[4]= 0.0;            // chemical potential
  }

  private:

    std::tr1::mt19937 engine;
    std::tr1::uniform_real<double> distribution;
    mutable std::tr1::variate_generator<std::tr1::mt19937, std::tr1::
        uniform_real<double> > rng;
};
```

The ease of specification of functions can be observed in the above code; DOLFIN already contains the class Expression, from which InitialConditions inherits the member eval for the assignment of the function values.

The Dirichlet boundary conditions are set by defining a portion of the boundary of the domain (file BoundaryConditions.h) making use of the member inside of the class SubDomain provided by DOLFIN:

```
class Left : public SubDomain
{
  bool inside(const double* x, bool on_boundary) const
  {
  if (x[0] < DOLFIN_EPS)
    return true;
  else
    return false;
  }
};
```

and then by defining an object of the class DirichletBC as follows:

```
Left left;
DirichletBC bc_u_1(Vu_x, zero_scalar, left);
```

where the arguments are the finite element sub-space, the value of the boundary condition and the boundary on which it has to hold.

5.3.4 Acceleration and velocity updates

The file Updates.h contains the functions for the update of the velocity and of the acceleration according to eqs.(4.12), (4.13); for example, the function defining how the acceleration is updated is the following:

```
  void update_a(Function& a, const Function& u, const Function& a0,
                const Function& v0,  const Function& u0,
                double beta, double dt)
  {
    // a = 1/(2*beta) * ((u-u0 - v0*dt)/(0.5*dt*dt) - (1-2*beta)*a0)
    a.vector()  = u.vector();
    a.vector() -= u0.vector();
    a.vector() *= 1.0/dt;
    a.vector() -= v0.vector();
    a.vector() *= 1.0/((0.5-beta)*dt);
    a.vector() -= a0.vector();
    a.vector() *= (0.5-beta)/beta;
  }
```

5.3.5 Non-linear solver

The DOLFIN class VariationalProblem already allows to solve non-linear problem with the Newton method; however, we had to code a user-defined class in order to achieve more flexibility regarding the non-linear solver. The code is the following:

```
class NonlinearProblem : public dolfin::NonlinearProblem
{
  public:

    // Constructor
    NonlinearProblem(const Form& a, const Form& L, std::vector<const
        DirichletBC*>& bcs,
                     const NewtonSolver& solver) : a(a), L(L), bcs(bcs),
                     solver(solver), reset_tensor(true)
    {
    }

    // Residual vector
    void F(GenericVector& b, const GenericVector& x)
    {
      // Assemble and modify for Dirichlet boundary conditions
      dolfin::Assembler::assemble(b, L);
      for (dolfin::uint i = 0; i < bcs.size(); i++)
        bcs[i]->apply(b, x);
    }

    // Jacobian
    void J(dolfin::GenericMatrix& A, const dolfin::GenericVector& x)
  {
    // Assemble and modify for Dirichlet boundary conditions
    if(solver.iteration() < 1)
    {
      dolfin::Assembler::assemble(A, a, reset_tensor);
      for (dolfin::uint i = 0; i < bcs.size(); i++)
        bcs[i]->apply(A);
    }
    reset_tensor  = false;

  }

 private:

  const Form& a;
  const Form& L;
            std::vector<const DirichletBC*>& bcs;
  const NewtonSolver& solver;
  bool reset_tensor;
};
```

Two members of the class have been defined: one (F) for the assembly of the residual vector given the linear form and the Dirichlet boundary conditions and one (J) for the assembly of the Jacobian matrix given the bi-linear form and the Dirichlet boundary conditions.

The creation of a user-defined class for the solution of the non-linear problem is justified by the *if* cycle in the class member J: it allows to switch between a Newton method and a *Quasi-Newton* method [Herceg et al. (1996)], [Ortega and Rheinboldt (1970)]; in fact, the assembly of the Jacobian matrix can be performed only for the first i iterations of the linear solver algorithm.

The adopted quasi-Newton method permits to considerably reduce the single-step computation time, especially for large dimension problems, where the the time needed to assemble the Jacobian matrix is a considerable amount of the total simulation time.

When this strategy is coupled with an iterative solver for the linearised system, the solution algorithm is said to belong to the *inexact quasi-Newton* family.

Chapter 6

Numerical Results

6.1 Introduction

This chapter is devoted to the comment of the results obtained through the numerical simulations; in order to put the solution of the equations in a more general framework and allow to compare the results obtained with what can be found experimentally, a preliminary section dealing with dimensional analysis is introduced; all the expressions for dimensionless coefficients will be derived in order to allow the comparison between the time and space scales resulting from the simulations and the real ones.

Before discussing the results, two major remarks need to be done:

- the time scale for diffusive and the thermal phenomena have been reduced to partly overcome the limits arising from our numerical implementation of the model (fixed time step, all equations solved at each time step; see Sect. 6.3).
- we did not implement any periodic boundary conditions; therefore simulations have to be considered as referred to a whole domain having the extension of a single grain, and not referred to a part of a larger domain (see Sect. 6.3).

6.2 Dimensional Analysis

In this subsection, dimensionless expressions for all the parameters to be used in the equations of the model will be derived. To this end, a dimensional basis will be chosen and the fundamental dimensions will be expressed in terms of the element of the basis.

This allows to compare results obtained by simulations with different values of the parameters and to compare the time and space scales of the model to the real ones, once real values for each element of the dimensional basis are known.

6.2.1 Dimensional basis and fundamental dimensions

The model proposed here encompasses chemical, mechanical and thermal phenomena; hence, the dimensional basis has to be constituted of four element. We will assume the following as the dimensional basis:

$$\{d, \rho, G, T_P\}\,, \tag{6.1}$$

where:

- d is a characteristic dimension of a single grain;
- ρ is the density;
- G is the coefficient of the hydrostatic part of the displacive free energy;
- T_P is the Pearlite transition temperature.

All the other parameters of the model can be made dimensionless by multiplying them by powers of the aforementioned element of the dimensional basis.

The fundamental dimensions, namely *length*, *time*, *mass* and *temperature* expressed in terms of the element of the dimensional basis are:

$$\tilde{L} = d; \quad \tilde{t} = \tfrac{\rho^{1/2}}{G^{1/2}} d; \quad \tilde{M} = \rho d^3; \quad \tilde{\theta} = T_P.$$

6.2.2 Dimensionless parameters

As stated in the previous subsection, dimensionless counterparts for all the parameters of the model can be obtained by multiplying the parameters by powers of the element of the dimensional basis; we obtain:

$$\Pi_{c_p} = \tfrac{c_p\ T_P}{G} \qquad \Pi_\kappa = \tfrac{\kappa\ \rho^{1/2}\ T_P}{d\ G^{3/2}} \qquad \Pi_{h_t} = \tfrac{h_T\ \rho^{1/2}\ T_P}{G^{3/2}} \qquad \Pi_\alpha = \alpha\ T_P$$

for thermal properties. Note that we have assumed the heat flux on the boundary (boundary condition eq.(3.64)$_4$) to be due to convective phenomena; this describes in a proper manner what happens in a heat treatment, in which the specimen is cooled by throwing it into a specific cooling fluid. The convective heat flux is the following:

$$g = h_t \left(T - T_{ext} \right), \tag{6.2}$$

where T_{ext} is the bulk temperature of the medium external to the body and h_t is the convective heat transfer coefficient, depending on the properties of both the cooling media and the material of the body.

The dimensionless parameters for the stress tensor are:

$$\Pi_D = \tfrac{D}{G} \qquad \Pi_E = \tfrac{E}{G} \qquad \Pi_F = \tfrac{F}{G} \qquad \Pi_H = \tfrac{H}{G}$$
$$\Pi_{x_{2c}} = \tfrac{x_{2c}}{G} \qquad \Pi_{K_e} = \tfrac{K_e}{d^2\ G} \qquad \Pi_\beta = \tfrac{\beta}{d\ \rho^{1/2}\ G^{1/2}};$$

note that x_{12} and x_{1c} are already dimensionless, as well as the displacive order parameter e_2 and the complementary displacive quantities e_1 and e_3.

The dimensionless temperatures are:

$$\Pi_{T_M} = \tfrac{T_M}{T_P} \qquad \Pi_{T_{ref}} = \tfrac{T_{ref}}{T_P} \qquad \Pi_{T_{ext}} = \tfrac{T_{ext}}{T_P}.$$

The dimensionless parameters for the chemical potential are:

$$\Pi_A = \tfrac{A}{G} \qquad \Pi_B = \tfrac{B}{G} \qquad \Pi_{K_c} = \tfrac{K_c}{d^2\ G};$$

note that the diffusive order parameter c is already dimensionless.

Regarding the mobility, a temperature-dependent function has been implemented for the simulations:

$$\gamma = \gamma_0 e^{-\frac{Q}{T}}, \tag{6.3}$$

where γ_0 is the mobility limit for $T \to \infty$ and Q is the activation energy. The dimensionless values for these quantities are:

$$\Pi_{\gamma_0} = \tfrac{\gamma_0\ G^{1/2}\ \rho^{1/2}}{d} \qquad \Pi_Q = \tfrac{Q}{T_P}.$$

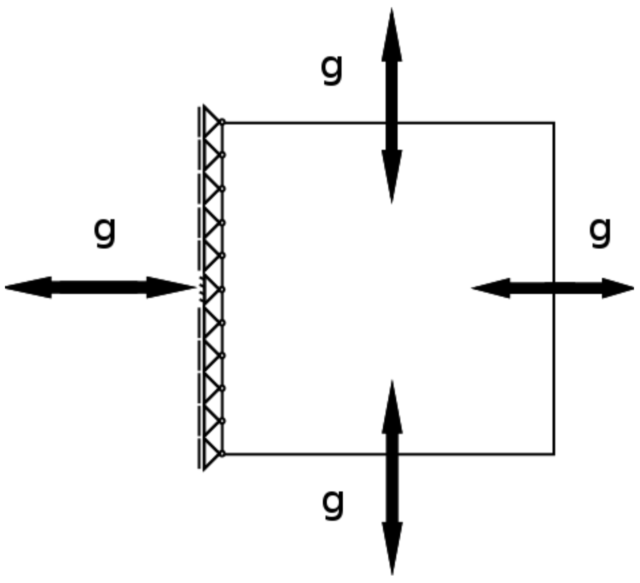


Figure 6.1: The unit squared domain with the applied boundary conditions.

6.2.3 Values of the Parameters

Simulations have been performed using the following values of the parameters:

$d = 1$	$D = 29.54$	$\beta = 5 \cdot 10^{-8}$
$\rho = 5 \cdot 10^{-7}$	$E = 4.95 \cdot 10^{-1}$	$T_M = 0.495$
$G = 4.97 \cdot 10^{-3}$	$F = 9.95 \cdot 10^{-3}$	$A = 7.31 \cdot 10^{-4}$
$T_P = 1$	$H = 4.97 \cdot 10^{-3}$	$B = 6.62 \cdot 10^{-4}$
$c_p = 1.35 \cdot 10^{-1}$	$x_{12} = 1.1$	$K_c = 1 \cdot 10^{-8}$
$\kappa = 1.78 \cdot 10^{-3}$	$x_{1c} = 0$	$\gamma_0 = 1.24 \cdot 10^{6}$
$h_t = 8.1 \cdot 10^{-5}$	$x_{2c} = 5$	$Q = 5$
$\alpha = 1.17 \cdot 10^{-2}$	$K_e = 1 \cdot 10^{-7}$	

6.3 Domain, Initial And Boundary Conditions

All the simulations have been performed on a squared domain of unitary edge length; hence, we suppose to simulate the behaviour of a single grain of material (see Sect. 2).

Different simulations have been carried out for different values of the initial temperature, in order to span the entire range form $T = 0.9$ (below the perlitic transition temperature T_P) to $T = 0.2$ (well below the martensitic transition temperature T_M).

The complete model, as has been numerically implemented, solves the given equations all at the same time. On the other hand experiments show that, displacive, diffusive and heat exchange phenomena run on very different time scales; the formers are extremely quick, while the other two take far more time to complete. This constitutes a limitation for the model in its current implementation: the maximum allowed time step is prescribed by the need for describing displacive phenomena.

In modelling a heat treatment in its cooling phase, the initial temperature should be above T_P and a heat flux of the kind of eq.(6.2) should be applied at the boundary; T_{ext} should be the bulk temperature of the cooling media and h_t its convective heat transfer coefficient. However, due to the fact that the time step is fixed and that all the equations are solved at each time step, we couldn't simulate the described conditions, because the computational time needed for any point of the domain to reach a temperature below T_P or even T_M would not have been affordable; we decided therefore to start from an initial temperature already below the transition temperature. This is a strong limitation, however:

- it has been possible to get the main physical aspects of the model, except for the influence of the term $\alpha\,(T - T_{ref})$ in triggering the displacive phase transition;

- it has been possible to solve many of the computational issues arising from the solution of the three equations which constitutes the model;

- time adaptivity and other numerical techniques implementation is planned as a future activity.

As a result, the initial temperature was varying from one simulated case to another, while the initial displacements and the initial value of the diffusive order parameter were set to a random small value; this simulates a small perturbation in the equilibrium state; if such equilibrium is unstable, the transition will occur. Indicating as $rnd(i)$ a random function with values contained in the interval $[-i\ ,\ i]$ and remembering that the initial value of the chemical potential is set to zero, the initial conditions were:

$$\begin{cases} c_0 = rnd\,(10^{-5}) \\ \mu_0 = 0 \\ \boldsymbol{u}_0 = rnd\,(10^{-6}) \\ T_0 = \tilde{T}_{ext} \end{cases};$$

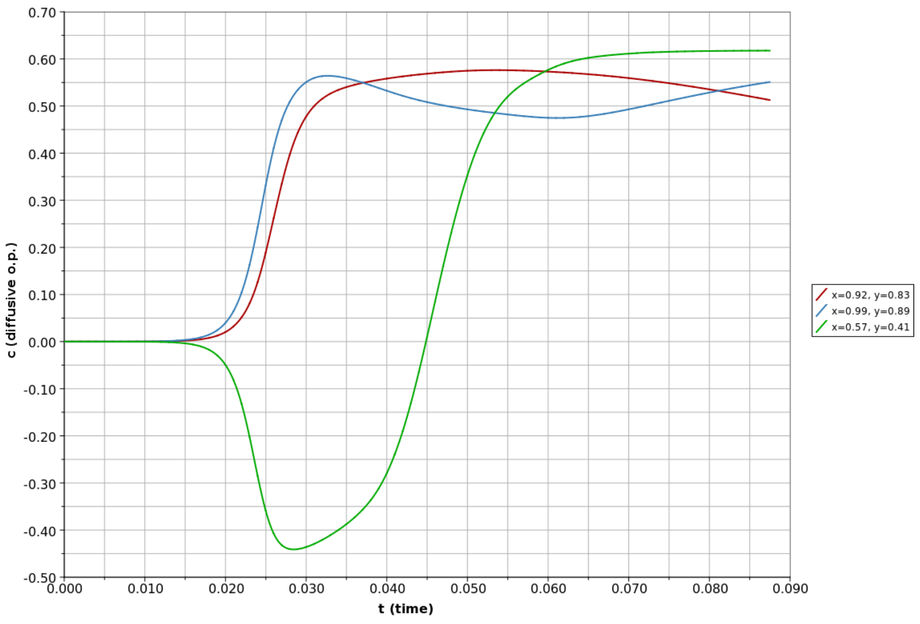


Figure 6.2: Diffusive o.p. evolution at three different points of the domain; $T_{ext} = 0.6$.

note that the initial temperature is equal to the bulk temperature of the cooling media (also the reference temperature for the undeformed configuration T_{ref} was set equal to T_{ext}); this corresponds to having a body of the dimension of a single grain, at $t = 0$, all at the same temperature of the cooling media and in the austenitic phase (removing completely the effect of the initial cooling and the related thermo-elastic effect, however).

The boundary conditions, as described in Sect. 3.5.2, were:

- of insulation type for the diffusive order parameter and for the long-distance interactions;
- Neumann condition (eq.(6.2)) throughout the boundary for the temperature (see fig. 6.1);
- Dirichlet boundary conditions at left edge for the displacements (see fig. 6.1).

6.4 Features Of A Diffusive Phase Transition

Figure 6.2 depicts the evolution of the diffusive order parameter (as a function of time) in a diffusive phase transition; the simulation was performed with the following parameters[1]:

· mesh: triangular elements; 128×128 elements;

· time step: $dt = 5 \cdot 10^{-6}$;

· initial, cooling media bulk and reference temperature: $\tilde{T}_{ext} = 0.6$.

The evolution is shown at three different points of the domain; the presence of the only first-order time derivative in the Cahn-Hilliard equation gives the curves the typical *sigmoidal* behaviour, with the order parameter that do not oscillate and tends, after a long time, to reach the stable value.

However, sometimes the value of the order parameter may significantly change (see for instance the green curve of fig. 6.2), passing from a negative value to a positive one (or vice-versa); this is due to the capability of the interfaces to move and the tendency of the phases to coarsen, in order to reach a more favourable configuration from an energy point of view.

In figure 6.15, the temperature evolution is related to the change in the order parameter value for two different points of the domain; the changes in temperature up to $t \simeq 0.065$ are driven by the thermo-chemical coupling term and the dissipation term in the heat equation, which act as heat sources when a diffusive process occur ($\dot{c} \neq 0$). A reduced form of the heat equation, obtained disregarding the heat conduction term and the others related to displacive phenomena, may help in highlighting this concept:

$$c_s(T)\dot{T} = T\ \frac{B}{T_P}\ c\ \dot{c} + \gamma|\nabla\mu|^2. \tag{6.4}$$

As a result, temperature may increase, or even decrease, accordingly to the observations stated in Sect. 3.5 (see also Sect. 6.5). On the other hand, changes in temperature after $t \simeq 0.065$ are mainly due to heat conduction. In fact, the diffusive order parameter does not change significantly; this excludes any effects of the aforementioned thermo-chemical coupling term in changing the temperature; moreover, the observed variation

[1]we report here only the parameters whose values change from one test-case to another; for the values of all the other see Sect. 6.2.3.

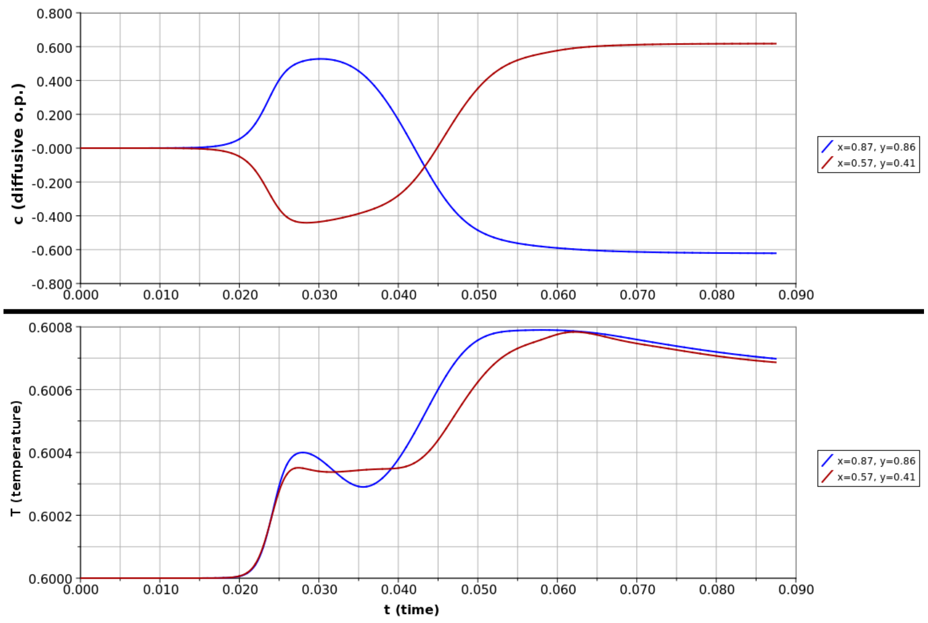


Figure 6.3: Diffusive o.p. and temperature evolution at two different points of the domain; $T_{ext} = 0.6$.

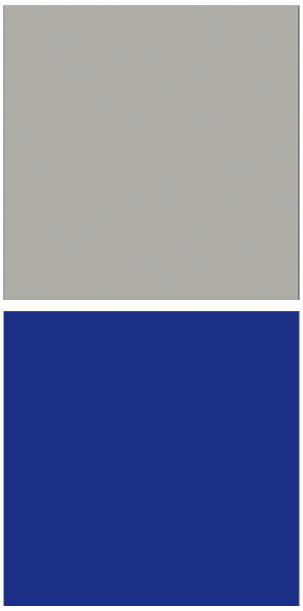
cannot be due to the dissipation term, as this can only cause an increase in temperature (see Sect. 3.5).

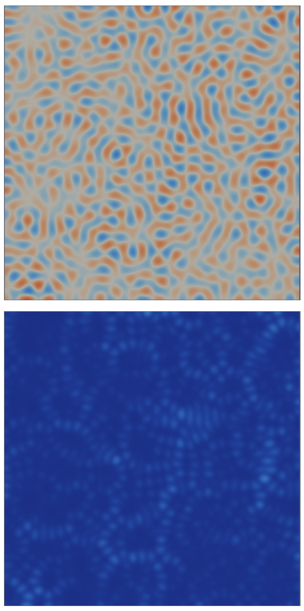
In figure 6.4, the evolution of the whole domain is depicted; only the temperature and the diffusive order parameter are represented, as the displacive order parameter, being the temperature in the domain above T_M, remains null[1]. Note that the transition starts almost everywhere at the same time, as no direct chemo-mechanical effects are present ($x_{1c} = 0$), and that phases tend to form a right angle with the edges of the boundary, as this configuration is energetically more favourable. After a long time, a little coarsening can be observed.

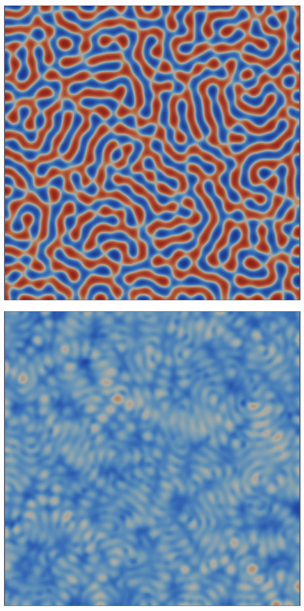
Temperature changes accordingly: there is a general increase due to the heat generated by the diffusive phase transition and after a long time heat conduction smothers the temperature difference between the regions of the domain.

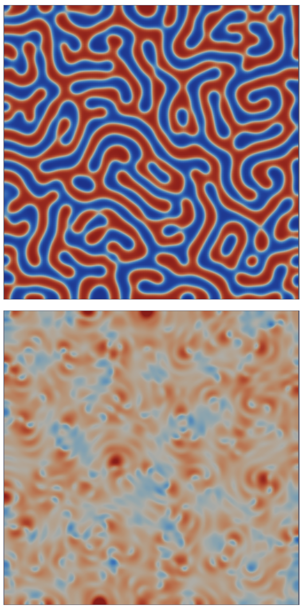
The width of the bands can be controlled by tuning the value of the gradient coefficient K_c, which plays a relevant role on the transition onset time as well: a high value of K_c

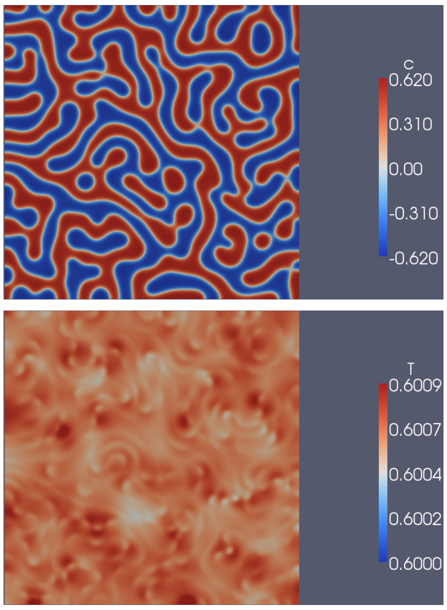
[1]i.e. its value changes, but remains between $\pm 10^{-6}$, the value of the initial random.


(a) $t = 0.010$


(b) $t = 0.023$


(c) $t = 0.030$


(d) $t = 0.050$



(e) $t = 0.080$

Figure 6.4: Diffusive o.p and temperature evolution throughout the domain; $T_{ext} = 0.6$.

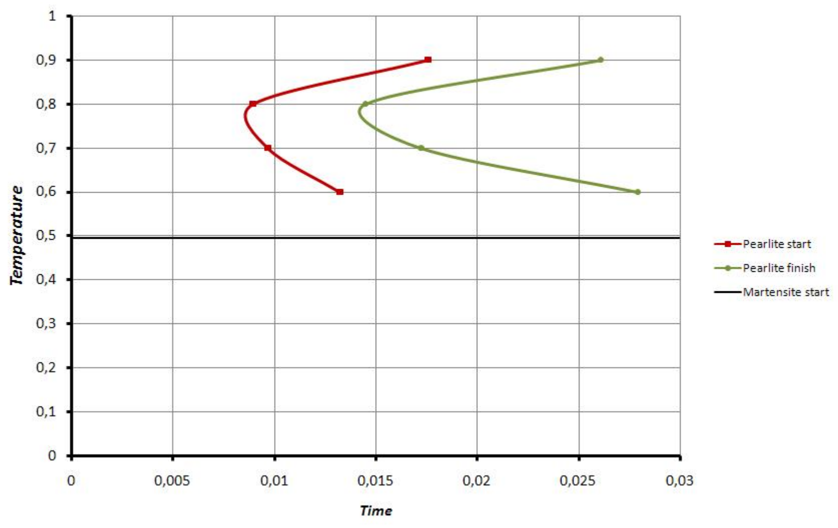


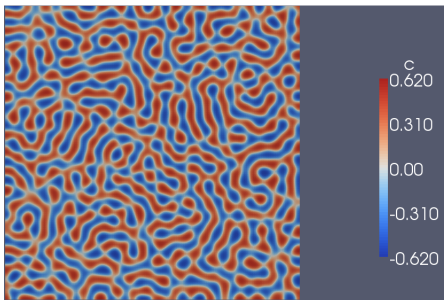
Figure 6.5: Onset and end curves for diffusive phase transition.

will cause the phase to be coarser from early stages of formation and will increase the nucleation time, because it will increase the energy related to the interface construction.

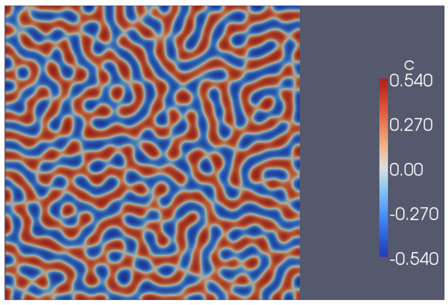
6.4.1 Diffusive phase transition at various temperatures

Figure 6.5 shows the effect of the concurrent interaction between the temperature-dependent mobility and the undercooling. For each test-case (from $T_{ext} = 1$ to $T_{ext} = T_M = 0.495$) two measurements of time have been conducted; one when the value of the diffusive o.p. at any point of the domain was at 5% of the stable value, and another when it was at 95% (note that, for the choice of the Helmholtz free energy we made, the *stable value* changes with temperature, see Sect. 3.4); the measurements have been collected and the resulting plot shows the typical behaviour of the Bain curves: at high temperature the *undercooling*, i.e. the difference between the temperature at a given point of the domain and the perlitic transition temperature, is small and the transition takes a long time to start; at low temperature (but above T_M, black line in fig. 6.5) the undercooling is relevant, but the mobility is low, eq.(6.3); hence, the transition is again inhibited and takes long time even to get to completion. At an intermediate temperature the optimum condition can be reached, and the transition is the quickest to start and complete.

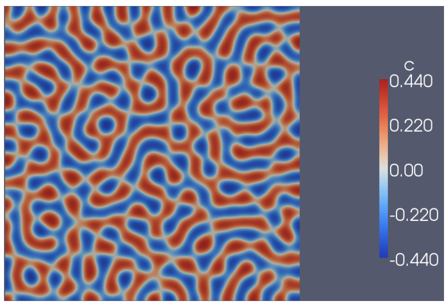
Figure 6.6 depicts a comparison between the morphology of the diffusive phase ob-



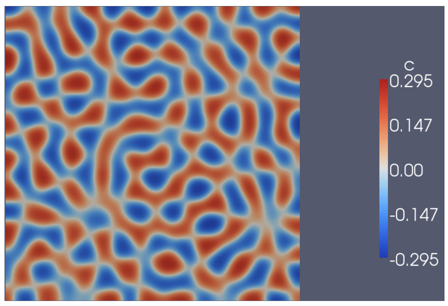
(a) $\tilde{T}_{ref} = 0.6$



(b) $\tilde{T}_{ref} = 0.7$



(c) $\tilde{T}_{ref} = 0.8$



(d) $\tilde{T}_{ref} = 0.9$

Figure 6.6: Morphology of the diffusive phase for different values of T_{ext}.

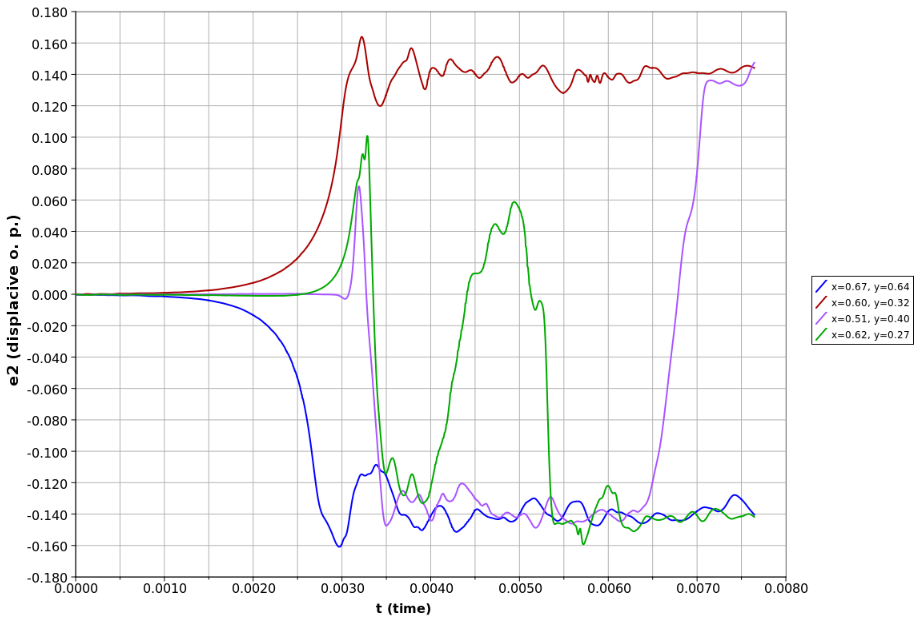


Figure 6.7: Displacive o.p. evolution at four different points of the domain; $T_{ext} = 0.4$.

tained for different values of $\tilde{T}_{ext}$; the images refer, for each test, to the time marked by the *Pearlite finish* curve of fig. 6.5 and are scaled to the stable value proper of the considered temperature, in order to set the attention on the morphology only and to highlight that at high temperature a coarse phase is obtained, while the structure becomes finer (thinner bands) as temperature decreases.

6.5 Features Of A Displacive Phase Transition

Figure 6.7 depicts the evolution of the displacive order parameter (as a function of time) in a displacive phase transition; the simulation was performed with the following parameters[1]:

· mesh: triangular elements; 196×196 elements;

· time step: $dt = 2.5 \cdot 10^{-6}$;

[1]we report here only the parameters whose values change from one test-case to another; for the values of all the other see Sect. 6.2.3.

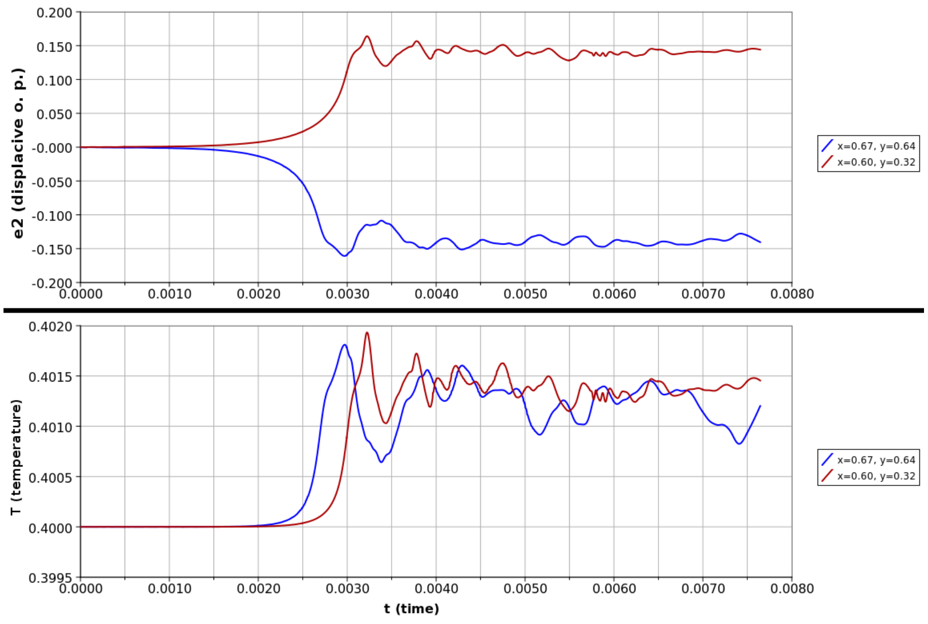


Figure 6.8: Displacive o.p. and temperature evolution at two different points of the domain; $T_{ext} = 0.4$.

· initial, cooling media bulk and reference temperature: $\tilde{T}_{ext} = 0.4$.

The evolution is shown at four different points of the domain; note that the time scale is one order of magnitude smaller with respect to the previous case (Sect. 6.4). As the balance of linear momentum contains also a second-order time derivative (*inertial*) term, the order parameter vibrates around the stable value; vibrations tend to be damped due to the presence of the dissipative term depending on β (see eq.$(3.54)_2$).

However, due to the moving interfaces and the tendency of the phases to coarsen, the value of the order parameter may significantly change (see for instance the violet and the green curves of fig. 6.7), passing from a negative value to a positive one (or vice-versa). Moreover, fig. 6.7 clearly shows that the phase transition may begin at very different times at different points of the domain; this behaviour will be explained later in this section (see fig. 6.10).

In figure 6.8, the temperature evolution is related to the change in the order parameter value at two different points of the domain; as diffusive phenomena cannot take place

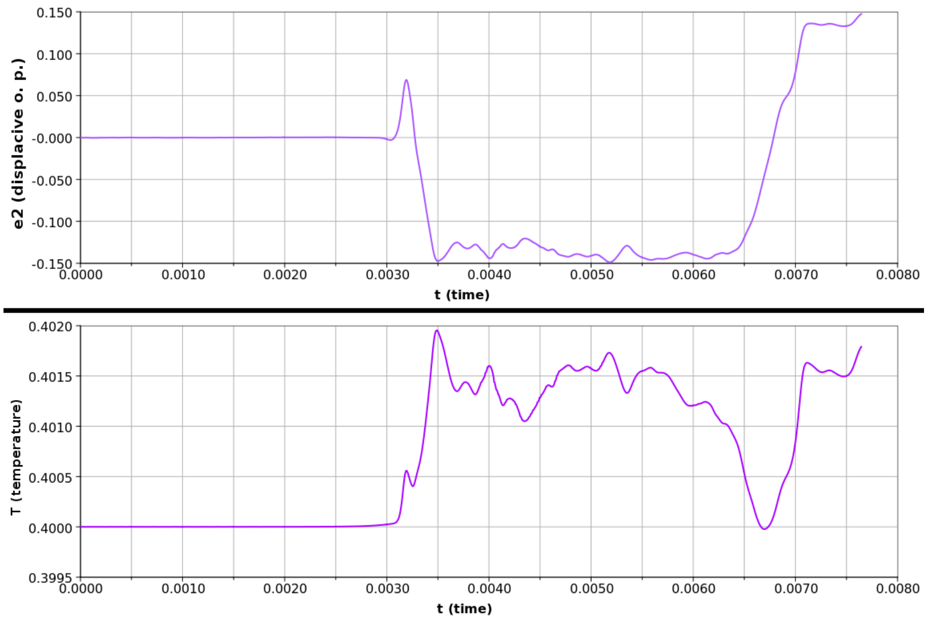


Figure 6.9: Displacive o.p. and temperature evolution at a point of the domain; $T_{ext} = 0.4$, point coordinates: $x = 0.51$, $y = 0.40$.

at this time scale[1], the changes in temperature are driven by the thermo-mechanical coupling term and the dissipation term in the heat equation, which act as heat sources when a displacive process occurs ($\dot{e}_2 \neq 0$). A reduced form of the heat equation, obtained disregarding the heat conduction term (heat conduction cannot occur at this time scale) and the others related to displacive phenomena, may help in highlighting this concept:

$$c_s(T)\dot{T} = T\left[\frac{F}{T_M}\;e_2\;\dot{e}_2 - 2\;G\;\alpha\;\dot{e}_1\right] + \beta\dot{e}_2^2. \tag{6.5}$$

As a result, temperature may increase, or even decrease, accordingly to the observations stated in Sect. 3.5. This behaviour is better highlighted in fig. 6.9: from $t \simeq 0.006$ the temperature basically decreases when the order parameter is increasing and has a negative value, while starts increasing again when the order parameter is still increasing, but has reached a positive value, accordingly to the sign of the thermo-mechanical coupling term in the heat equation:

$$T\;\frac{F}{T_M}\;e_2\;\dot{e}_2.$$

[1]even if for these simulations diffusive phenomena have been accelerated (see Sect. 6.1).

Note also that the temperature and the coefficient F of the Helmholtz free energy act as a gain factor for the production of heat, similarly to B for the case discussed in Sect. 6.4; this is coherent with the observation that, leaving unchanged the ratios between all the coefficients, the aforementioned part of the free energy can be written as:

$$\psi^*_{\text{displ}} = F\left[\frac{D}{6\,F}e_2^6 - \frac{E}{4\,F}e_2^4 + \frac{1}{2}\frac{T-T_M}{T_M}e_2^2\right],$$

showing that if F is increased, the difference:

$$\psi^*_{\text{displ}}\left(e_2=0\right) - min\left(\psi^*_{\text{displ}}\right)$$

increases.

The behaviour described above is slightly modified by the dissipative term:

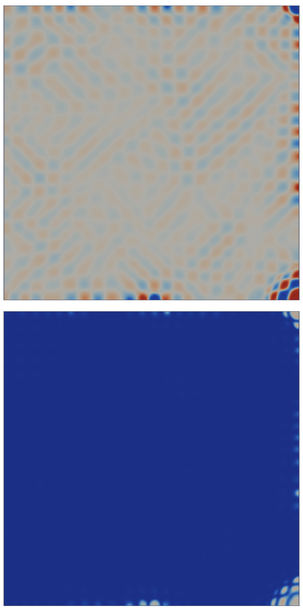
$$\beta \dot{e_2}^2,$$

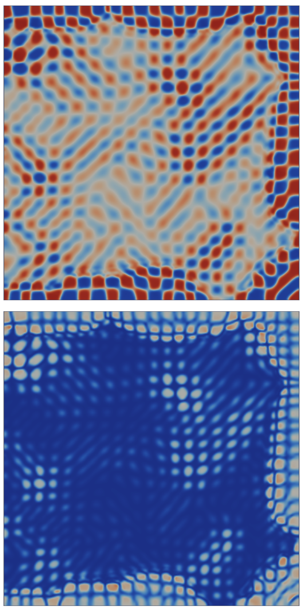
which always tends to increase the temperature if a displacive phase change occurs ($\dot{e_2} \neq 0$); the presence of this term causes the trend of the temperature to be always increasing each time the moving interface produces a strong change in the value of the order parameter, as dissipations are connected with the switch from one Martensite variant to the other (see fig. 6.11).

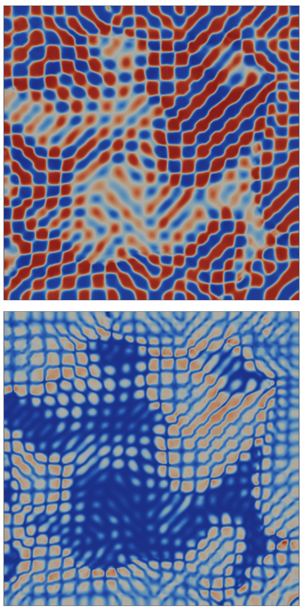
In figure 6.10, the evolution of the whole domain is depicted; only the temperature and the displacive order parameter are represented, as diffusive phenomena cannot take place in such short time at this temperature, which is above T_M, and the value of c remains therefore null[1].

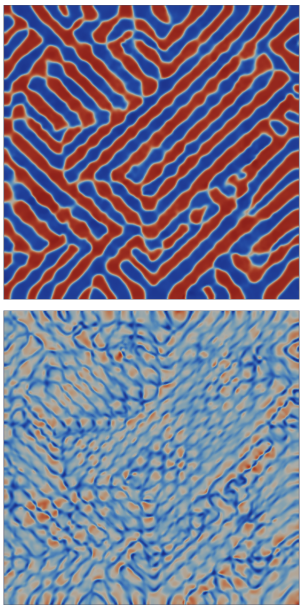
Note that the transition starts from the free edges first, in contrast with the behaviour of the diffusive phase transition. Phases tend to form a right angle with the edges of the boundary, but the most energetically favourable configuration at a distance from the edges is a 45° banded pattern of the different layers, which represents the two possible variants of Martensite in two dimensions, often referred to as M^+ and M^- and identified with a distortion of the lattice in one direction or in the other one (see fig. 6.11); the two variants are described, from an energy point of view, by the double-minimum displacive part of the Helmholtz free energy (see fig. 3.2). The composition of alternate layers gives rise to the so-called *twinned Martensite*, which is the only allowed by our model if no

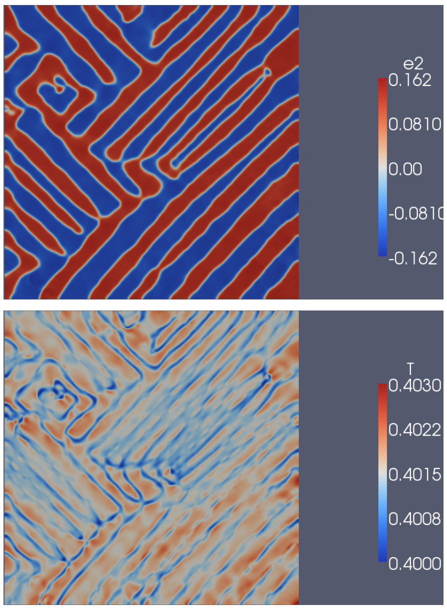
[1]i.e. its value changes, but remains between $\pm 10^{-5}$, the value of the initial random.


(a) $t = 0.0020$


(b) $t = 0.00275$


(c) $t = 0.0030$


(d) $t = 0.0040$



(e) $t = 0.0075$

Figure 6.10: Displacive o.p and temperature evolution throughout the domain; $T_{ext} = 0.4$.

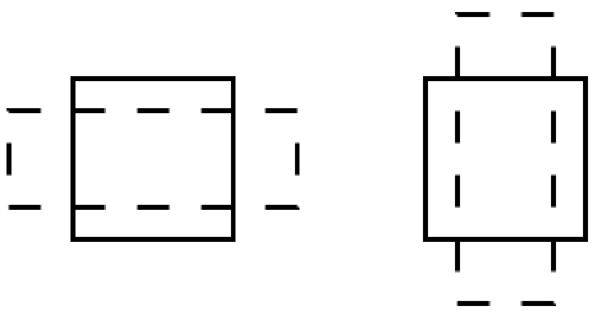


Figure 6.11: Martensite variants in two dimensions.

external load is applied. Applying an external load would result in having one of the two minima depicted in fig. 3.2 to be lower than the other, making one variant more favourable with respect to the other; the resulting microstructure would be a *de-twinned* Martensite[1].

After some time, a little coarsening can be observed.

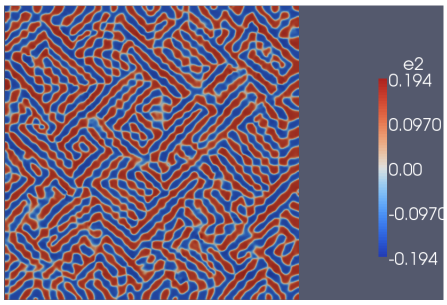
Temperature changes accordingly: there is a general increase due to the heat generated by the displacive transition, while where interfaces between the two variants are located the temperature remains low.

As in the previous case of Sect. 6.4,the width of the bands can be controlled by tuning the value of the gradient coefficient K_e, which plays a relevant role on the transition onset time as well: a high value of K_e will cause the Martensite bands to be larger from early stages of formation and will increase the nucleation time, because it will increase the energy related to the interface construction.

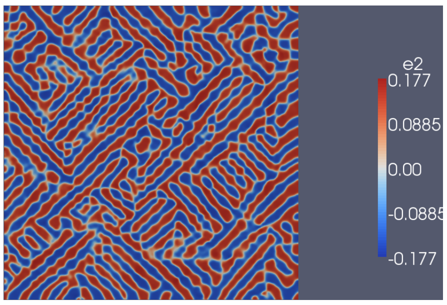
6.5.1 Displacive phase transition at various temperatures

Figure 6.12 depicts a comparison between the morphology of the Martensite obtained for different values of $\tilde{T}_{ext}$; the images refer, for each test, to the time at which Martensite has formed in every part of the domain, and are scaled to the stable value proper of the considered temperature, in order to set the attention on the morphology only and to highlight that at high temperature a coarse phase is obtained, while the structure becomes finer (thinner bands) as temperature decreases.

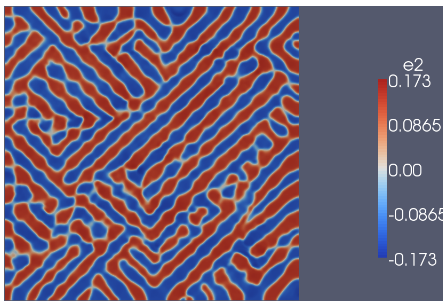
[1]this behaviour is typical of shape memory alloys; our model permits it, but we did not consider this aspect in our investigations.



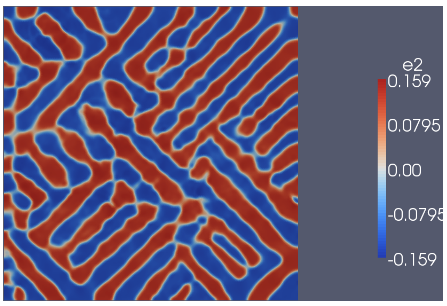
(a) $\tilde{T}_{ref} = 0.2$



(b) $\tilde{T}_{ref} = 0.3$

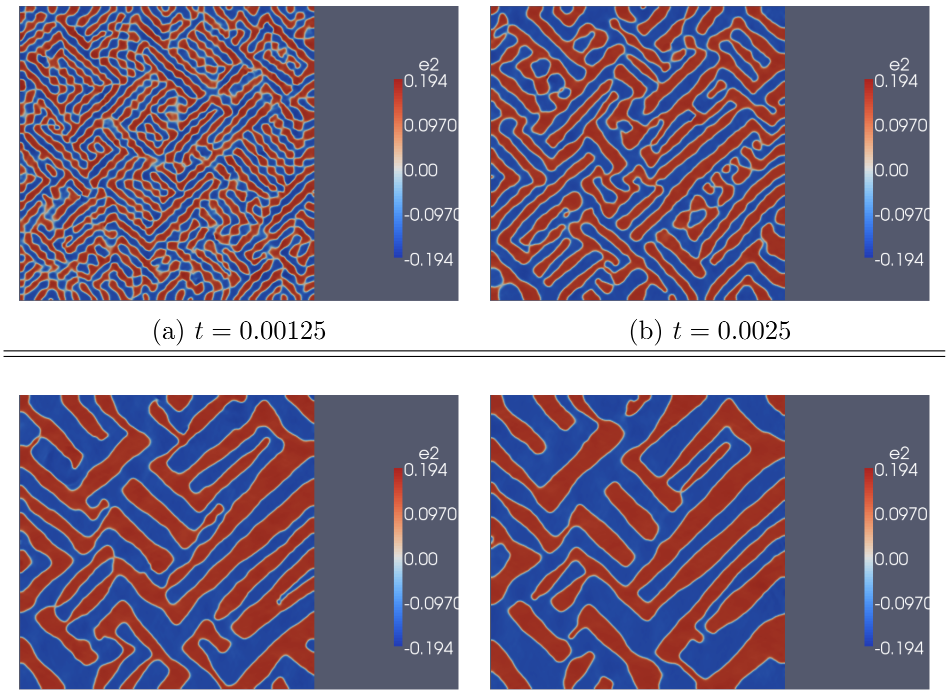


(c) $\tilde{T}_{ref} = 0.4$



(d) $\tilde{T}_{ref} = 0.45$

Figure 6.12: Morphology of the displacive phase for different values of T_{ext}.



(a) $t = 0.00125$ (b) $t = 0.0025$

(c) $t = 0.0050$ (d) $t = 0.0070$

Figure 6.13: Coarsening effect in time; $\tilde{T}_{ext} = 0.2$.

This trend, however, tends to be deleted after a long time, because coarsening phenomena are relevant (see fig. 6.13 and compare fig. 6.13d with fig. 6.12d).

6.6 Open Issues - Overshoots in Temperature

One of the remaining issues regards the excessive overshoots in the temperature experienced when a displacive phase transition occurs; as can be seen from figure 6.15, especially between $t \simeq 0.0010$ and $t \simeq 0.0015$, temperature exhibits a suspicious overshoot. This behaviour is thought to be partly physical, as also if a diffusive phase transition - governed by a first-order time derivative equation - occurs it persists, but partly numerical, and it is thought to be related to the thermo-mechanical coupling term and the dissipative term in the heat equation, containing, respectively, the time derivative of e_2 and its square.

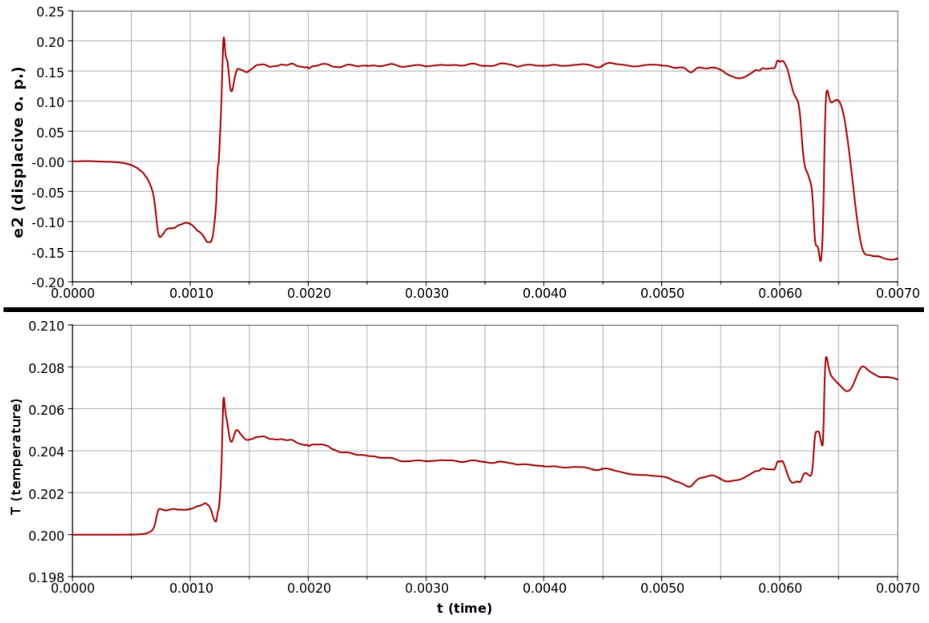


Figure 6.14: Overshoots in temperature; $\tilde{T}_{ext} = 0.2$.

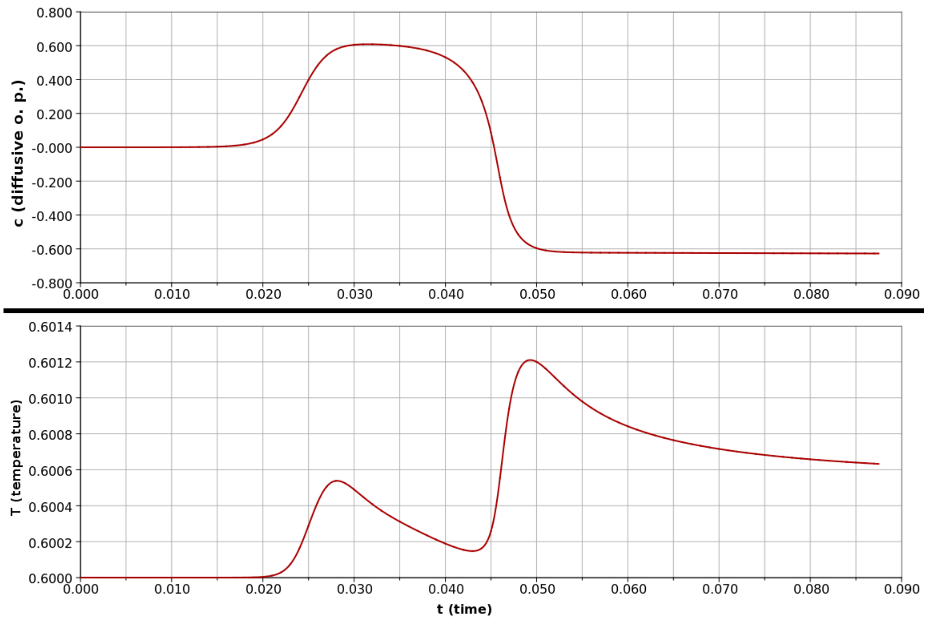


Figure 6.15: Overshoots in temperature; $\tilde{T}_{ext} = 0.6$.

Chapter 7

Concluding Remarks

A phase field, diffuse interface model has been proposed for modelling both diffusive and displacive phase transitions in steels; the description of the phenomena is accomplished by means of two order parameters and of their gradients.

A thermodynamical consistent framework has been proposed for the treatment of the model and specifically to deal with its non-local character; three balance equations have been introduced: the balance of linear momentum equation, the Cahn-Hilliard equation as solute mass balance and the balance of internal energy equation. Thermodynamics considerations allowed to highlight the dissipation sources of the model and to derive proper constitutive relations for the stress tensor and the chemical potential, both composed of a local and a non-local part. A convenient form of the Helmholtz free energy as a sum of various contributions has been proposed and initial as well as local and non-local boundary conditions have been assumed and justified.

The resulting system of equations has been cast in a variational form and coded in *DOLFIN* making use of different numerical techniques. The results of the simulations conducted have shown that the model is able to describe the main features of both diffusive and displacive phase transitions in a satisfactory manner; especially the effect of the phase transitions on the temperature (due to the thermo-chemo-mechanical coupling effects and the dissipations) have been highlighted and examined.

Appendix A

DOLFIN/FFC Code

ThermoChemicalMechanical.ufl

```
# Copyright (C) 2009 Mirko Maraldi and Garth N. Wells
# Licensed under the GNU LGPL Version 2.1.
#
# First added:  2008
# Last changed: 2010
#

P2v = VectorElement("Lagrange", "triangle", 2)
P1  = FiniteElement("Lagrange", "triangle", 1)
ME  = MixedElement([P2v, P1, P1, P1])

# Displacement, temperature, concentration, chem potential
(r,   q,  s,   y) = TestFunctions(ME)
dU = TrialFunction(ME)
U1 = Coefficient(ME)              # Current solution
U0 = Coefficient(ME)              # Previous converged step
du, dT, dc, dmu = split(dU)
u1, T1, c1, mu1 = split(U1)
u0, T0, c0, mu0 = split(U0)

```

```
#------------------------------------------------------------------------------
# Time stepping parameters
dt       = Constant(triangle)   # time step
theta    = Constant(triangle)
theta_m  = Constant(triangle)
theta_s  = Constant(triangle)
beta     = Constant(triangle)
gamma    = Constant(triangle)

# Parameters
cT       = Constant(triangle)  # heat capacity
kT       = Constant(triangle)  # conductivity
hT       = Constant(triangle)  # heat flux coefficient
T_ext    = Constant(triangle)  # external temperature
rho      = Constant(triangle)  # density
eta      = Constant(triangle)  # damping coefficient
Mob      = Constant(triangle)  # chemical mobility
Q        = Constant(triangle)  # activation energy
x12      = Constant(triangle)  # energy coupling parameter
x1c      = Constant(triangle)  # energy coupling parameter
x2c      = Constant(triangle)  # energy coupling parameter
K4       = Constant(triangle)  # strain gradient term
lmbda    = Constant(triangle)  # surface energy term
T_ref    = Coefficient(P1)     # reference temperature for undeformed configuration
alpha    = Constant(triangle)  # thermal dilatation coefficient

#------------------------------------------------------------------------------
# Acceleration, velocity (previous converged step)
a0       = Coefficient(P2v)
v0       = Coefficient(P2v)

# Acceleration, velocity (current)
a1      = (u1 - u0 - dt*v0 - (0.5-beta)*dt*dt*a0)/(beta*dt*dt)
v1      = v0 + (1.0-gamma)*dt*a0 + gamma*dt*a1

# Facet normal and mesh size
n = P1.cell().n
```

```
h = Constant(triangle)
h_avg = (h('+') + h('-'))/2.0

#------------------------------------------------------------------------
# Displacement, velocity, acceleration, temp, concentration, chem potential
# Theta-method
u_mid  = (1.0-theta)*u1  + theta*u0
v_mid  = (1.0-theta)*v1  + theta*v0
a_mid  = (1.0-theta)*a1  + theta*a0
T_mid  = (1.0-theta)*T1  + theta*T0
c_mid  = (1.0-theta)*c1  + theta*c0
mu_mid = (1.0-theta)*mu1 + theta*mu0
# Generalised-alpha method
u_mid_g  = (1.0-theta_s)*u1  + theta_s*u0
v_mid_g  = (1.0-theta_s)*v1  + theta_s*v0
a_mid_g  = (1.0-theta_m)*a1  + theta_m*a0
T_mid_g  = (1.0-theta_s)*T1  + theta_s*T0
c_mid_g  = (1.0-theta_s)*c1  + theta_s*c0

# Displacement, temperature, concentration (rates)
u_rate = (u1 - u0)/dt
T_rate = (T1 - T0)/dt
c_rate = (c1 - c0)/dt

#------------------------------------------------------------------------
# Strain terms
# Hydrostatic strain (1/sqrt(2))*(eps_xx + eps_yy)
def e1(r):
    return (1.0/sqrt(2.0))*(r[0].dx(0) + r[1].dx(1))

# Strain (1/sqrt(2))*(eps_xx - eps_yy)
def e2(r):
    return (1.0/sqrt(2.0))*(r[0].dx(0) - r[1].dx(1))

# Shear strain (1/2)(dv/dx + du/dy)
def e3(r):
    return 0.5*(r[0].dx(1) + r[1].dx(0))
```

```
#----------------------------------------------------------------------------------------
# Free energy
#
# Variables:
# n + theta
e1_mid   = variable(e1(u_mid))
e2_mid   = variable(e2(u_mid))
e3_mid   = variable(e3(u_mid))
grad_e2 = variable(grad(e2(u_mid)))
T_mid      = variable(T_mid)
c_mid      = variable(c_mid)

# n + 1
c1       = variable(c1)
e2_1     = variable(e2(u1))
grad_c1 = variable(grad(c1))

# n + theta_s
e1_mid_g   = variable(e1(u_mid_g))
e2_mid_g   = variable(e2(u_mid_g))
e3_mid_g   = variable(e3(u_mid_g))
grad_e2_g = variable(grad(e2(u_mid_g)))
T_mid_g      = variable(T_mid_g)
c_mid_g      = variable(c_mid_g)

# Displacive/elastic free energy (Bouville et al)
# f_disp = D*(e2^6)/6.0 - E*(e2^4)/4.0 + F*((T-TM)/TM)*(e2^2)/2.0
#          + (G/2.0)*e1*(e1 - 2*sqrt(2)*(alpha(T-T_ref) + x1c*c + x12*e2^2) )
#          + (H/2.0)*e3^2
#          + (K4/2.0)*(grad e2)^2
#
D  = 29.535
E  = 0.49482
F  = 0.009948
G  = 0.004974
H  = 0.004974
TM = 0.495
```

```

def f_disp(a, b, c, p, s, q):
    return (D/6.0)* b*b*b*b*b*b \
         - (E/4.0)* b*b*b*b \
         + (F/2.0)*((q-TM)/TM) *b*b \
         + (G/2.0)*a* (a - 2.0*sqrt(2.0)* (alpha*(q-T_ref) + x1c*s + x12*b*b) ) \
         + (H/2.0)*c*c \
         + (K4/2.0)*inner(p, p)

# Chemical free energy
# f_diff = (A*c^4)/4 + B*(T-Tp)*c^2/(2*Tp) + lmbda/2 (grad c)^2
#
A     = 0.00073142
B     = 0.00066246
Tp    = 1.0

def f_diff(s, l, q):
    return (A/4.0)*s*s*s*s + B*(q-Tp)*s*s/(2.0*Tp) + (lmbda/2.0)*inner(l, l)

# Coupling free energy
def f_cpl(s, m):
    return x2c * s*s * m*m

# Thermal free energy
f_therm = - (T_mid-T_ref) - cT/2.0 * (T_mid - T_ref)*(T_mid - T_ref)

#------------------------------------------------------------------------------
# Balance of linear momentum (at generalised-alpha-point)
#
sigma1  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, c_mid_g, T_mid_g),
    e1_mid_g)
sigma2  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, c_mid_g, T_mid_g),
    e2_mid_g) \
        + diff(f_cpl(c_mid_g, e2_mid_g), e2_mid_g)
sigma3  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, c_mid_g, T_mid_g),
    e3_mid_g)
```

```
sigma_g = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, c_mid_g, T_mid_g),
    grad_e2_g)


L_stress_intertia = rho*dot(r, a_mid_g)*dx
L_stress_viscous  = eta*e2(v_mid_g)*sqrt(2.0)*e2(r)*dx
L_stress_regular  = e1(r)*sigma1*dx + e2(r)*sigma2*dx + 2.0*e3(r)*sigma3*dx
L_stress_gradient = inner(grad(e2(r)), sigma_g)*dx \
                  - inner(jump(e2(r), n), avg(sigma_g))*dS \
                  - inner(avg(K4*grad(e2(r))), jump(e2(u_mid_g), n))*dS \
                  + 8.0*(K4('+')/h_avg)*inner(jump(e2(r)), jump(e2(u_mid_g)))*dS

L_stress = L_stress_intertia + L_stress_viscous + L_stress_regular + L_stress_gradient
a_stress = derivative(L_stress, U1, dU)

#------------------------------------------------------------------------------
# Mass diffusion (at mid-point)

# Temperature dependent mobility
Mob_T = Mob*exp(-Q/T_mid)

# Chemical potential df/dc
mu_c    = diff(f_diff(c1, grad_c1, T1), c1) \
        + diff(f_cpl(c1, e2_1), c1) \
        + diff(f_disp(e1(u1), e2(u1), e3(u1), grad(e2(u1)), c1, T1), c1)
mu_grad = diff(f_diff(c1, grad_c1, T1), grad_c1)

# Weak form
L_c_mass        =  s*c_rate*dx + Mob_T*inner(grad(s), grad(mu_mid))*dx
L_c_potential   =  y*mu1*dx - y*mu_c*dx - inner(grad(y), mu_grad)*dx

L_mass = L_c_mass  + L_c_potential
a_mass = derivative(L_mass, U1, dU)

#------------------------------------------------------------------------------
# Heat equation (at mid-point)

# Heat flux - Fourier law
q_flux = -kT*grad(T_mid)

# (Dsigma/DT)(de/dt)
sigma1_t  = diff(f_disp(e1_mid, e2_mid, e3_mid, grad_e2, c_mid, T_mid), e1_mid)
sigma2_t  = diff(f_disp(e1_mid, e2_mid, e3_mid, grad_e2, c_mid, T_mid), e2_mid) \
          + diff(f_cpl(c_mid, e2_mid), e2_mid)
sigma3_t  = diff(f_disp(e1_mid, e2_mid, e3_mid, grad_e2, c_mid, T_mid), e3_mid)

D_sigma_DTde1 = diff(sigma1_t, T_mid)*e1(u_rate)
D_sigma_DTde2 = diff(sigma2_t, T_mid)*e2(u_rate)
D_sigma_DTde3 = diff(sigma3_t, T_mid)*e3(u_rate)
D_sigma_DTde  = D_sigma_DTde1 + D_sigma_DTde2 + D_sigma_DTde3

# (Dmu/DT)(dc/dt)
D_mu_DTdc = diff( diff(f_diff(c_mid, grad(c_mid), T_mid), c_mid), T_mid )*c_rate

# Heat capacity
c_therm =- diff( diff(f_therm, T_mid), T_mid)

# Weak form
L_heat    = q*T_mid*c_therm*T_rate*dx - q*T_mid*D_sigma_DTde*dx \
                                       - q*T_mid*D_mu_DTdc*dx \
          - q*Mob_T*inner(grad(mu_mid), grad(mu_mid))*dx \
          - q*eta*sqrt(2.0)*e2(v_mid)*e2(v_mid)*dx \
          - inner(grad(q), q_flux)*dx \
          + q*hT*(T_mid - T_ext)*ds

a_heat = derivative(L_heat, U1, dU)

#----------------------------------------------------------------------
# Total forms
a = a_stress + a_heat + a_mass
L = L_stress + L_heat + L_mass
```

A. DOLFIN/FFC CODE

ThermoMechanical.ufl

```
# Copyright (C) 2009 Mirko Maraldi and Garth N. Wells
# Licensed under the GNU LGPL Version 2.1.
#
# First added:  2008
# Last changed: 2010
#

P2v= VectorElement("Lagrange", "triangle", 2)
P1 = FiniteElement("Lagrange", "triangle", 1)
P0 = FiniteElement("Discontinuous Lagrange", "triangle", 0)
ME = MixedElement([P2v, P1])

# Displacement, temperature
(r, q) = TestFunctions(ME)
dU     = TrialFunction(ME)
U1     = Coefficient(ME)
U0     = Coefficient(ME)
du, dT = split(dU)
u1, T1 = split(U1)             # current solution
u0, T0 = split(U0)             # previous converged step

#------------------------------------------------------------------------------
# Time stepping parameters
dt      = Constant(triangle)   # time step
theta   = Constant(triangle)
theta_m = Constant(triangle)
theta_s = Constant(triangle)
beta    = Constant(triangle)
gamma   = Constant(triangle)

# Parameters
cT      = Constant(triangle)  # heat capacity
kT      = Constant(triangle)  # conductivity
hT      = Constant(triangle)  # heat flux coefficient
T_ext   = Constant(triangle)  # external temperature
rho     = Constant(triangle)  # density
```

```
eta     = Constant(triangle)  # damping coefficient
x12     = Constant(triangle)
x2c     = Constant(triangle)
K4      = Constant(triangle)  # strain gradient term
T_ref   = Coefficient(P1)     # reference temperature for undeformed configuration
alpha   = Constant(triangle)  # thermal dilatation coefficient

#-------------------------------------------------------------------------------
# Acceleration, velocity (previous converged step)
a0      = Coefficient(P2v)
v0      = Coefficient(P2v)

# Acceleration, velocity (current)
a1     = (u1 - u0 - dt*v0 - (0.5-beta)*dt*dt*a0)/(beta*dt*dt)
v1     = v0 + (1.0-gamma)*dt*a0 + gamma*dt*a1

# Facet normal and mesh size
n = P1.cell().n
h = Constant(triangle)
h_avg = (h('+') + h('-'))/2.0

#-------------------------------------------------------------------------------
# Displacement, velocity, acceleration, temperature
# Theta-method
u_mid  = (1.0-theta)*u1  + theta*u0
v_mid  = (1.0-theta)*v1  + theta*v0
T_mid  = (1.0-theta)*T1  + theta*T0
# Generalised-alpha method
u_mid_g  = (1.0-theta_s)*u1  + theta_s*u0
v_mid_g  = (1.0-theta_s)*v1  + theta_s*v0
a_mid_g  = (1.0-theta_m)*a1  + theta_m*a0
T_mid_g  = (1.0-theta_s)*T1  + theta_s*T0

# Displacement, temperature (rates)
u_rate = (u1 - u0)/dt
T_rate = (T1 - T0)/dt

```

```
#-----------------------------------------------------------------------
# Strain terms
# Hydrostatic strain (1/sqrt(2))*(eps_xx + eps_yy)
def e1(r):
    return (1.0/sqrt(2.0))*(r[0].dx(0) + r[1].dx(1))

# Strain (1/sqrt(2))*(eps_xx - eps_yy)
def e2(r):
    return (1.0/sqrt(2.0))*(r[0].dx(0) - r[1].dx(1))

# Shear strain (1/2)(dv/dx + du/dy)
def e3(r):
    return 0.5*(r[0].dx(1) + r[1].dx(0))
#-----------------------------------------------------------------------
# Free energy
#
# Variables:
# n + theta
e1_mid  = variable(e1(u_mid))
e2_mid  = variable(e2(u_mid))
e3_mid  = variable(e3(u_mid))
grad_e2 = variable(grad(e2(u_mid)))
T_mid     = variable(T_mid)

# n + 1
e2_1    = variable(e2(u1))

# n + theta_s
e1_mid_g  = variable(e1(u_mid_g))
e2_mid_g  = variable(e2(u_mid_g))
e3_mid_g  = variable(e3(u_mid_g))
grad_e2_g = variable(grad(e2(u_mid_g)))
T_mid_g     = variable(T_mid_g)

# Displacive/elastic free energy (Bouville et al)
# f_disp = D*(e2^6)/6.0 - E*(e2^4)/4.0 + F*((T-TM)/TM)*(e2^2)/2.0
#          + (G/2.0)*e1*(e1 - 2*sqrt(2)*(alpha(T-T_ref) + x12*e2^2) )
#           + (H/2.0)*e3^2
#           + (K4/2.0)*(grad e2)^2
#
D  = 29.535
E  = 0.49482
F  = 0.009948
G  = 0.004974
H  = 0.004974
TM = 0.495

def f_disp(a, b, c, p, q):
    return (D/6.0)* b*b*b*b*b*b \
        - (E/4.0)* b*b*b*b \
        + (F/2.0)*((q-TM)/TM) *b*b \
        + (G/2.0)*a* (a - 2.0*sqrt(2.0)* (alpha*(q-T_ref) + x12*b*b) ) \
        + (H/2.0)*c*c \
        + (K4/2.0)*inner(p, p)

# Thermal free energy
f_therm = - (T_mid-T_ref) - cT/2.0 * (T_mid - T_ref)*(T_mid - T_ref)

#------------------------------------------------------------------------------
# Balance of linear momentum (at mid-point)
#
sigma1  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, T_mid_g), e1_mid_g)
sigma2  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, T_mid_g), e2_mid_g)
sigma3  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, T_mid_g), e3_mid_g)
sigma_g = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, T_mid_g), grad_e2_g)


L_stress_intertia = rho*dot(r, a_mid_g)*dx
L_stress_viscous  = eta*e2(v_mid_g)*sqrt(2.0)*e2(r)*dx
L_stress_regular  = e1(r)*sigma1*dx + e2(r)*sigma2*dx + 2.0*e3(r)*sigma3*dx
L_stress_gradient = inner(grad(e2(r)), sigma_g)*dx \
                    - inner(jump(e2(r), n), avg(sigma_g))*dS \
                    - inner(avg(K4*grad(e2(r))), jump(e2(u_mid_g), n))*dS \
                    + 8.0*(K4('+')/h_avg)*inner(jump(e2(r)), jump(e2(u_mid_g)))*dS
```

```

L_stress = L_stress_intertia + L_stress_viscous + L_stress_regular + L_stress_gradient
a_stress = derivative(L_stress, U1, dU)

#--------------------------------------------------------------------------
# Heat equation (at mid-point)

# Heat flux - Fourier law
q_flux = -kT*grad(T_mid)

# (Dsigma/DT)(de/dt)
sigma1_t  = diff(f_disp(e1_mid, e2_mid, e3_mid, grad_e2, T_mid), e1_mid)
sigma2_t  = diff(f_disp(e1_mid, e2_mid, e3_mid, grad_e2, T_mid), e2_mid)
sigma3_t  = diff(f_disp(e1_mid, e2_mid, e3_mid, grad_e2, T_mid), e3_mid)

D_sigma_DTde1 = diff(sigma1_t, T_mid)*e1(u_rate)
D_sigma_DTde2 = diff(sigma2_t, T_mid)*e2(u_rate)
D_sigma_DTde3 = diff(sigma3_t, T_mid)*e3(u_rate)
D_sigma_DTde  = D_sigma_DTde1 + D_sigma_DTde2 + D_sigma_DTde3

# Heat capacity
c_therm = - diff( diff(f_therm, T_mid), T_mid)

# Weak form
L_heat     = q*T_mid*c_therm*T_rate*dx - q*T_mid*D_sigma_DTde*dx \
            - q*eta*sqrt(2.0)*e2(v_mid)*e2(v_mid)*dx \
            - inner(grad(q), q_flux)*dx \
            + q*hT*(T_mid - T_ext)*ds

a_heat = derivative(L_heat, U1, dU)

#--------------------------------------------------------------------------
# Total forms
a = a_stress + a_heat
L = L_stress + L_heat
```

ChemicalMechanical.ufl

```
# Copyright (C) 2009 Mirko Maraldi and Garth N. Wells
# Licensed under the GNU LGPL Version 2.1.
#
# First added:  2008
# Last changed: 2010
#

P2v = VectorElement("Lagrange", "triangle", 2)
P1  = FiniteElement("Lagrange", "triangle", 1)
ME  = MixedElement([P2v, P1, P1])

# Displacement, temperature, concentration, chem potential
(r,    s,    y) = TestFunctions(ME)
dU = TrialFunction(ME)
U1 = Coefficient(ME)              # Current solution
U0 = Coefficient(ME)              # Previous converged step
du, dc, dmu = split(dU)
u1, c1, mu1 = split(U1)
u0, c0, mu0 = split(U0)

#-----------------------------------------------------------------------------
# Time stepping parameters
dt      = Constant(triangle)   # time step
theta   = Constant(triangle)
theta_m = Constant(triangle)
theta_s = Constant(triangle)
beta    = Constant(triangle)
gamma   = Constant(triangle)

# Parameters
T1      = Constant(triangle)  # external temperature
rho     = Constant(triangle)  # density
eta     = Constant(triangle)  # damping coefficient
Mob     = Constant(triangle)  # chemical mobility
Q       = Constant(triangle)  # activation energy
x12     = Constant(triangle)  # energy coupling parameter
```

```
x1c      = Constant(triangle)  # energy coupling parameter
x2c      = Constant(triangle)  # energy coupling parameter
K4       = Constant(triangle)  # strain gradient term
lmbda    = Constant(triangle)  # surface energy term

#------------------------------------------------------------------------------
# Acceleration, velocity (previous converged step)
a0        = Coefficient(P2v)
v0        = Coefficient(P2v)

# Acceleration, velocity (current)
a1      = (u1 - u0 - dt*v0 - (0.5-beta)*dt*dt*a0)/(beta*dt*dt)
v1      = v0 + (1.0-gamma)*dt*a0 + gamma*dt*a1

# Facet normal and mesh size
n = P1.cell().n
h = Constant(triangle)
h_avg = (h('+') + h('-'))/2.0

#------------------------------------------------------------------------------
# Displacement, velocity, acceleration, temp, concentration, chem potential
# Theta-method
mu_mid = (1.0-theta)*mu1 + theta*mu0
# Generalised-alpha method
u_mid_g  = (1.0-theta_s)*u1  + theta_s*u0
v_mid_g  = (1.0-theta_s)*v1  + theta_s*v0
a_mid_g  = (1.0-theta_m)*a1  + theta_m*a0
c_mid_g  = (1.0-theta_s)*c1  + theta_s*c0

# Displacement, temperature, concentration (rates)
c_rate = (c1 - c0)/dt

#------------------------------------------------------------------------------
# Strain terms
# Hydrostatic strain (1/sqrt(2))*(eps_xx + eps_yy)
def e1(r):
    return (1.0/sqrt(2.0))*(r[0].dx(0) + r[1].dx(1))
```

```

# Strain (1/sqrt(2))*(eps_xx - eps_yy)
def e2(r):
    return (1.0/sqrt(2.0))*(r[0].dx(0) - r[1].dx(1))

# Shear strain (1/2)(dv/dx + du/dy)
def e3(r):
    return 0.5*(r[0].dx(1) + r[1].dx(0))
#------------------------------------------------------------------------------
# Free energy
#
# Variables:
# n + 1
c1      = variable(c1)
e2_1    = variable(e2(u1))
grad_c1 = variable(grad(c1))

# n + theta_s
e1_mid_g  = variable(e1(u_mid_g))
e2_mid_g  = variable(e2(u_mid_g))
e3_mid_g  = variable(e3(u_mid_g))
grad_e2_g = variable(grad(e2(u_mid_g)))
c_mid_g   = variable(c_mid_g)

# Displacive/elastic free energy (Bouville et al)
# f_disp = D*(e2^6)/6.0 - E*(e2^4)/4.0 + F*((T-TM)/TM)*(e2^2)/2.0
#          + (G/2.0)*e1*(e1 - 2*sqrt(2)*(x1c*c + x12*e2^2) )
#          + (H/2.0)*e3^2
#          + (K4/2.0)*(grad e2)^2
#
D  = 29.535
E  = 0.49482
F  = 0.009948
G  = 0.004974
H  = 0.004974
TM = 0.495

```

```
def f_disp(a, b, c, p, s):
    return (D/6.0)* b*b*b*b*b*b \
          - (E/4.0)* b*b*b*b \
          + (F/2.0)*((T1-TM)/TM) *b*b \
          + (G/2.0)*a* (a - 2.0*sqrt(2.0)* (x1c*s + x12*b*b) ) \
          + (H/2.0)*c*c \
          + (K4/2.0)*inner(p, p)

# Chemical free energy
# f_diff = (A*c^4)/4 + B*(T-Tp)*c^2/(2*Tp) + lmbda/2 (grad c)^2
#
A      = 0.00073142
B      = 0.00066246
Tp     = 1.0

def f_diff(s, l):
    return (A/4.0)*s*s*s*s + B*(T1-Tp)*s*s/(2.0*Tp) + (lmbda/2.0)*inner(l, l)

# Coupling free energy
def f_cpl(s, m):
    return x2c * s*s * m*m

#-----------------------------------------------------------------------------
# Balance of linear momentum (at generalised-alpha-point)
#
sigma1  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, c_mid_g), e1_mid_g)
sigma2  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, c_mid_g), e2_mid_g) \
        + diff(f_cpl(c_mid_g, e2_mid_g), e2_mid_g)
sigma3  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, c_mid_g), e3_mid_g)
sigma_g = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g, c_mid_g), grad_e2_g)


L_stress_intertia = rho*dot(r, a_mid_g)*dx
L_stress_viscous  = eta*e2(v_mid_g)*sqrt(2.0)*e2(r)*dx
L_stress_regular  = e1(r)*sigma1*dx + e2(r)*sigma2*dx + 2.0*e3(r)*sigma3*dx
L_stress_gradient = inner(grad(e2(r)), sigma_g)*dx \
                  - inner(jump(e2(r), n), avg(sigma_g))*dS \
                    - inner(avg(K4*grad(e2(r))), jump(e2(u_mid_g), n))*dS \
                    + 8.0*(K4('+')/h_avg)*inner(jump(e2(r)), jump(e2(u_mid_g)))*dS

L_stress = L_stress_intertia + L_stress_viscous + L_stress_regular + L_stress_gradient
a_stress = derivative(L_stress, U1, dU)

#-------------------------------------------------------------------------------
# Mass diffusion (at mid-point)

# Temperature dependent mobility
Mob_T = Mob*exp(-Q/T1)

# Chemical potential df/dc
mu_c     = diff(f_diff(c1, grad_c1), c1) \
         + diff(f_cpl(c1, e2_1), c1) \
         + diff(f_disp(e1(u1), e2(u1), e3(u1), grad(e2(u1)), c1), c1)
mu_grad = diff(f_diff(c1, grad_c1), grad_c1)

# Weak form
L_c_mass        =  s*c_rate*dx + Mob_T*inner(grad(s), grad(mu_mid))*dx
L_c_potential   =  y*mu1*dx - y*mu_c*dx - inner(grad(y), mu_grad)*dx

L_mass = L_c_mass  + L_c_potential
a_mass = derivative(L_mass, U1, dU)

#-------------------------------------------------------------------------------
# Total forms
a = a_stress + a_mass
L = L_stress + L_mass
```

A. DOLFIN/FFC CODE

Mechanical.ufl

```
# Copyright (C) 2008 Mirko Maraldi and Garth N. Wells
# Licensed under the GNU LGPL Version 2.1.
#
# First added:  2008
# Last changed: 2010

P2v= VectorElement("Lagrange", "triangle", 2)
P0 = FiniteElement("Discontinuous Lagrange", "triangle", 0)

# Displacement
r   = TestFunction(P2v)
du  = TrialFunction(P2v)
u1  = Coefficient(P2v)          # current solution
u0  = Coefficient(P2v)          # previous converged step

# Time stepping parameters
dt       = Constant(triangle)   # time step
theta_m  = Constant(triangle)
theta_s  = Constant(triangle)
beta     = Constant(triangle)
gamma    = Constant(triangle)

# Parameters
T1       = Constant(triangle)  # temperature
rho      = Constant(triangle)  # density
eta      = Constant(triangle)  # damping coefficient
K4       = Constant(triangle)  # strain gradient term
x12      = Constant(triangle)  # coupling between e1 and e2

#----------------------------------------------------------------------------
# Acceleration, velocity (previous converged step)
a0       = Coefficient(P2v)
v0       = Coefficient(P2v)

# Acceleration, velocity (current)
a1      = (u1 - u0 - dt*v0 - (0.5-beta)*dt*dt*a0)/(beta*dt*dt)
```

```
v1      = v0 + (1.0-gamma)*dt*a0 + gamma*dt*a1

# Facet normal and mesh size
n = P2v.cell().n
h = Constant(triangle)
h_avg = (h('+') + h('-'))/2.0

#-----------------------------------------------------------------------------
# Displacement, velocity, acceleration
# Generalised-alpha method
u_mid_g  = (1.0-theta_s)*u1  + theta_s*u0
v_mid_g  = (1.0-theta_s)*v1  + theta_s*v0
a_mid_g  = (1.0-theta_m)*a1  + theta_m*a0

#-----------------------------------------------------------------------------
# Strain terms
# Hydrostatic strain (1/sqrt(2))*(eps_xx + eps_yy)
def e1(r):
    return (1.0/sqrt(2.0))*(r[0].dx(0) + r[1].dx(1))

# Strain (1/sqrt(2))*(eps_xx - eps_yy)
def e2(r):
    return (1.0/sqrt(2.0))*(r[0].dx(0) - r[1].dx(1))

# Shear strain (1/2)(dv/dx + du/dy)
def e3(r):
    return 0.5*(r[0].dx(1) + r[1].dx(0))

#-----------------------------------------------------------------------------
# Free energy
#
# Variables:
# n + theta_s
e1_mid_g   = variable(e1(u_mid_g))
e2_mid_g   = variable(e2(u_mid_g))
e3_mid_g   = variable(e3(u_mid_g))
grad_e2_g = variable(grad(e2(u_mid_g)))
```

```

# Displacive/elastic free energy (Bouville et al)
# f_disp = D*(e2^6)/6.0 - E*(e2^4)/4.0 + F*((T-TM)/TM)*(e2^2)/2.0
#          + (G/2.0)*e1*(e1 - 2*sqrt(2)*x12*e2^2) )
#          + (H/2.0)*e3^2
#          + (K4/2.0)*(grad e2)^2
#
D  = 29.535
E  = 0.49482
F  = 0.009948
G  = 0.004974
H  = 0.004974
TM = 0.495

def f_disp(a, b, c, p):
    return (D/6.0)* b*b*b*b*b*b \
         - (E/4.0)* b*b*b*b \
         + (F/2.0)*((T1-TM)/TM) *b*b \
         + (G/2.0)*a* (a - 2.0*sqrt(2.0)* x12*b*b ) \
         + (H/2.0)*c*c \
         + (K4/2.0)*inner(p, p)

#------------------------------------------------------------------------------
# Balance of linear momentum (at mid-point)
#
sigma1  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g), e1_mid_g)
sigma2  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g), e2_mid_g)
sigma3  = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g), e3_mid_g)
sigma_g = diff(f_disp(e1_mid_g, e2_mid_g, e3_mid_g, grad_e2_g), grad_e2_g)


L_stress_intertia = rho*dot(r, a_mid_g)*dx
L_stress_viscous  = eta*e2(v_mid_g)*sqrt(2.0)*e2(r)*dx
L_stress_regular  = e1(r)*sigma1*dx + e2(r)*sigma2*dx + 2.0*e3(r)*sigma3*dx
L_stress_gradient = inner(grad(e2(r)), sigma_g)*dx \
                  - inner(jump(e2(r), n), avg(sigma_g))*dS \
                  - inner(avg(K4*grad(e2(r))), jump(e2(u_mid_g), n))*dS \
                        + 8.0*(K4('+')/h_avg)*inner(jump(e2(r)), jump(e2(u_mid_g)))*dS

L = L_stress_intertia + L_stress_viscous + L_stress_regular + L_stress_gradient
a = derivative(L, u1, du)
```

e2.ufl

```
# Copyright (C) 2009 Mirko Maraldi and Garth N. Wells.
# Licensed under the GNU LGPL Version 2.1.
#
# First added:  2009-01-22
# Last changed: 2009
#
# Projection
#
# Compile this form with FFC: ffc -l dolfin e2.ufl

P2 = VectorElement("Lagrange", "triangle", 2)
P1 = FiniteElement("Lagrange", "triangle", 1)
P0 = FiniteElement("Discontinuous Lagrange", "triangle", 0)

# displacement, velocity
v      = TestFunction(P1)
e2     = TrialFunction(P1)
u      = Coefficient(P2)

a = v*e2*dx
L = (1.0/sqrt(2.0))*v*(u[0].dx(0) - u[1].dx(1))*dx
```

Model.h

```
// Copyright (C) 2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2009
// Last changed:
//

#ifndef __HEAT_MODEL_H
#define __HEAT_MODEL_H

#include <string>
#include <boost/assign/list_of.hpp>
#include <boost/scoped_ptr.hpp>
#include <boost/shared_ptr.hpp>
#include <dolfin/fem/DirichletBC.h>
#include <dolfin/fem/VariationalProblem.h>
#include <dolfin/function/CoefficientAssigner.h>
#include <dolfin/function/Expression.h>
#include <dolfin/function/Function.h>
#include <dolfin/function/FunctionSpace.h>
#include <dolfin/function/SubSpace.h>
#include <dolfin/function/SpecialFunctions.h>
#include "Parameters.h"
#include "e2.h"

namespace dolfin
{
  namespace heat
  {

    class Model
    {
    public:

      // Constructor
      Model(boost::shared_ptr<FunctionSpace> V, std::string model)
```

```
        : V(V), W(V), W0(V), h(V->mesh()), model(model)
    {
    }

    // Return bilinear form
    const Form& a() const
    { return *_a; }

    // Return linear form
    const Form& L() const
    { return *_L; }

    // Return vector containing boundary conditions
    virtual std::vector<const DirichletBC*>& bcs() = 0;

    // Vector associate with complete (possible mixed) field
    GenericVector& vector()
    { return W.vector(); }

    // Return displacement field
    virtual const Function& displacement() const = 0;

    // Return temperature field
    virtual Function& temperature()
    { error("Model::temperature not implemented"); return W; }

    // Return concentration field
    virtual Function& concentration()
    { error("Model::concentration not implemented"); return W; }

    // Return string describing model
    std::string str() const
    { return model; }

    // Perform updates at the end of a step
    virtual void update() = 0;

    // Compute and return e2 for post-processing
    Function e2() const
    {
      const Function& u = displacement();
      boost::shared_ptr<e2::FunctionSpace> Ve(new e2::FunctionSpace(V->mesh()));
      e2::BilinearForm __a(Ve, Ve);
      e2::LinearForm __L(Ve, u);
      VariationalProblem eps(__a, __L);
      eps.parameters["linear_solver"] = "iterative";

      Function e2(Ve);
      eps.solve(e2);
      return e2;
    }

  protected:

    // Solution function space
    boost::shared_ptr<FunctionSpace> V;

    // Solution function
    Function W;
    Function W0;

    // Forms
    boost::shared_ptr<Form> _a;
    boost::shared_ptr<Form> _L;

    // Cell size
    CellSize h;

  private:

    // Model name
    const std::string model;

  };

  }
}
#endif
```

Updates.h

```
// Copyright (C) 2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2009
// Last changed: 2010
//

#ifndef __HEAT_UPDATES_H
#define __HEAT_UPDATES_H

#include <dolfin/function/Function.h>
#include <dolfin/la/GenericVector.h>

namespace dolfin
{
  namespace heat
  {
    // Acceleration update
    void update_a(Function& a, const Function& u, const Function& a0,
                  const Function& v0,  const Function& u0,
                  double beta, double dt)
    {
      // a = 1/(2*beta) * ((u-u0 - v0*dt)/(0.5*dt*dt) - (1-2*beta)*a0)
      a.vector()  = u.vector();
      a.vector() -= u0.vector();
      a.vector() *= 1.0/dt;
      a.vector() -= v0.vector();
      a.vector() *= 1.0/((0.5-beta)*dt);
      a.vector() -= a0.vector();
      a.vector() *= (0.5-beta)/beta;
    }

    // Velocity update
    void update_v(const Function& a, Function& v, const Function& a0,
                    const Function& v0, double gamma, double dt)
    {
        // v = dt * ((1-Gamma)*a0 + Gamma*a) + v0
        v.vector()  = a0.vector();
        v.vector() *= (1.0-gamma)/gamma;
        v.vector() += a.vector();
        v.vector() *= dt*gamma;
        v.vector() += v0.vector();
      }
    }
}
#endif
```

ThermoChemoMechanicalModel.h

```cpp
// Copyright (C) 2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2009
// Last changed: 2010
//

#ifndef __HEAT_THERMOCHEMOMECHANICAL_MODEL_H
#define __HEAT_THERMOCHEMOMECHANICAL_MODEL_H

#include <boost/assign/list_of.hpp>
#include "Parameters.h"
#include "Model.h"
#include "Updates.h"
#include "ThermoChemicalMechanical.h"

namespace dolfin
{
  namespace heat
  {

    // User defined nonlinear problem
    class ThermoChemoMechanicalModel : public Model
    {
    public:

      // Constructor
      ThermoChemoMechanicalModel(const Mesh& mesh)
          : Model(boost::shared_ptr<FunctionSpace>(new ThermoChemicalMechanical::
              FunctionSpace(mesh)),
            "thermo-chemo-mechanical"),
            // Subspaces
            Vu(new SubSpace(*V, 0)), Vu_x(new SubSpace(*V, 0, 0)),
            Vu_y(new SubSpace(*V, 0, 1)),
            Vt(new SubSpace(*V, 1)), Vc(new SubSpace(*V, 2)),
            // Displacement, velocity, acceleration
          u(new Function(Vu)), u0(new Function(Vu)),
          v(new Function(Vu)), v0(new Function(Vu)),
          acc(new Function(Vu)), acc0(new Function(Vu)),
          // Boundary conditions
          bc_u_1(*Vu_x, zero_scalar, left),
          bc_u_2(*Vu_y, zero_scalar, centre_left, "pointwise")
    {
      // Insert bc into vector
      _bcs.push_back(&bc_u_1);
      _bcs.push_back(&bc_u_2);

      // Initial conditions for displacement and temperature
      InitialConditions UU(ic_dim);
      if (UU.value_dimension(0) != 5)
        error("Wrong initial condition vector dimension. You should have 5 initial
            conditions");
      W.interpolate(UU);
      W0 = W;
      update_ic();

      // Set reference temperature to intial temperature
      reference_temperature.reset(new Function(W[1]));

      // Create forms
      _a.reset(new ThermoChemicalMechanical::BilinearForm(V, V));
      _L.reset(new ThermoChemicalMechanical::LinearForm(V));

      // Attach coefficients
      ThermoChemicalMechanical::BilinearForm* aa = dynamic_cast<
          ThermoChemicalMechanical::BilinearForm*>(_a.get());
      ThermoChemicalMechanical::LinearForm* LL = dynamic_cast<ThermoChemicalMechanical
          ::LinearForm*>(_L.get());

      aa->U1 = W; aa->U0 = W0; aa->a0 = *acc0; aa->v0 = *v0;
      aa->dt = dt; aa->theta = theta; aa->theta_m = theta_m; aa->theta_s = theta_s;
      aa->beta = beta; aa->gamma = gamma;
      aa->h = h;
      aa->x12 = x12; aa->x1c = x1c; aa->x2c = x2c;
      aa->rho = rho; aa->eta = eta; aa->K4 = K4;
      aa->kT = k; aa->cT = c; aa->hT = hT; aa->alpha = alpha;
      aa->lmbda = lambda; aa->Mob = Mob; aa->Q = Q;

      LL->U1 = W; LL->U0 = W0; LL->a0 = *acc0; LL->v0 = *v0;
      LL->dt = dt; LL->theta = theta; LL->theta_m = theta_m; LL->theta_s = theta_s;
      LL->beta = beta; LL->gamma = gamma;
      LL->h = h;
      LL->x12 = x12; LL->x1c = x1c; LL->x2c = x2c;
      LL->rho = rho; LL->eta = eta; LL->K4 = K4;
      LL->kT = k; LL->cT = c; LL->hT = hT;
      LL->T_ext = temp_ref; LL->alpha = alpha; LL->T_ref = *reference_temperature;
      LL->lmbda = lambda; LL->Mob = Mob; LL->Q = Q;
    }

    std::vector<const DirichletBC*>& bcs()
    { return _bcs; }

    const Function& displacement() const
    { return *u; }

    Function& temperature()
    { return W[1]; }

    Function& concentration()
    { return W[2]; }

    // Update
    void update()
    {
      *u = W[0];
      update_a(*acc, *u, *acc0, *v0, *u0, beta, dt);
      update_v(*acc, *v, *acc0, *v0, gamma, dt);
      *u0 = *u; *v0 = *v; *acc0 = *acc;
      W0 = W;
    }
```

```
      // Initial update
      void update_ic()
      {
        *u = W[0];
        update_a(*acc, *u, *acc0, *v0, *u0, beta, dt);
        Constant v_ic(0.0, 0.0); *v = v_ic;
        *u0 = *u; *v0 = *v; *acc0 = *acc;
        W0 = W;
      }

    private:

      // Function spaces
      boost::shared_ptr<FunctionSpace> Vu;
      boost::shared_ptr<FunctionSpace> Vu_x;
      boost::shared_ptr<FunctionSpace> Vu_y;
      boost::shared_ptr<FunctionSpace> Vt;
      boost::shared_ptr<FunctionSpace> Vc;

      // Displacement, velocity, acceleration
      boost::shared_ptr<Function> u, u0, v, v0, acc, acc0;

      // Reference temperature (at t = 0)
      boost::shared_ptr<Function> reference_temperature;

      // Boundary conditions
      DirichletBC bc_u_1;
      DirichletBC bc_u_2;
      std::vector<const DirichletBC*> _bcs;
    };

  }
}
#endif
```

ThermoMechanicalModel.h

```
// Copyright (C) 2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2009
// Last changed: 2010
//

#ifndef __HEAT_THERMOMECHANICAL_MODEL_H
#define __HEAT_THERMOMECHANICAL_MODEL_H

#include <boost/assign/list_of.hpp>
#include "Parameters.h"
#include "Model.h"
#include "Updates.h"
#include "ThermoMechanical.h"

namespace dolfin
{
  namespace heat
  {

    class ThermoMechanicalModel : public Model
    {
    public:

      // Constructor
      ThermoMechanicalModel(const Mesh& mesh)
        : Model(boost::shared_ptr<FunctionSpace>(new ThermoMechanical::FunctionSpace(
            mesh)),
          "thermo-mechanical"),
          // Supspaces
          Vu(new SubSpace(*V, 0)), Vu_x(new SubSpace(*V, 0, 0)),
          Vu_y(new SubSpace(*V, 0, 1)),
          Vt(new SubSpace(*V, 1)),
          // Displacement, velocity, acceleration
          u(new Function(Vu)), u0(new Function(Vu)),
        v(new Function(Vu)), v0(new Function(Vu)),
        acc(new Function(Vu)), acc0(new Function(Vu)),
        // Boundary conditions
        bc_u_1(*Vu_x, zero_scalar, left),
        bc_u_2(*Vu_y, zero_scalar, centre_left, "pointwise")
  {
    // Insert bc into vector
    _bcs.push_back(&bc_u_1);
    _bcs.push_back(&bc_u_2);

    // Initial conditions for displacement and temperature
    InitialConditions UU(ic_dim);
    if (UU.value_dimension(0) != 3)
      error("Wrong initial condition vector dimension. You should have 3 initial
          conditions");
    W.interpolate(UU);
    W0 = W;
    update_ic();

    // Set reference temperature to intial temperature
    reference_temperature.reset(new Function(W[1]));

    // Create forms
    _a.reset(new ThermoMechanical::BilinearForm(V, V));
    _L.reset(new ThermoMechanical::LinearForm(V));

    // Attach coefficients
    ThermoMechanical::BilinearForm* aa = dynamic_cast<ThermoMechanical::BilinearForm
        *>(_a.get());
    ThermoMechanical::LinearForm* LL = dynamic_cast<ThermoMechanical::LinearForm*>(
        _L.get());

    aa->U1 = W; aa->U0 = W0; aa->a0 = *acc0; aa->v0 = *v0;
    aa->dt = dt; aa->theta = theta; aa->theta_m = theta_m; aa->theta_s = theta_s;
    aa->beta = beta; aa->gamma = gamma;
    aa->h = h;
    aa->x12 = x12;
```

```
      aa->rho = rho; aa->eta = eta; aa->K4 = K4;
      aa->kT = k; aa->cT = c; aa->hT = hT;
      aa->alpha = alpha;


      LL->U1 = W; LL->U0 = W0; LL->v0 = *v0; LL->a0 = *acc0;
      LL->dt = dt; LL->theta = theta; LL->theta_m = theta_m; LL->theta_s = theta_s;
      LL->beta = beta; LL->gamma = gamma;
      LL->h = h;
      LL->x12 = x12;
      LL->rho = rho; LL->eta = eta; LL->K4 = K4;
      LL->kT = k; LL->cT = c; LL->hT = hT; LL->alpha = alpha;
      LL->T_ext = temp_ref; LL->T_ref = *reference_temperature;
  }

    std::vector<const DirichletBC*>& bcs()
    { return _bcs; }

    const Function& displacement() const
    { return *u; }

    Function& temperature()
    { return W[1]; }

    // Update
    void update()
    {
      *u = W[0];
      update_a(*acc, *u, *acc0, *v0, *u0, beta, dt);
      update_v(*acc, *v, *acc0, *v0, gamma, dt);
      *u0 = *u; *v0 = *v; *acc0 = *acc;
      W0 = W;
    }
    // Initial update
    void update_ic()
    {
      *u = W[0];
```

```
          update_a(*acc, *u, *acc0, *v0, *u0, beta, dt);
          Constant v_ic(0.0, 0.0); *v = v_ic;
          *u0 = *u; *v0 = *v; *acc0 = *acc;
          W0 = W;
        }

      private:

        // Function spaces
        boost::shared_ptr<FunctionSpace> Vu;
        boost::shared_ptr<FunctionSpace> Vu_x;
        boost::shared_ptr<FunctionSpace> Vu_y;
        boost::shared_ptr<FunctionSpace> Vt;

        // Displacement, velocity, acceleration
        boost::shared_ptr<Function> u, u0, v, v0, acc, acc0;

        // Reference temperature (at t = 0)
        boost::shared_ptr<Function> reference_temperature;

        // Dirichlet boundary conditions
        DirichletBC bc_u_1;
        DirichletBC bc_u_2;
        std::vector<const DirichletBC*> _bcs;
      };
  }
}
#endif
```

ChemoMechanicalModel.h

```
// Copyright (C) 2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2009
// Last changed:
//

#ifndef __HEAT_CHEMO_MECHANICAL_MODEL_H
#define __HEAT_CHEMO_MECHANICAL_MODEL_H

#include <boost/assign/list_of.hpp>
#include <dolfin/function/Function.h>
#include "Parameters.h"
#include "Model.h"
#include "Updates.h"
#include "ChemicalMechanical.h"

namespace dolfin
{
  namespace heat
  {

    // User defined nonlinear problem
    class ChemoMechanicalModel : public Model
    {
    public:

      // Constructor
      ChemoMechanicalModel(const Mesh& mesh)
          : Model(boost::shared_ptr<FunctionSpace>(new ChemicalMechanical::FunctionSpace(
                mesh)),
            "chemo-mechanical"),
            // Subspaces
            Vu(new SubSpace(*V, 0)), Vc(new SubSpace(*V, 1)),
            // Displacement, velocity, acceleration
            u(new Function(Vu)), u0(new Function(Vu)),
          v(new Function(Vu)), v0(new Function(Vu)),
          acc(new Function(Vu)), acc0(new Function(Vu)),
          // Boundary condition
          bc_u(*Vu, zero_vector, boundary)
      {
        // Insert bc into vector
        _bcs.push_back(&bc_u);

        // Initial conditions for displacement and concentration
        InitialConditions UU(ic_dim);
        if (UU.value_dimension(0) != 4)
          error("Wrong initial condition vector dimension. You should have 4 initial
              conditions");
        W.interpolate(UU);
        W0 = W;
        update_ic();

        // Create forms
        _a.reset(new ChemicalMechanical::BilinearForm(V, V));
        _L.reset(new ChemicalMechanical::LinearForm(V));

        // Attach coefficients
        ChemicalMechanical::BilinearForm* aa = dynamic_cast<ChemicalMechanical::
             BilinearForm*>(_a.get());
        ChemicalMechanical::LinearForm* LL = dynamic_cast<ChemicalMechanical::LinearForm
            *>(_L.get());

        aa->U1 = W; aa->U0 = W0;
        aa->dt = dt; aa->theta = theta; aa->theta_m = theta_m; aa->theta_s = theta_s;
        aa->beta = beta; aa->gamma = gamma;
        aa->x12 = x12; aa->x1c = x1c; aa->x2c = x2c; aa->T1 = temp_ref;
        aa->h = h;
        aa->rho = rho; aa->eta = eta; aa->K4 = K4;
        aa->Mob = Mob; aa->lmbda = lambda; aa->Q = Q;

        LL->U1 = W; LL->U0 = W0; LL->v0 = *v0; LL->a0 = *acc0;
        LL->dt = dt; LL->theta = theta; LL->theta_m = theta_m; LL->theta_s = theta_s;
      LL->beta = beta; LL->gamma = gamma;
      LL->x12 = x12; LL->x1c = x1c; LL->x2c = x2c; LL->T1 = temp_ref;
      LL->h = h;
      LL->rho = rho; LL->eta = eta; LL->K4 = K4;
      LL->Mob = Mob; LL->lmbda = lambda; LL->Q = Q;
   }

    std::vector<const DirichletBC*>& bcs()
    { return _bcs; }

    const Function& displacement() const
    { return *u; }

    Function& concentration()
    { return W[1]; }

    // Update
    void update()
    {
      *u = W[0];
      update_a(*acc, *u, *acc0, *v0, *u0, beta, dt);
      update_v(*acc, *v, *acc0, *v0, gamma, dt);
      *u0 = *u; *v0 = *v; *acc0 = *acc;
      W0 = W;
    }
    // Initial update
    void update_ic()
    {
      *u = W[0];
      update_a(*acc, *u, *acc0, *v0, *u0, beta, dt);
      Constant v_ic(2, 0.0, 0.0); *v = v_ic;
      *u0 = *u; *v0 = *v; *acc0 = *acc;
      W0 = W;
    }

  private:

      // Function spaces
      boost::shared_ptr<FunctionSpace> Vu;
      boost::shared_ptr<FunctionSpace> Vc;

      // Displacement, velocity, acceleration
      boost::shared_ptr<Function> u, u0, v, v0, acc, acc0;

      // Dirichlet boundary conditions
      DirichletBC bc_u;
      std::vector<const DirichletBC*> _bcs;

    };

  }
}
#endif
```

MechanicalModel.h

```cpp
// Copyright (C) 2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2009
// Last changed: 2010
//

#ifndef __HEAT_MECHANICAL_MODEL_H
#define __HEAT_MECHANICAL_MODEL_H

#include <vector>
#include <boost/assign/list_of.hpp>
#include "Parameters.h"
#include "Model.h"
#include "Updates.h"
#include "Mechanical.h"
#include "InitialConditions.h"

namespace dolfin
{
  namespace heat
  {

    // User defined nonlinear problem
    class MechanicalModel : public Model
    {
    public:

      // Constructor
      MechanicalModel(const Mesh& mesh)
            : Model(boost::shared_ptr<FunctionSpace>(new Mechanical::FunctionSpace(mesh)),
              "mechanical"),
              // Spaces
              Vu(V),
              // Displacement, velocity, acceleration
              u(new Function(Vu)), u0(new Function(Vu)),
        v(new Function(Vu)), v0(new Function(Vu)),
        acc(new Function(Vu)), acc0(new Function(Vu)),
        // Boundary conditions
        bc(*Vu, zero_vector, boundary)
    {
      // Insert bc into vector
      _bcs.push_back(&bc);

      // Initial conditions for displacement
      InitialConditions UU(ic_dim);
      if (UU.value_dimension(0) != 2)
        error("Wrong initial condition vector dimension. You should have 2 initial
            conditions");
      W0.interpolate(UU);
      W = W0;

      // Compute u,  v, a, u0, v0, a0
      update_ic();

      // Create forms
      _a.reset(new Mechanical::BilinearForm(V, V));
      _L.reset(new Mechanical::LinearForm(V));

      assert(_a);
      assert(_L);

      // Attach coefficients to forms
      Mechanical::BilinearForm* aa = dynamic_cast<Mechanical::BilinearForm*>(_a.get())
          ;
      Mechanical::LinearForm* LL = dynamic_cast<Mechanical::LinearForm*>(_L.get());

      aa->u1 = W; aa->u0 = *u0;
      aa->dt = dt; aa->theta_m = theta_m; aa->theta_s = theta_s;
      aa->beta = beta; aa->gamma = gamma;
      aa->rho = rho; aa->eta = eta; aa->K4 = K4;
      aa->h = h;
      aa->T1 = temp_ref; aa->x12 = x12;

      LL->u1 = W; LL->u0 = *u0; LL->v0 = *v0; LL->a0 = *acc0;
      LL->dt = dt; LL->beta = beta; LL->gamma = gamma;
      LL->theta_m = theta_m; LL->theta_s = theta_s;
      LL->rho = rho; LL->eta = eta; LL->K4 = K4;
      LL->h = h;
      LL->x12 = x12; LL->T1 = temp_ref;
   }

    std::vector<const DirichletBC*>& bcs()
    { return _bcs; }

    const Function& displacement() const
    { return *u; }

    // Update
    void update()
    {
      *u = W;
      update_a(*acc, *u, *acc0, *v0, *u0, beta, dt);
      update_v(*acc, *v, *acc0, *v0, gamma, dt);
      *u0 = *u; *v0 = *v; *acc0 = *acc;
    }
    // Initial update
    void update_ic()
    {
      *u = W[0];
      update_a(*acc, *u, *acc0, *v0, *u0, beta, dt);
      Constant v_ic(2, 0.0, 0.0); *v = v_ic;
      *u0 = *u; *v0 = *v; *acc0 = *acc;
      W0 = W;
    }

  private:

    // Function spaces
    boost::shared_ptr<FunctionSpace> Vu;
```

```

      // Displacement, velocity, acceleration
      boost::shared_ptr<Function> u, u0, v, v0, acc, acc0;

      // Boundary conditions
      DirichletBC bc;
      std::vector<const DirichletBC*> _bcs;
    };
  }
}
#endif
```

NonLinearProblem.h

```
// Copyright (C) 2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2009
// Last changed: 2010
//

#ifndef __HEAT_PROBLEM_H
#define __HEAT_PROBLEM_H

#include <dolfin/nls/NewtonSolver.h>
#include <dolfin/nls/NonlinearProblem.h>
#include <dolfin/fem/DirichletBC.h>
#include <dolfin/fem/Assembler.h>
#include <dolfin/fem/SystemAssembler.h>

namespace dolfin
{
namespace heat
{

// User defined nonlinear problem
 class NonlinearProblem : public dolfin::NonlinearProblem
{
  public:

    // Constructor
    NonlinearProblem(const Form& a, const Form& L, std::vector<const DirichletBC*>& bcs,
                     const NewtonSolver& solver) : a(a), L(L), bcs(bcs),
                     solver(solver), reset_tensor(true)
    {
    }

    // Residual vector
    void F(GenericVector& b, const GenericVector& x)
    {
      // Assemble and modify for Dirichlet boundary conditions
      dolfin::Assembler::assemble(b, L);
      for (dolfin::uint i = 0; i < bcs.size(); i++)
        bcs[i]->apply(b, x);
    }

    // Jacobian
    void J(dolfin::GenericMatrix& A, const dolfin::GenericVector& x)
    {
      // Assemble and modify for Dirichlet boundary conditions
      if(solver.iteration() < 1)
      {
        dolfin::Assembler::assemble(A, a, reset_tensor);
        for (dolfin::uint i = 0; i < bcs.size(); i++)
          bcs[i]->apply(A);
      }
      reset_tensor  = false;

    }

  private:

    const Form& a;
    const Form& L;
    std::vector<const DirichletBC*>& bcs;
    const NewtonSolver& solver;
    bool reset_tensor;
};

}
}
#endif
```

main.cpp

```
// Copyright (C) 2008-2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2008-07-11
// Last changed: 2010
//

#include "ThermoMechanicalModel.h"
#include "MechanicalModel.h"
#include "ChemoMechanicalModel.h"
#include "ThermoChemoMechanicalModel.h"
#include "NonLinearProblem.h"
#include <dolfin.h>

using namespace dolfin;
using namespace heat;

//-----------------------------------------------------------------------------
void solver(heat::Model& model, Mesh& mesh)
{
  // Seed random number generator
  dolfin::seed(3);

  // Stretch mesh
  MeshGeometry& mesh_geometry = mesh.geometry();
  for (VertexIterator v(mesh); !v.end(); ++v)
  {
    double* x = mesh_geometry.x(v->index());
    x[0] *= mesh_stretch;
    x[1] *= mesh_stretch;
  }

  // Set parameters for linear solver
  linear_solver.parameters["monitor_convergence"] = linearsolv_conv;
  linear_solver.parameters["relative_tolerance"] = linearsolv_reltol;

```

```
// Set parameters for newton solver
newton_solver.parameters["maximum_iterations"]    = newton_maxit;
newton_solver.parameters["relative_tolerance"]    = newton_reltol;
newton_solver.parameters["absolute_tolerance"]    = newton_abstol;

// Create nonlinear problem
heat::NonlinearProblem nonlinear_problem(model.a(), model.L(), model.bcs(),
    newton_solver);

// Save initial condition to file
File file_u("results/u.pvd", "compressed");
File file_T("results/T.pvd", "compressed");
File file_c("results/c.pvd", "compressed");
File file_e2("results/e2.pvd", "compressed");

file_u  << model.displacement();
if (model.str() == "thermo-mechanical" || model.str() == "thermo-chemo-mechanical")
  file_T  << model.temperature();
if (model.str() == "chemo-mechanical" || model.str() == "thermo-chemo-mechanical")
  file_c  << model.concentration();
file_e2 << model.e2();

// Start time stepping
dolfin::uint step = 1;
while( t < T)
{
  t += dt;

  // Solve
  cout << "Model: " << model.str() << endl;
  cout << "Step, time: " << step << ", " << t << endl;
  tic();
  newton_solver.solve(nonlinear_problem, model.vector());
  cout << "Newton step required " << toc() << " seconds." << endl;

  // Update model
  model.update();
```

```

    // Save functions to file
    if( step % save_step == 0)
    {
      file_u  << model.displacement();
      if (model.str() == "thermo-mechanical" || model.str() == "thermo-chemo-mechanical"
          )
        file_T  << model.temperature();
      if (model.str() == "chemo-mechanical" || model.str() == "thermo-chemo-mechanical")
        file_c  << model.concentration();
      file_e2 << model.e2();
    }
    ++step;
  }

  // Save solution vector
  File file_solution("U_solution.xml");
  file_solution << model.vector();
}
//-----------------------------------------------------------------------------

int main(void)
{
  parameters["linear_algebra_backend"] = "PETSc";

  // Initialise mesh facets
  mesh.init(1);

  // Create model
  if (model_type == "Mechanical")
  {heat::MechanicalModel model(mesh);               solver(model, mesh);}
  if (model_type == "ThermoMechanical")
  {heat::ThermoMechanicalModel model(mesh);         solver(model, mesh);}
  if (model_type == "ChemoMechanical")
  {heat::ChemoMechanicalModel model(mesh);          solver(model, mesh);}
  if (model_type == "ThermoChemoMechanical")
  {heat::ThermoChemoMechanicalModel model(mesh);  solver(model, mesh);}
  else
  {error("Unknown model");}
}
```

Parameters.h

```
// Copyright (C) 2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2009
// Last changed: 2010
//

#ifndef __HEAT_PARAMETERS_H
#define __HEAT_PARAMETERS_H

#include <boost/assign/list_of.hpp>
#include <dolfin/function/Constant.h>
#include "InitialConditions.h"
#include "BoundaryConditions.h"
#include <dolfin.h>

namespace dolfin
{
  namespace heat
  {

  // Specify model type
  std::string model_type("ThermoChemoMechanical");

  // Create linear solver and set parameters
  KrylovSolver linear_solver("bicgstab", "sor");
  static bool   linearsolv_conv(true);
  static double linearsolv_reltol = 1.0e-4;

  // Create nonlinear solver and set parameters
  DefaultFactory factory;
  NewtonSolver newton_solver(linear_solver, factory);
  static int    newton_maxit  = 10;
  static double newton_reltol = 1e-6;
  static double newton_abstol = 1e-12;
//-----------------------------------------------------------------------------
```

```
  // Mesh
  UnitSquare mesh(128, 128, "crossed");
  static double mesh_stretch = 1.0;    // mesh stretching parameter

  // Time stepping parameters
  static Constant dt(2.5e-6);          // time step
  static double t = 0.0;               // initial simulation time
  static double T = 10000*dt;          // total simulation time
  static int save_step = 2;            // save results every # steps
  static Constant theta(0.5);          // evaluation point
  static Constant theta_m(-1.0);       // evaluation point g-alpha
  static Constant theta_s(0.0);        // evaluation point g-alpha
  static Constant beta(1.0);           // acceleration weight
  static Constant gamma(1.5);          // velocity weight

  // Model parameters
  static Constant c(1.35e-1);          // heat capacity
  static Constant k(1.784e-3);         // heat diffusivity
  static Constant hT(8.1e-5);          // heat flux coefficient
  static Constant temp_ref(0.4);       // temperature
  static Constant alpha(1.17e-2);      // thermal dilatation coefficient
  static Constant rho(5.0e-7);         // density
  static Constant eta(5.0e-8);         // damping coefficient
  static Constant Mob(1.24e6)   ;      // chemical mobility
  static Constant Q(5.0);              // activation energy
  static Constant K4(1.0e-7);          // displacive interface energy term
  static Constant lambda(1.0e-8);      // diffusive interface energy term
  static Constant x12(1.1);            // coupling between e1 and e2
  static Constant x1c(0.0);            // coupling between c and e1
  static Constant x2c(5.0);            // chemo-mechanical coupling

//------------------------------------------------------------------------
  uint ic_dim = 5;                     // number of initial conditions

//------------------------------------------------------------------------
  // Boundary conditions
  std::vector<double> _zero_vector = boost::assign::list_of(0)(0);
  static Constant zero_vector(0.0, 0.0);
  static Constant zero_scalar(0.0);
  static Left left;
  static CentreLeft centre_left;

  }
}
#endif
```

A. DOLFIN/FFC CODE

InitialConditions.h

```
// Copyright (C) 2008-2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2008-07-11
// Last changed: 2010
//

#ifndef __HEAT_INITIAL_CONDITIONS_H
#define __HEAT_INITIAL_CONDITIONS_H

#include <dolfin/function/Expression.h>
#include <tr1/random>

namespace dolfin
{
namespace heat
{
// Initial conditions
class InitialConditions : public Expression
{
public:

  InitialConditions(uint ic_dim)
       : Expression(ic_dim),  engine(2), distribution(-0.01, 0.01),
         rng(engine, distribution)
  {
  }

  void eval(Array<double>& values, const Array<const double>& x) const
  {
    values[0]= 0.0001*rng();  // x displacement
    values[1]= 0.0001*rng();  // y displacement
    values[2]= 0.4;           // temperature
    values[3]= 0.001*rng();   // chemical concentration
    values[4]= 0.0;           // chemical potential
  }

  private:

    std::tr1::mt19937 engine;
    std::tr1::uniform_real<double> distribution;
    mutable std::tr1::variate_generator<std::tr1::mt19937, std::tr1::uniform_real<double
        > > rng;
};

}
}
#endif
```

A. DOLFIN/FFC CODE

BoundaryConditions.h

```
// Copyright (C) 2008-2009 Mirko Maraldi and Garth N. Wells.
// Licensed under the GNU LGPL Version 2.1.
//
// First added:  2008-07-11
// Last changed: 2010
//

#ifndef __HEAT_BOUNDARY_CONDITIONS_H
#define __HEAT_BOUNDARY_CONDITIONS_H

#include <dolfin/function/Function.h>
#include <dolfin/mesh/SubDomain.h>
#include <dolfin.h>

using namespace dolfin;

// Boundary
class Boundary : public SubDomain
{
  bool inside(const double* x, bool on_boundary) const
  {
  return on_boundary;
  }
};

class Left : public SubDomain
{
  bool inside(const double* x, bool on_boundary) const
  {
  if (x[0] < DOLFIN_EPS)
    return true;
  else
    return false;
  }
};

```

```
class CentreLeft : public SubDomain
{
  bool inside(const double* x, bool on_boundary) const
  {
  if ( (x[0] < DOLFIN_EPS) && ((x[1]-0.51) < DOLFIN_EPS) && ((0.49-x[1]) < DOLFIN_EPS))
    return true;
  else
    return false;
  }
};

class Bottom : public SubDomain
{
  bool inside(const double* x, bool on_boundary) const
  {
  if (x[1] < DOLFIN_EPS)
    return true;
  else
    return false;
  }
};

#endif
```

Appendix B

Experimental Characterisation Of An Eutectoid Steel

B.1 Introduction

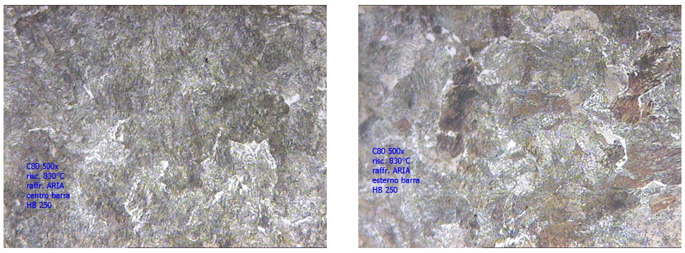
A step forward for the work presented in this thesis would be the development of a macroscopic scale model; in order to proceed in this direction, an experimental campaign is envisioned for:

- evaluate the importance (at the macroscopic scale) of the mechanical phenomena for a displacive phase transition;
- estimate the value of some of the model parameters.

For these reasons, an experimental characterisation carried out on an eutectoid steel is presented in the following; the work contained here has been done in collaboration with *DEMM officine meccaniche Spa*. The specimens were of cylindrical shape, with a length of 60 mm and a diameter of 30 mm.

Element	C	Mn	Si	Cr	Ni	Mo	P	S	Cu	W	Fe
Quantity [%]	0.78	0.20	0.26	0.15	0.08	0.01	0.02	0.01	0.02	0.02	98.45

Table B.1: Chemical composition of the analysed eutectoid steel.



(a) (b)

Figure B.1: Heating temperature: 830° C, cooling: air; a) position: centre; b) position surface.

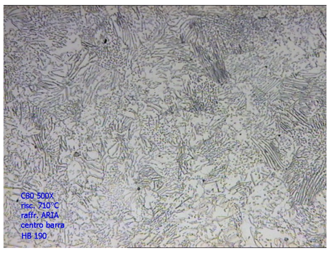
B.2 Chemical Analysis

A chemical analysis has been carried out, to determine the element percentages for the material under consideration. The results are presented in table B.1.

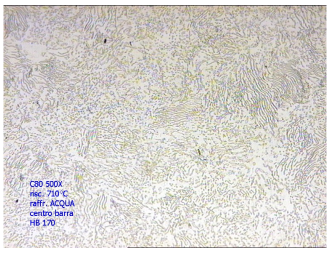
B.3 Cooling Test

The specimens were heated to different temperatures, in the range of 710° C to 830° C, in 20° C increments. For each temperature, two specimens were used; one cooled in air, the other one in water.

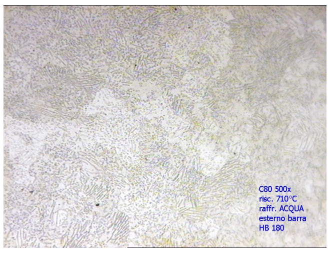
To conduct a characterisation of the different microstructures resulting from the different heat treatments, the specimens were prepared for optical microscope observation after cooling. For the water-cooled specimens, microstructure was observed both at the centre and at the surface. For the air-cooled specimens only the centre was observed, as no microstructural difference was found at other locations (see fig. B.1). Images from B.2 to B.4 show the resulting observations.



(a)

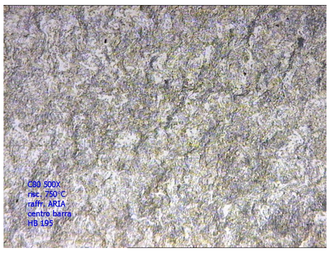


(b)

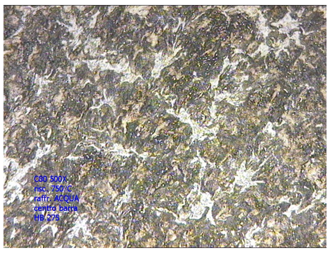


(c)

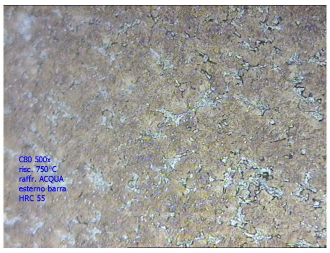
Figure B.2: Heating temperature: 710° C; a) cooling: air, position: centre; b) cooling: water, position: centre; c) cooling: water, position: surface.



(a)

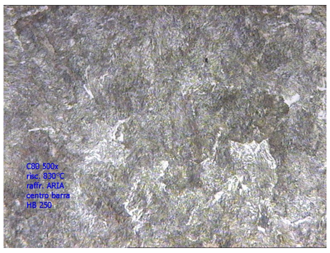


(b)

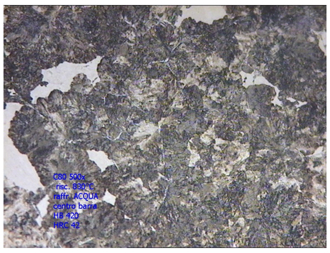


(c)

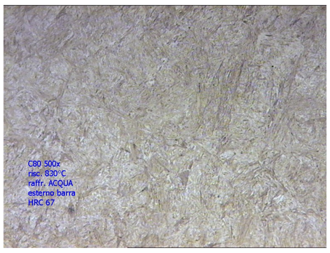
Figure B.3: Heating temperature: 750° C; a) cooling: air, position: centre; b) cooling: water, position: centre; c) cooling: water, position: surface.



(a)



(b)



(c)

Figure B.4: Heating temperature: 830° C; a) cooling: air, position: centre; b) cooling: water, position: centre; c) cooling: water, position: surface.

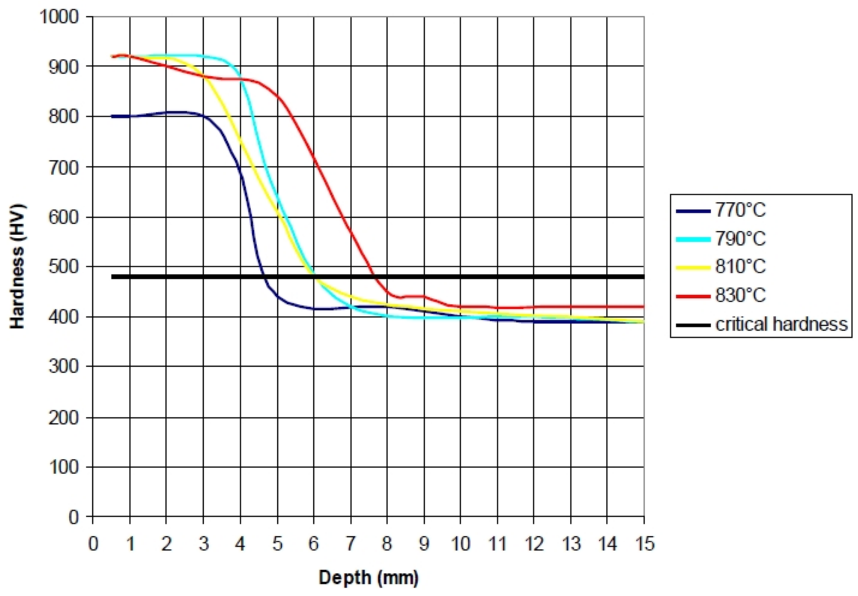


Figure B.5: *U* curves for the analysed material.

In table B.2 the microstructure obtained, as well as the hardness measurements for all the tests, are listed: It can be seen that no Martensite can be obtained by cooling the specimens in air. A martensitic structure is obtained when the specimens' heating temperature is above the threshold value, and the cooling rate is high (hence, possible in water).

B.4 *U* curves

U curves have been estimated from hardness measurements of the specimens, and are depicted in fig. B.5. It can be seen that increasing heating temperature over a threshold will not result in an increase in the maximum hardness, as the limit for an alloy of such a chemical composition is reached.

Critical depth has been estimated for the various test to which the above figure refers; this is the depth at which 50% of Martensite can be observed (set at a hardness of 480 HV). Results are listed in table B.3.

As expected, increasing the heating temperature results in a grater critical depth

Temperature [$^\circ C$]	Cooling	Location	Microstructure	Hardness
710	air	centre	Pearlite	190 [HB]
710	air	centre	Pearlite	170 [HB]
710	air	surface	Pearlite	180 [HB]
730	air	centre	Pearlite	190 [HB]
730	air	centre	Pearlite + Martensite	270 [HB]
730	air	surface	Martensite + Pearlite	56 [HRC]
750	air	centre	Pearlite	195 [HB]
750	air	centre	Pearlite + Martensite	275 [HB]
750	air	surface	Martensite + Pearlite	55 [HRC]
770	air	centre	Pearlite	225 [HB]
770	air	centre	Pearlite + Martensite	35 [HRC]
770	air	surface	Martensite	64 [HRC]
790	air	centre	Pearlite	230 [HB]
790	air	centre	Pearlite + Martensite	37 [HRC]
790	air	surface	Martensite	67 [HRC]
810	air	centre	Pearlite	240 [HB]
810	air	centre	Pearlite + Martensite	39 [HRC]
810	air	surface	Martensite	67 [HRC]
830	air	centre	Pearlite	250 [HB]
830	air	centre	Pearlite + Martensite	42 [HRC]
830	air	surface	Martensite	67 [HRC]

Table B.2: Microstructures.

Heating temperature [$°C$]	Critical depth [mm]
770	4.8
790	6.5
810	6.5
830	7.5

Table B.3: Critical depth evaluation for different heating temperature.

(despite the maximum hardness not increasing anymore, as stated before).

References

Ahluwalia, R., Lookman, T., Saxena, A., 2006. Dynamic strain loading of cubic to tetragonal martensites. Acta Materialia 54, 2109–2120.

Albers, R. C., Ahluwalia, R., Lookman, T., Saxena, A., 2004. Modeling solid-solid phase transformations: from single crystal to polycrystal behaviour. Computational and Applied Mathematics 23, 345–361.

Alnæs, M. S., Logg, A., Mardal, K.-A., Skavhaug, O., Langtangen, H., 2009. Unified framework for finite element assembly. International Journal of Computational Science and Engineering 4 (4), 231–244.

Alt, H. W., Pawlow, I., 1992. A mathematical model of dynamics of non-isothermal phase separation. Physica D 59, 389–416.

Artemev, A., Jin, Y., Khachaturyan, A., 2001. Three-dimensional phase field model of proper martensitic transformation. Acta Materialia 49 (7), 1165–1177.

Barsch, G., Krumhansl, J., 1984. Twin boundaries in ferroelastic media without interface dislocations. Physical Review Letters 53, 1070–1072.

Bergheau, J. M., Duranton, P., Porzner, H., Boitout, F., 2000. Coupled carbon diffusion, metallurgical transformation and heat transfer models applied to the case-hardening process. Proceedings of EUROPAM 2000, Nanates.

Berti, V., Fabrizio, M., Grandi, D., in press. Phase transitions in shape memory alloys: a non-isothermal ginzburg-landau model. Physica D.

Bhadeshia, H. K. D. H., 1987. Worked examples in the geometry of crystals. The Institute of Metals, London.

Bouville, M., Ahluwalia, R., 2007. Interplay between diffusive and displacive phase transformations: time-temperature-transformation diagrams and microstructures. Physical Review B 75, 054110.

Brokate, M., Sprekels, J., 1996. Hysteresis and phase transformation. Springer and Verlag, New York.

Cahn, J. W., 1961. On spinodal decomposition. Acta Metallurgica 9, 795–801.

Chung, J., Hulbert, G. M., 1993. A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized-alpha method. Journal of Applied Mechanics 60, 371–375.

Del Piero, G., 2007. A new class of fit regions. Note di Matematica 27 (2), 55–67.

Del Piero, G., 2009. On the method of virtual powers in continuum mechanics. Journal of Mechanics of Materials and Structures 4, 281–292.

Dembo, R. S., Eisenstat, S. C., Steihaug, T., 1982. Inexact newton methods. SIAM Journal on Numerical Analysis 19 (2), 400–408.

Erlicher, S., Bonaventura, L., Bursi, O. S., 2002. The analysis of the generalized-alpha method for non-linear dynamic problems. Computational Mechanics 28, 83–104.

Falk, F., 1980. Model free energy, mechanics and thermodynamics of shape memory alloys. Acta Metallurgica 28, 1773–1780.

Fried, E., Gurtin, M. E., 1993. Continuum theory of thermally induced phase transitions based on an order parameter. Physica D 68, R 326–343.

Ginzburg, V. L., Landau, L. D., 1950. On the theory of superconductivity. Zh. Eksperim. i Teor. Fiz. 20, 1064.

Gurtin, M. E., 1996. Generalized ginzburg-landau and cahn-hilliard equations based on a microforce balance. Physica D 92, 178–192.

Herceg, D., Krejić, N., Lužanin, Z., 1996. Quasi-newton method with correction. Novi Sad Journal of Mathematics 26 (1), 115–127.

Hömberg, D., 1995. A mathematical model for the phase transitions in eutectoid carbon steel. IMA Journal of Applied Mathematics 54, 31–57.

Kirby, R. C., 2004. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. ACM Transactions on Mathematical Software 30 (4), 502–516.

Kurdjumov, G., Kaminsky, E., 1928. X-ray studies of the structure of quenched carbon steel. Nature 122, 475.

Lemaitre, J., Chaboche, J. L., 1990. Mechanics of solid materials. Cambridge University Press, Cambridge.

Levitas, V. I., 2000. Thermomechanical and kinetic approaches to diffusional-displacive phase transitions in inelastic materials. Mechanics Research Communications 27, 217–227.

Levitas, V. I., Preston, D. L., 2002. Three-dimensional landau theory for multivariant stress-induced martensitic phase transformations i. austenite↔martensite. Physical Review B 66, 134206–1–9.

Lieberman, D. S., Wechsler, M. S., Read, T. A., 1955. Cubic to orthorombic diffusionless phase change - experimental and theoretical studies of aucd. Journal of Applied Physics 26, 473–484.

Logg, A., Wells, G. N., 2010. Dolfin: automated finite element computing. ACM Transactions on Mathematical Software 37 (2).

Malvern, L. E., 1969. Introduction to the mechanics of a continuous medium. Prentice-Hall, Englewood Cliffs.

Matteoli, L., 1990. Il diagramma di stato ferro-carbonio e le curve TTT. Guida teorico-pratica ai trattamenti termici fondamentali degli acciai. Associazione Italiana di Metallurgia, Milano.

Maugin, G. A., 1990. Internal variables and dissipative structures. Journal of Non-Equilibrium Thermodynamics 15, 173–192.

Maugin, G. A., 1992. The thermomechanics of plasticity and fracture. Cambridge University Press, Cambridge.

Maugin, G. A., 1998. The thermomechanics of nonlinear irreversible behaviours: an introduction. World Scientific Publishing Company, Singapore.

Mehl, J., 1972. ASM metals handbook - atlas of microstructures of industrial alloys (Vol.7). American Society for Metals, Material Park, OH.

Ølgaard, K. B., Logg, A., Wells, G. N., 2008. Automated code generation for discontinuous galerkin methods. SIAM Journal on Scientific Computing 31 (2), 849–864.

Onuki, A., Furukawa, A., 2001. Phase transitions of binary alloys with elastic inhomogeneity. Physical Review Letters 83, 452–455.

Ortega, J. M., Rheinboldt, W. C., 1970. Iterative solution of nonlinear equations of several variables. Academic Press, New York.

Polizzotto, C., 2003. Gradient elasticity and nonstandard boundary conditions. International Journal of Solids and Structures 40, 7399–7423.

Rao, M., Sengupta, S., 2003. Nucleation of solids in solids: ferrites and martensites. Physical Review Letters 91, 045502–1–3.

Shenoy, S. R., Lookman, T., Saxena, A., Bishop, A. R., 1999. Martensitic textures: multiscale consequences of elastic compatibility. Physical Review B 60 (R 12), 537–541.

Tolédano, J. C., Tolédano, P., 1987. The Landau Theory of the Phase Transitions. World Scientific Editor, Singapore.

Wang, Y., Khachaturyan, A. G., 1997. Three dimensional field model and computer modeling of martensitic transformations. Acta Materialia 45, 759–773.

Xiao, L., Fan, Z., Jinxiu, Z., 1995. Lattice-parameter variation with carbon content of martensite. i. x-ray-diffraction experimental study. Physical Review B 52 (14), 9970–9978.

Yano, K., Horie, Y., 2002. Mesomechanics of the $\alpha - \varepsilon$ transition in iron. International Journal of Plasticity 18, 1427–1446.